ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και Υλοποίηση Ενσωματωμένων Συστημάτων με Προγραμματιζόμενες Ψηφίδες Xilinx

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιορδάνης Γ. Ασλανίδης

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και Υλοποίηση Ενσωματωμένων Συστημάτων με Προγραμματιζόμενες Ψηφίδες Xilinx

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιορδάνης Γ. Ασλανίδης

Επιβλέπων : Κιαμάλ Ζ. Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Ιουλίου 2005.

.....	.....	.....
Κιαμάλ Πεκμεστζή	Παναγιώτης Τσανάκας	Ηλίας Κουκούτσης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Κ Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

.....
Ιορδάνης Γ. Ασλανίδης

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

Copyright © Ιορδάνης Ασλανίδης, 2005.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

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

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

Περίληψη

Αντικείμενο της διπλωματικής εργασίας αποτελεί η σχεδίαση και η υλοποίηση, απλών ενσωματωμένων συστημάτων με την βοήθεια της ομάδας εργαλείων του Embedded Development Kit (EDK), της Xilinx. Η παρουσίαση των συστημάτων έχει χωριστεί σε δύο εφαρμογές. Στην πρώτη γίνεται σχεδιασμός βήμα-βήμα ενός απλού συστήματος που περιέχει πλήθος περιφερειακών, χρονιστή, που εξυπηρετεί διακοπές, και το οποίο υποστηρίζει την εκσφαλμάτωση υλικού και λογισμικού, ενώ στην δεύτερη περιγράφεται η δημιουργία ενός φίλτρου που επικοινωνεί με προσωπικό υπολογιστή μέσω σειριακής επικοινωνίας. Εκτός από το EDK, χρησιμοποιήθηκαν, από την ίδια εταιρία, το Integrated Software Environment (ISE) και το ChipScope Pro. Τέλος για τις ανάγκες της παρουσίασης της διπλωματικής έγινε χρήση του MATLAB της MathWorks για την εύρεση των φίλτρων που χρησιμοποιήθηκαν και του Labview της National Instruments για την παρουσίαση των αποτελεσμάτων της λειτουργίας των φίλτρων. Τα συστήματα που θα παρουσιαστούν σε αυτό το σύγγραμμα, υλοποιήθηκαν σε τρεις κάρτες, την Digilab 2, μαζί με την κάρτα επέκτασης DIO2, της Digilent, που χρησιμοποιεί το FPGA Spartan-II της Xilinx, και την Spartan-3 Starter Kit της Xilinx, που φέρει το FPGA Spartan-3, της Xilinx επίσης. Ο μικροεπεξεργαστής που υλοποιήθηκε και χρησιμοποιήθηκε είναι ο MicroBlaze, τα πνευματικά δικαιώματα του οποίου ανήκουν στην Xilinx.

Η διπλωματική εργασία χωρίζεται σε πέντε κεφάλαια. Στο πρώτο κεφάλαιο αναφέρονται επιγραμματικά κάποια στοιχεία για τα ενσωματωμένα συστήματα. Στο δεύτερο κεφάλαιο βρίσκεται μια περιγραφή των κυριότερων στοιχείων του επεξεργαστή MicroBlaze. Στο κεφάλαιο τρία παρουσιάζονται οι τρεις κάρτες που χρησιμοποιήθηκαν για την υλοποίηση των ενσωματωμένων συστημάτων, τα οποία σχεδιάστηκαν και συντέθηκαν με την βοήθεια των εργαλείων του EDK, τα κυριότερα από τα οποία παρουσιάζονται στο κεφάλαιο τέσσερα. Τέλος, στο πέμπτο κεφάλαιο αναλύονται τα βήματα που ακολουθήθηκαν για τον σχεδιασμό και την υλοποίηση των συστημάτων, που όπως έχει ήδη αναφερθεί χωρίζονται σε δύο εφαρμογές.

Λέξεις Κλειδιά

Ενσωματωμένα συστήματα, MicroBlaze, EDK, Digilab 2, DIO2, Spartan-II, Spartan-3, Ψηφιακό φίλτρο.

Abstract

The purpose of the present diploma thesis is to demonstrate the procedure of planning and implementing a simple embedded system, using the set of tools included in the Embedded Development Kit (EDK) of Xilinx. The presentation has been split in two parts. In the first part, one can follow the procedure of planning a system which contains various peripherals, timer, buttons, leds, 7-segment display, which handles interrupts and supports software and hardware debugging. The second part involves the implementation of a filter which communicates with a PC through a serial port. Besides EDK, Integrated Software Environment (ISE) and ChipScope Pro, both of Xilinx, were used, while MATLAB of MathWorks provided us with the filters that were implemented and Labview of National Instruments showed us in graphics the results of the use of the filters. The systems that are described in this diploma thesis were implemented on three boards, Digilab 2 of Digilent, containing the Spartan-II FPGA of Xilinx, along with the expansion board DIO2 of Digilent, and the Spartan-3 Starter Kit board of Xilinx, containing the Spartan-3 FPGA. The microprocessor that was implemented is MicroBlaze which belongs to Xilinx.

The diploma thesis comprises five chapters. Chapter one refers to the embedded systems generally. Chapter two contains a short analysis of MicroBlaze and its features. In chapter three there is the outline of the three boards where the systems were implemented. Chapter four attempts a presentation of the basic tools of EDK, and finally chapter five presents the steps for planning and implementing the embedded systems that were mentioned above.

KeyWords

Embedded systems, MicroBlaze, EDK, Digilab 2, DIO2, Spartan-II, Spartan-3, Digital filter.

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω κατά κύριο λόγο τον καθηγητή κ. Κ. Ζ. Πεκμεστζή και τον λέκτορα κ. Γ. Οικονομάκο, για τις συμβουλές και τις ιδέες τους σε όλη την διάρκεια της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Μικροϋπολογιστών και Ψηφιακών Συστημάτων, και ιδιαίτερα τους Ι. Σιδερή και Κ. Αναγνωστόπουλο, των οποίων η συμβολή υπήρξε καθοριστική για την ολοκλήρωση της εργασίας. Τέλος θεωρώ χρέος μου να ευχαριστήσω την οικογένειά μου και τους φίλους μου για την ψυχολογική αλλά και υλική υποστήριξη τους και κυρίως την Ευαγγελία Κουρτέση που με βοήθησε στο να μετατρέψω το παρόν σύγγραμμα στην τωρινή του μορφή και στην προετοιμασία της παρουσίασης της διπλωματικής, καθώς και τον Μπάμπη Λυμπερόπουλο για την συνδρομή του στην εκτύπωση απαραίτητων βοηθημάτων σχετικών με την εργασία.

Πίνακας Περιεχομένων

Ευρετήριο Σχημάτων

Κεφάλαιο 1ο

Κεφάλαιο 2ο

Κεφάλαιο 3ο

Κεφάλαιο 4ο

Κεφάλαιο 5ο

Ευρετήριο Πινάκων

Κεφάλαιο 2ο

Κεφάλαιο 3ο

Κεφάλαιο 4^{ο}

Κεφάλαιο 5^{ο}

1. Ενσωματωμένα Συστήματα

1.1 Γενικά

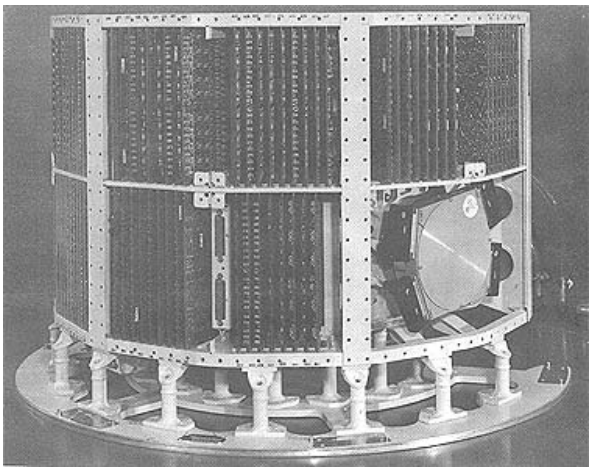
Ένα ενσωματωμένο σύστημα (embedded system), είναι ένα υπολογιστικό σύστημα ειδικού σκοπού, που είναι πολλές φορές ενσωματωμένο μέσα στην συσκευή που ελέγχει. Ένα ενσωματωμένο σύστημα έχει ειδικές απαιτήσεις και εκτελεί προκαθορισμένες λειτουργίες, σε αντίθεση με έναν γενικού σκοπού προσωπικό υπολογιστή.

1.2 Ιστορικά στοιχεία

Το πρώτο σύγχρονο και αναγνωρίσιμο ενσωματωμένο σύστημα ήταν το Apollo Guidance Computer, που αναπτύχθηκε από τον Charles Stark Draper σε εργαστήριο του Massachusetts Institute of Technology (MIT). Κάθε πτήση στο φεγγάρι χρησιμοποιούσε δύο από αυτά, στο σύστημα αδρανειακής καθοδήγησης. Στην έναρξη του διαστημικού προγράμματος Apollo το συγκεκριμένο υπολογιστικό σύστημα θεωρήθηκε ως το πιο ριψοκίνδυνο μέρος του προγράμματος. Η χρήση των νέων, ακόμα, ολοκληρωμένων κυκλωμάτων, προκειμένου να μειωθούν το μέγεθος και το βάρος, καθιστούσαν το ρίσκο μεγαλύτερο.

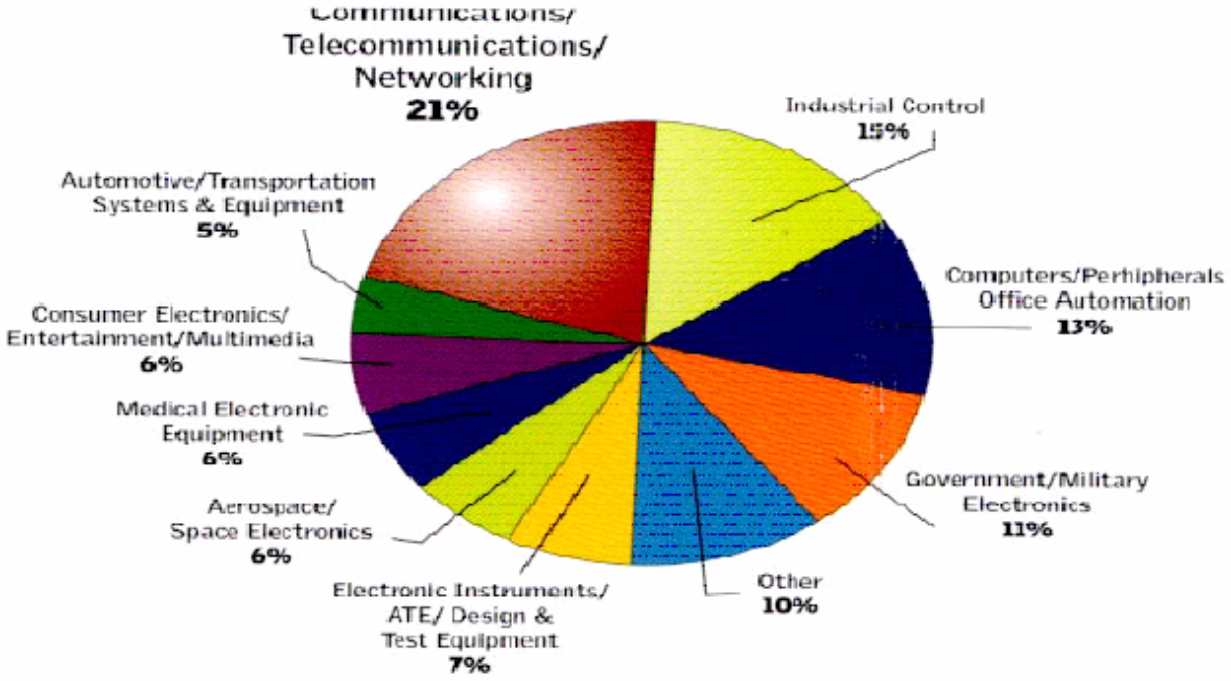
Σχήμα 1.1 – Apollo Guidance Computer

Το πρώτο ενσωματωμένο σύστημα μαζικής παραγωγής, ήταν ο υπολογιστής κατεύθυνσης για τον πύραυλο Minuteman το 1961. Το σύστημα ονομαζόταν Autonetics D-17 και χρησιμοποιούσε διακριτή λογική τρανζίστορ και έναν σκληρό δίσκο σαν κύρια μνήμη. Όταν το 1966 ξεκίνησε η παραγωγή του Minuteman II, το D-17 αντικαταστάθηκε από ένα νέο υπολογιστικό σύστημα, του οποίου ήταν ο κύριος χρήστης. Δίχως αυτό το πρόγραμμα ίσως τα ολοκληρωμένα κυκλώματα να μην είχαν φτάσει σε τόσο προσιτές τιμές. Το σημαντικότερο σχεδιαστικό χαρακτηριστικό του υπολογιστή του Minuteman ήταν ότι ο αλγόριθμος καθοδήγησής του άλλαζε όσο εκτελούταν το πρόγραμμα, προκειμένου ο πύραυλος να έχει μεγαλύτερη ακρίβεια, ενώ μπορούσε και να δοκιμάσει τον πύραυλο γλιτώνοντας έτσι τις επιπλέον συνδέσεις.



Σχήμα 1.2 – Autonetics D-17

1.3 Παραδείγματα



Σχήμα 1.3 – Τα ποσοστά επί του συνόλου των ενσωματωμένων συστημάτων που χρησιμοποιήθηκαν σε διαφόρους τομείς το 1999

Μερικές από τις συσκευές στις οποίες χρησιμοποιούνται ενσωματωμένα συστήματα είναι οι παρακάτω:

- Συσκευές αυτόματης συναλλαγής (ATMs)
- Κινητά τηλέφωνα και τηλεφωνικοί μεταγωγείς
- Κινητά τηλέφωνα με επιπλέον δυνατότητες, προσωπικούς ψηφιακούς οδηγούς (PDAs) υποστήριξη Java
- Εξοπλισμός δικτύων υπολογιστών, όπως δρομολογητές, timeservers και τείχη προστασίας
- Εκτυπωτές υπολογιστών
- Πολυλειτουργικοί εκτυπωτές (MFPs)

- Αντιγραφικά
- Οδηγοί δίσκων (δισκέτες και σκληροί δίσκοι)
- Ελεγκτές μηχανών και ελεγκτές κατά του μπλοκαρίσματος των τροχών για τα φρένα των αυτοκινήτων
- Προϊόντα αυτοματισμού για το σπίτι, όπως θερμοστάτες, κλιματιστικά, ψεκαστήρες και συστήματα ασφαλείας
- Οικιακές συσκευές, όπως φούρνοι μικροκυμάτων, πλυντήρια, σύνθετα τηλεόρασης και μηχανές παιξίματος / εγγραφής ψηφιακών δίσκων
- Συστήματα αδρανειακής καθοδήγησης, υλικό / λογισμικό ελέγχου πτήσης και άλλα ολοκληρωμένα συστήματα στην αεροπλοΐα και την πυραυλική τεχνολογία
- Ιατρικός εξοπλισμός
- Εξοπλισμός μετρήσεων, όπως παλμογράφοι, αναλυτές κυκλωμάτων και αναλυτές φάσματος
- Υπολογιστές τσέπης
- Πολυλειτουργικά ρολόγια χειρός
- Σταθερές και φορητές παιχνιδομηχανές
- Προσωπικοί ψηφιακοί οδηγοί (PDAs)
- Προγραμματιζόμενοι ελεγκτές (PLCs) για βιομηχανικό αυτοματισμό και παρακολούθηση

1.4 Χαρακτηριστικά

Τα ενσωματωμένα συστήματα συνήθως σχεδιάζονται ώστε να εκτελούν επιλεγμένες λειτουργίες με χαμηλό κόστος. Το σύστημα μπορεί να χρειαστεί να είναι πολύ γρήγορο για κάποιες λειτουργίες, αλλά για τις περισσότερες από τις υπόλοιπες πιθανότατα κάτι τέτοιο δεν θα χρειάζεται. Με αυτό το σκεπτικό, πολλά μέρη ενός ενσωματωμένου συστήματος θα έχουν μειωμένη απόδοση. Η βραδύτητα δεν σχετίζεται μονάχα με την ταχύτητα του ρολογιού. Όλη η αρχιτεκτονική ενός ενσωματωμένου συστήματος είναι συχνά σκόπιμα απλουστευμένη, με στόχο το μικρότερο κόστος, σε σχέση με το υλικό των υπολογιστών γενικού σκοπού. Για παράδειγμα, τα ενσωματωμένα συστήματα χρησιμοποιούν συχνά περιφερειακά που ελέγχονται από σειριακές διεπαφές, οι οποίες είναι δεκάδες έως εκατοντάδες φορές πιο αργές σε σύγκριση με τα περιφερειακά που χρησιμοποιούνται στους προσωπικούς υπολογιστές. Εφόσον πολλά ενσωματωμένα συστήματα παράγονται μαζικά κατά εκατομμύρια, η μείωση του κόστους είναι ένα κυρίαρχο ζήτημα.. Μερικά ενσωματωμένα συστήματα δεν απαιτούν μεγάλη υπολογιστική ισχύ ή πόρους και αυτό επιτρέπει την ελαχιστοποίηση του κόστους παραγωγής χρησιμοποιώντας έναν, σχετικά, αργό επεξεργαστή και μικρό μέγεθος μνήμης.

Με τον όρο firmware, εννοούμε το λογισμικό που είναι ενσωματωμένο σε μία συσκευή, για παράδειγμα σε μια μνήμη ROM ή Flash. Τα προγράμματα σε ένα ενσωματωμένο σύστημα συχνά εκτελούνται με περιορισμούς πραγματικού χρόνου και με περιορισμένους πόρους υλικού, όπως χωρίς σκληρό δίσκο, λειτουργικό σύστημα, πληκτρολόγιο ή οθόνη. Το πρόγραμμα μπορεί να μην διαθέτει κανένα σύστημα αρχείων. Αν υποστηρίζει μία διεπαφή χρήστη, αυτή μπορεί να αποτελείται από ένα πληκτρολόγιο και μία οθόνη υγρών κρυστάλλων. Τα ενσωματωμένα συστήματα τοποθετούνται σε μηχανήματα που αναμένεται να λειτουργούν διαρκώς και για χωρίς σφάλματα. Με βάση τα παραπάνω το firmware αναπτύσσεται και δοκιμάζεται με μεγαλύτερη προσοχή από ότι το λογισμικό των προσωπικών υπολογιστών. Τα περισσότερα ενσωματωμένα συστήματα δεν χρησιμοποιούν μηχανικά μέρη, όπως οδηγούς δίσκων, διακόπτες ή κουμπιά, καθότι αυτά είναι αναξιόπιστα σε σχέση με τα σταθερά μέρη που χρησιμοποιούν τρανζίστορ, όπως μια μνήμη flash.

Τέλος πολλά ενσωματωμένα συστήματα μπορεί να τοποθετούνται εκεί όπου δεν είναι δυνατή η πρόσβαση από άνθρωπο και για το λόγο αυτό τα ενσωματωμένα συστήματα αυτά θα πρέπει να έχουν την δυνατότητα να μηδενίζονται και να αρχικοποιούνται ακόμα και σε

περιπτώσεις καταστροφικών διακοπών στην λειτουργία τους. Αυτό συνήθως επιτυγχάνεται με ένα πρότυπο ηλεκτρονικό κομμάτι που ονομάζεται χρονόμετρο watchdog, το οποίο μηδενίζει τον υπολογιστή εάν το λογισμικό δεν μηδενίζει περιοδικά το χρονόμετρο.

1.5 Πλατφόρμα υλοποίησης

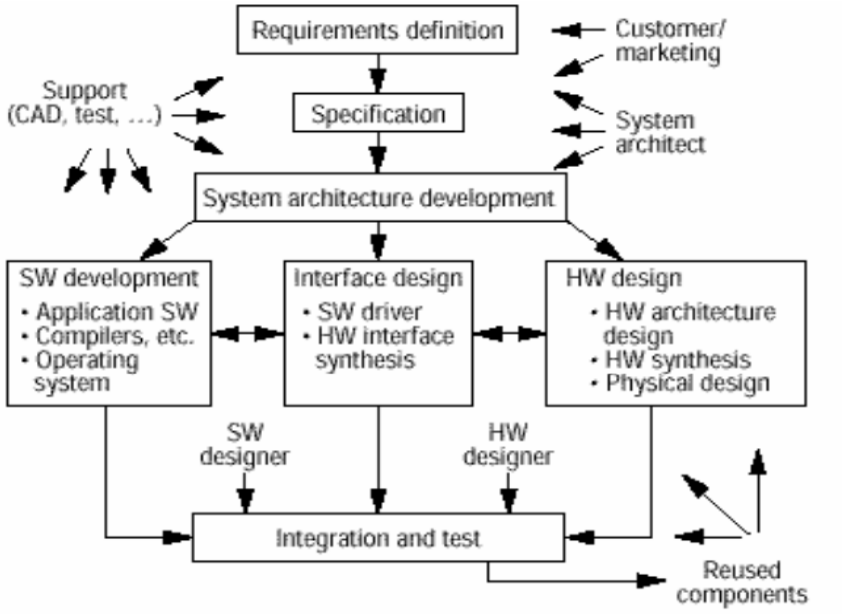
Υπάρχουν πολλές διαφορετικές αρχιτεκτονικές επεξεργαστών που χρησιμοποιούνται στα ενσωματωμένα συστήματα όπως οι ARM, MIPS, Coldfire/68k, PowerPC, X86, PIC, 8051 κ.α. Αυτό έρχεται σε αντίθεση με την αγορά των προσωπικών υπολογιστών, όπου ο ανταγωνισμός μεταξύ διαφορετικών αρχιτεκτονικών είναι περιορισμένος και κυρίως μεταξύ των Intel/AMD x86, και των Apple/Motorola/IBM PowerPC για τα Macintosh.

Μία διαφορετική τεχνική για τα ενσωματωμένα συστήματα, η οποία ακολουθήθηκε για τις εφαρμογές του παρόντος συγγράμματος, είναι η διαμόρφωση ενός προγραμματιζόμενου ολοκληρωμένου κυκλώματος (FPGA), ώστε να περιλαμβάνει τον επιθυμητό επεξεργαστή. Σε αυτήν την περίπτωση ο επεξεργαστής προμηθεύεται σαν πνευματική ιδιοκτησία, Intellectual Property (IP).

1.6 Σχεδίαση

Τα ενσωματωμένα συστήματα χρησιμοποιούν είτε μικροεπεξεργαστές είτε μικροελεγκτές. Ο σχεδιασμός του λογισμικού του συστήματος εξαρτάται από την προοριζόμενη χρήση του συστήματος, τις δυνατότητες του υλικού, και την ανάγκη ή όχι επικοινωνίας με τον χρήστη.

Όπως γίνεται αντιληπτό όταν χρησιμοποιηθούν FPGAs απαιτείται μία παράλληλη σχεδίαση τόσο του υλικού όσο και του λογισμικού του συστήματος, κάτι που φαίνεται στο σχήμα 1.4.



Σχήμα 1.4 – Η πορεία σχεδίασης ενός ενσωματωμένου συστήματος

Σύμφωνα με το παραπάνω σχήμα, αρχικά ορίζονται τα χαρακτηριστικά του συστήματος που θα σχεδιαστεί, τα οποία εξαρτώνται από την εφαρμογή που θα υλοποιεί. Στη συνέχεια οι σχεδιαστές αποφασίζουν ποια από αυτά θα υλοποιούνται από το λογισμικό και ποια από το υλικό. Έτσι, οι σχεδιαστές υλικού κατασκευάζουν την αρχιτεκτονική που έχει επιλεγεί

χρησιμοποιώντας κατάλληλα εργαλεία περιγραφής και σύνθεσης υλικού. Η κατασκευή αυτή δεν ξεκινάει από μηδενική βάση, καθώς τα εργαλεία που χρησιμοποιούνται δίνουν στους σχεδιαστές τη δυνατότητα να χρησιμοποιήσουν ξανά τμήματα υλικού που έχουν ήδη κατασκευαστεί και χρησιμοποιηθεί σε άλλα συστήματα. Από την άλλη μεριά, οι σχεδιαστές του λογισμικού υλοποιούν τις εφαρμογές χρησιμοποιώντας το απαιτούμενο λειτουργικό σύστημα και τους κατάλληλους μεταγλωττιστές (compilers). Επίσης, κατασκευάζονται οι οδηγοί (drivers) που είναι απαραίτητοι για την σωστή λειτουργία των περιφερειακών του συστήματος. Μόλις ολοκληρωθεί η κατασκευή του υλικού ακολουθεί η ενοποίηση υλικού-λογισμικού και η τελική δοκιμή του συστήματος.

1.7 Εργαλεία

Σαν ένας τυπικός προγραμματιστής υπολογιστών, ο σχεδιαστής ενσωματωμένων συστημάτων χρησιμοποιεί μεταγλωττιστές (compilers), συμβολομεταφραστές (assemblers) και εκσφαλματωτές (debuggers) για να αναπτύξει ένα σύστημα. Τα εργαλεία αυτά του λογισμικού μπορεί να προέρχονται από διάφορες πηγές:

- Εταιρίες λογισμικού που ειδικεύονται στην αγορά των ενσωματωμένων συστημάτων
- Μερικές φορές τα εργαλεία ανάπτυξης για προσωπικούς υπολογιστές μπορούν να χρησιμοποιηθούν αν ο επεξεργαστής του ενσωματωμένου συστήματος είναι συγγενής με τον κοινό επεξεργαστή ενός προσωπικού υπολογιστή.

Για την περίπτωση που προτιμηθεί η λύση του FPGA αντί ενός έτοιμου επεξεργαστή χρειάζεται ένα εργαλείο περιγραφής και σύνθεσης υλικού για την συγκεκριμένη αναπτυξιακή πλακέτα που περιέχει το FPGA που θα διαμορφωθεί. Η ομάδα εργαλείων του EDK της Xilinx που χρησιμοποιήθηκε στις εφαρμογές που θα ακολουθήσουν χειρίζεται τόσο το υλικό όσο και το λογισμικό του συστήματος.

1.8 Εκσφαλμάτωση

Η εκσφαλμάτωση συνήθως εκτελείται από κάποιον προσομοιωτή ή κάποιον εκσφαλματωτή που μπορεί να διακόπτει εσωτερικά τον κώδικα του μικροεπεξεργαστή. Η διακοπή της ροής του κώδικα επιτρέπει στον εκσφαλματωτή να επέμβει στο υλικό όπου μόνο ο επεξεργαστής βρίσκεται σε λειτουργία. Οι σχεδιαστές θα πρέπει να επιμένουν στην εκσφαλμάτωση της γλώσσας υψηλού επιπέδου, εισάγοντας breakpoints, εκτελώντας τις εντολές βήμα-βήμα, καθότι αυτή η τακτική είναι ευρέως αποδεκτή.

Καθώς η πολυπλοκότητα των ενσωματωμένων συστημάτων αυξάνεται, εργαλεία υψηλού επιπέδου και λειτουργικά συστήματα χρησιμοποιούνται εκεί όπου είναι απαραίτητα. Για παράδειγμα, τα κινητά τηλέφωνα, τα PDAs, και άλλα υπολογιστικά συστήματα συχνά χρειάζονται λογισμικό, το οποίο προμηθεύονται από πηγές διαφορετικές από αυτές που κατασκευάζουν τα ηλεκτρονικά μέρη, και το οποίο απαιτεί ένα ανοιχτό προγραμματιστικό περιβάλλον όπως το Linux, το OSGi ή η Java για να λειτουργήσει. Τα περιβάλλοντα αυτά τρέχουν και σε προσωπικούς υπολογιστές, επομένως το λογισμικό για τα συστήματα μπορεί να αναπτυχθεί σε έναν τυπικό προσωπικό υπολογιστή. Παρόλα αυτά η σύνδεση ενός τέτοιου περιβάλλοντος με τα ειδικευμένα ηλεκτρονικά και η ανάπτυξη των οδηγών (drivers) για τις συσκευές αυτών, παραμένει στην ευθύνη ενός μηχανικού λογισμικού των ενσωματωμένων συστημάτων.

Πηγές

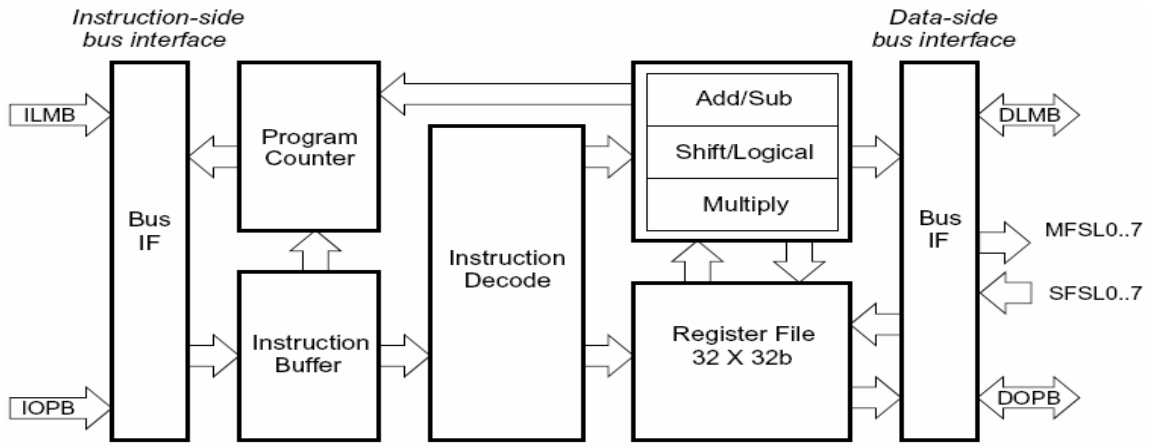
1. http://en.wikipedia.org/wiki/Embedded_system

2. Η Αρχιτεκτονική του MicroBlaze

2.1 Σύντομη περιγραφή

Ο επεξεργαστής MicroBlaze είναι ένας 32-bit RISC επεξεργαστής, ιδανικός για υλοποίηση σε FPGAs. Οι κυριότερες ιδιότητες και τα χαρακτηριστικά του ακολουθούν παρακάτω:

- Διαθέτει 32 καταχωρητές γενικού σκοπού και δύο καταχωρητές ειδικού σκοπού, όλοι μήκους 32 bit
- Το μήκος της εντολής του MicroBlaze είναι 32 bit και χρησιμοποιεί μέχρι τρεις τελεστές και δύο μεθόδους διευθυνσιοδότησης
- Διαθέτει ξεχωριστούς διαδρόμους δεδομένων, των 32 bit, για τις εντολές και για τα δεδομένα, που ακολουθούν τις προδιαγραφές του διαδρόμου δεδομένων OPB (On-chip Peripheral Bus) της IBM
- Διαθέτει ξεχωριστούς διαδρόμους δεδομένων, των 32 bit, για τις εντολές και για τα δεδομένα, με άμεση σύνδεση με block RAM, μέσω του διαδρόμου δεδομένων LMB (Local Memory Bus)
- Το μήκος των διευθύνσεων που χρησιμοποιεί είναι 32 bit
- Εφαρμόζει την παράλληλη, μέσω pipeline, εκτέλεση εντολών, με το pipeline να χωρίζεται σε τρία στάδια
- Διαθέτει γρήγορη μνήμη τόσο για τις εντολές όσο και για τα δεδομένα
- Υποστηρίζει την γρήγορη, σημείο – με – σημείο σύνδεση μέσω της διεπαφής FSL (Fast Simplex Link)
- Διαθέτει hardware πολλαπλασιαστή



Σχήμα 2.1 – Διάγραμμα του πυρήνα του MicroBlaze

2.2 Αναλυτική περιγραφή

2.2.1 Καταχωρητές

Ο MicroBlaze διαθέτει 32 καταχωρητές γενικού σκοπού και δύο καταχωρητές ειδικού σκοπού. Οι καταχωρητές γενικού σκοπού έχουν τις ονομασίες R0 έως R31, με τον R0 να είναι πάντοτε μηδενικός αγνοώντας οτιδήποτε γράφεται σε αυτόν. Οι δύο καταχωρητές ειδικού σκοπού είναι ο Program Counter (PC) και ο Machine Status Register (MSR). Όπως μαρτυρά η

ονομασία του, ο PC δείχνει σε κάθε στιγμή την διεύθυνση της επόμενης εντολής που πρόκειται να φορτωθεί για εκτέλεση. Ο MSR περιλαμβάνει το flag για το κρατούμενο υπερχείλισης και τα bits ενεργοποίησης για τις διακοπές, για την γρήγορη μνήμη (cache), για την διακοπή λειτουργίας του OPB καθώς και άλλες πληροφορίες κατάστασης και ελέγχου. Μια πλήρης περιγραφή της χρήσης των bits του MSR γίνεται στον πίνακα 2.1

Πίνακας 2.1 – Ο καταχωρητής MSR

Bits	Name	Description	Reset Value
0	CC	Arithmetic Carry Copy Copy of the Arithmetic Carry (bit 29). Read only.	0
1:23	Reserved		
24	DCE	Data Cache Enable 0 Data Cache is Disabled 1 Data Cache is Enabled	0
25	DZ	Dvision by Zero 0 No divison by zero has occured 1 Division by zero has occured	0
26	ICE	Instruction Cache Enable 0 Instruction Cache is Disabled 1 Instruction Cache is Enabled	0
27	FSL	FSL Error 0 FSL get/put had no error 1 FSL get/put had mismatch in instruction type and value type	0
28	BIP	Break in Progress 0 No Break in Progress 1 Break in Progress Source of break can be software break instruction or hardware break from Ext_Brk or Ext_NM_Brk pin.	0

2.2.2 Εντολές

Όλες οι εντολές του MicroBlaze έχουν μήκος 32 bit και χωρίζονται σε δύο κατηγορίες, τις εντολές τύπου A και τις εντολές τύπου B. Οι εντολές τύπου A χρησιμοποιούν δύο καταχωρητές προέλευσης και έναν καταχωρητή προορισμού, ενώ οι εντολές τύπου B έναν καταχωρητή

προέλευσης μαζί με ένα άμεσο όρισμα των 16 bit και έναν καταχωρητή προορισμού. Όλες οι εντολές με την σημασιολογία (semantics) τους αναφέρονται στον πίνακα 2.2.

Πίνακας 2.2 – Σύνοψη των εντολών του MicroBlaze

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + 1
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + C
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + 1
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + C
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000001	Rd := Rb cmp Ra (signed)
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000011	Rd := Rb cmp Ra (unsigned)
ADDI Rd,Ra,Imm	001000	Rd	Ra	Imm		Rd := s(Imm) + Ra
RSUBI Rd,Ra,Imm	001001	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + 1
ADDIC Rd,Ra,Imm	001010	Rd	Ra	Imm		Rd := s(Imm) + Ra + C
RSUBIC Rd,Ra,Imm	001011	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + C
ADDIK Rd,Ra,Imm	001100	Rd	Ra	Imm		Rd := s(Imm) + Ra
RSUBIK Rd,Ra,Imm	001101	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + 1
ADDIKC Rd,Ra,Imm	001110	Rd	Ra	Imm		Rd := s(Imm) + Ra + C
RSUBIKC Rd,Ra,Imm	001111	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + C
MUL Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000000	Rd := Ra * Rb
BSRL Rd,Ra,Rb	010001	Rd	Ra	Rb	00000000000	Rd : = Ra >> Rb
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	01000000000	Rd := Ra[0], (Ra >> Rb)
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	10000000000	Rd := Ra << Rb
MULI Rd,Ra,Imm	011000	Rd	Ra	Imm		Rd := Ra * s(Imm)
BSRLI Rd,Ra,Imm	011001	Rd	Ra	0000	0000.. Imm	Rd : = Ra >> Imm
BSRAI Rd,Ra,Imm	011001	Rd	Ra	0000	0100.. Imm	Rd := Ra[0], (Ra >> Imm)
BSLLI Rd,Ra,Imm	011001	Rd	Ra	0000	1000.. Imm	Rd := Ra << Imm
IDIV Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000000	Rd := Rb/Ra, signed
IDIVU Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000001	Rd := Rb/Ra, unsigned
GET Rd,FSLx	011011	Rd	00000	0000	FSLx	Rd := FSLx (blocking data read)
PUT Ra,FSLx	011011	00000	Ra	1000	FSLx	FSLx := Ra (blocking data write)
nGET Rd,FSLx	011011	Rd	00000	0100	FSLx	Rd := FSLx (non-blocking data read)
nPUT Ra,FSLx	011011	00000	Ra	1100	FSLx	FSLx := Ra (non-blocking data write)

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
cGET Rd,FSLx	011011	Rd	00000	0010	FSLx	Rd := FSLx (blocking control read)
cPUT Ra,FSLx	011011	00000	Ra	1010	FSLx	FSLx := Ra (blocking control write)
ncGET Rd,FSLx	011011	Rd	00000	0110	FSLx	Rd := FSLx (non-blocking control read)
ncPUT Ra,FSLx	011011	00000	Ra	1110	FSLx	FSLx := Ra (non-blocking control write)
OR Rd,Ra,Rb	100000	Rd	Ra	Rb	00000000000	Rd := Ra or Rb
AND Rd,Ra,Rb	100001	Rd	Ra	Rb	00000000000	Rd := Ra and Rb
XOR Rd,Ra,Rb	100010	Rd	Ra	Rb	00000000000	Rd := Ra xor Rb
ANDN Rd,Ra,Rb	100011	Rd	Ra	Rb	00000000000	Rd := Ra and $\overline{Rb}$
SRA Rd,Ra	100100	Rd	Ra	0000000000000001		Rd := Ra[0], (Ra >> 1); C := Ra[31]
SRC Rd,Ra	100100	Rd	Ra	0000000000100001		Rd := C, (Ra >> 1); C := Ra[31]
SRL Rd,Ra	100100	Rd	Ra	0000000001000001		Rd := 0, (Ra >> 1); C := Ra[31]
SEXT8 Rd,Ra	100100	Rd	Ra	0000000001100000		Rd[0:23] := Ra[24]; Rd[24:31] := Ra[24:31]
SEXT16 Rd,Ra	100100	Rd	Ra	0000000001100001		Rd[0:15] := Ra[16]; Rd[16:31] := Ra[16:31]
WIC Rd,Ra	100100	Ra	Ra	Rb	01101000	ICache_Tag := Ra, ICache_Data := Rb
WDC Rd,Ra	100100	Ra	Ra	Rb	01100100	DCache_Tag := Ra, DCache_Data := Rb
MTS Sd,Ra	100101	00000	Ra	110000000000000d		Sd := Ra , where S1 is MSR
MFS Rd,Sa	100101	Rd	00000	100000000000000a		Rd := Sa , where S0 is PC and S1 is MSR
BR Rb	100110	00000	00000	Rb	00000000000	PC := PC + Rb
BRD Rb	100110	00000	10000	Rb	00000000000	PC := PC + Rb
BRLD Rd,Rb	100110	Rd	10100	Rb	00000000000	PC := PC + Rb; Rd := PC
BRA Rb	100110	00000	01000	Rb	00000000000	PC := Rb
BRAD Rb	100110	00000	11000	Rb	00000000000	PC := Rb
BRALD Rd,Rb	100110	Rd	11100	Rb	00000000000	PC := Rb; Rd := PC
BRK Rd,Rb	100110	Rd	01100	Rb	00000000000	PC := Rb; Rd := PC; MSR[BIP] := 1
BEQ Ra,Rb	100111	00000	Ra	Rb	00000000000	if Ra = 0: PC := PC + Rb
BNE Ra,Rb	100111	00001	Ra	Rb	00000000000	if Ra /= 0: PC := PC + Rb
BLT Ra,Rb	100111	00010	Ra	Rb	00000000000	if Ra < 0: PC := PC + Rb
BLE Ra,Rb	100111	00011	Ra	Rb	00000000000	if Ra <= 0: PC := PC + Rb
BGT Ra,Rb	100111	00100	Ra	Rb	00000000000	if Ra > 0: PC := PC + Rb
BGE Ra,Rb	100111	00101	Ra	Rb	00000000000	if Ra >= 0: PC := PC + Rb
BEQD Ra,Rb	100111	10000	Ra	Rb	00000000000	if Ra = 0: PC := PC + Rb

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
BNED Ra,Rb	100111	10001	Ra	Rb	00000000000	if Ra /= 0: PC := PC + Rb
BLTD Ra,Rb	100111	10010	Ra	Rb	00000000000	if Ra < 0: PC := PC + Rb
BLED Ra,Rb	100111	10011	Ra	Rb	00000000000	if Ra <= 0: PC := PC + Rb
BGTD Ra,Rb	100111	10100	Ra	Rb	00000000000	if Ra > 0: PC := PC + Rb
BGED Ra,Rb	100111	10101	Ra	Rb	00000000000	if Ra >= 0: PC := PC + Rb
ORI Rd,Ra,Imm	101000	Rd	Ra	Imm		Rd := Ra or s(Imm)
ANDI Rd,Ra,Imm	101001	Rd	Ra	Imm		Rd := Ra and s(Imm)
XORI Rd,Ra,Imm	101010	Rd	Ra	Imm		Rd := Ra xor s(Imm)
ANDNI Rd,Ra,Imm	101011	Rd	Ra	Imm		Rd := Ra and $\overline{\text{s(Imm)}}$
IMM Imm	101100	00000	00000	Imm		Imm[0:15] := Imm
RTSD Ra,Imm	101101	10000	Ra	Imm		PC := Ra + s(Imm)
RTID Ra,Imm	101101	10001	Ra	Imm		PC := Ra + s(Imm); MSR[IE] := 1
RTBD Ra,Imm	101101	10010	Ra	Imm		PC := Ra + s(Imm); MSR[BIP] := 0
BRID Imm	101110	00000	10000	Imm		PC := PC + s(Imm)
BRLID Rd,Imm	101110	Rd	10100	Imm		PC := PC + s(Imm); Rd := PC
BRAI Imm	101110	00000	01000	Imm		PC := s(Imm)
BRAID Imm	101110	00000	11000	Imm		PC := s(Imm)
BRALID Rd,Imm	101110	Rd	11100	Imm		PC := s(Imm); Rd := PC
BRKI Rd,Imm	101110	Rd	01100	Imm		PC := s(Imm); Rd := PC; MSR[BIP] := 1
BEQI Ra,Imm	101111	00000	Ra	Imm		if Ra = 0: PC := PC + s(Imm)
BNEI Ra,Imm	101111	00001	Ra	Imm		if Ra /= 0: PC := PC + s(Imm)
BLTI Ra,Imm	101111	00010	Ra	Imm		if Ra < 0: PC := PC + s(Imm)
BLEI Ra,Imm	101111	00011	Ra	Imm		if Ra <= 0: PC := PC + s(Imm)
BGTI Ra,Imm	101111	00100	Ra	Imm		if Ra > 0: PC := PC + s(Imm)
BGEI Ra,Imm	101111	00101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
BEQID Ra,Imm	101111	10000	Ra	Imm		if Ra = 0: PC := PC + s(Imm)
BNEID Ra,Imm	101111	10001	Ra	Imm		if Ra /= 0: PC := PC + s(Imm)
BLTID Ra,Imm	101111	10010	Ra	Imm		if Ra < 0: PC := PC + s(Imm)
BLEID Ra,Imm	101111	10011	Ra	Imm		if Ra <= 0: PC := PC + s(Imm)
BGTID Ra,Imm	101111	10100	Ra	Imm		if Ra > 0: PC := PC + s(Imm)
BGEID Ra,Imm	101111	10101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
LBU Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd[0:23] := 0, Rd[24:31] := *Addr

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
LHU Rd,Ra,Rb	110001	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd[0:15] := 0, Rd[16:31] := *Addr
LW Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; Rd := *Addr
SB Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; *Addr := Rd[24:31]
SH Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; *Addr := Rd[16:31]
SW Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000	Addr := Ra + Rb; *Addr := Rd
LBUI Rd,Ra,Imm	111000	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd[0:23] := 0, Rd[24:31] := *Addr
LHUI Rd,Ra,Imm	111001	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd[0:15] := 0, Rd[16:31] := *Addr
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm); Rd := *Addr
SBI Rd,Ra,Imm	111100	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd[24:31]
SHI Rd,Ra,Imm	111101	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd[16:31]
SWI Rd,Ra,Imm	111110	Rd	Ra	Imm		Addr := Ra + s(Imm); *Addr := Rd

Όταν ενεργοποιηθεί το σήμα Reset ή το Debug_Rst, ο MicroBlaze ξεκινά να εκτελεί την εντολή που βρίσκεται στην διεύθυνση 0, ακολουθούμενη από τις εντολές που καθορίζει το πρόγραμμα που είναι φορτωμένο στην μνήμη. Εκτός όμως από τις εντολές που εκτελούνται μέσω του software, ιδιαίτερη μνεία αξίζει να γίνει στις εντολές που εκτελούνται όταν συμβαίνουν διακοπές (interrupts), εξαιρέσεις (exceptions) και σταμάτημα λειτουργίας (break), που οφείλονται στο υλικό (hardware).

Όταν συμβεί μία διακοπή, ο MicroBlaze αναβάλλει την εκτέλεση της επόμενης εντολής προκειμένου να την εξυπηρετήσει. Η διεύθυνση της επόμενης εντολής που επρόκειτο να εκτελεστεί αποθηκεύεται στον καταχωρητή γενικού σκοπού R14 και ο MicroBlaze διακλαδίζεται στην διεύθυνση 0x00000010. Παράλληλα απενεργοποιεί τυχόν μελλοντικές διακοπές θέτοντας το κατάλληλο flag, bit 30, ‘0’ στον καταχωρητή MSR. Όταν το bit αυτό είναι 0 οι διακοπές αγνοούνται, όπως συμβαίνει και στην περίπτωση που το bit 28 του MSR είναι ‘1’, καθώς τα breaks έχουν μεγαλύτερη προτεραιότητα από τις διακοπές.

Ομοίως με τις διακοπές, όταν συμβεί μία εξαίρεση ο MicroBlaze αναβάλλει την εκτέλεση της επόμενης εντολής προκειμένου να την χειριστεί. Η διεύθυνση της επόμενης εντολής που επρόκειτο να εκτελεστεί αποθηκεύεται στον καταχωρητή γενικού σκοπού R17 και ο MicroBlaze διακλαδίζεται στην διεύθυνση 0x00000008.

Τέλος έχουμε τα break υλικού που οδηγούνται μέσω των ακροδεκτών Ext_Brk και Ext_NM_Brk. Και στις δύο περιπτώσεις ο MicroBlaze αναβάλλει την εκτέλεση της επόμενης εντολής προκειμένου να χειριστεί το break. Η διεύθυνση της επόμενης εντολής που επρόκειτο

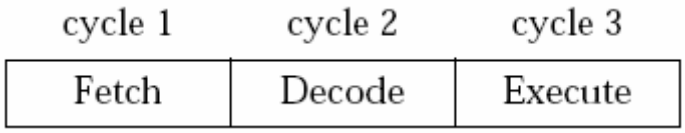
να εκτελεστεί αποθηκεύεται στον καταχωρητή γενικού σκοπού R16 και ο MicroBlaze διακλαδίζεται στην διεύθυνση 0x00000018. Παράλληλα ο MicroBlaze αποκλείει τυχόν μελλοντικά break θέτοντας το bit 28 του MSR '1' ώστε να δείχνει ότι υπάρχει ήδη ένα break σε εξέλιξη. Μονάχα όταν το bit αυτό έχει την τιμή '0', είναι δυνατόν να χειριστεί ένα break οδηγούμενο από το σήμα Ext_Brk, στην περίπτωση όμως που το break προκαλείται από το σήμα Ext_NM_Brk, τότε ο MicroBlaze το χειρίζεται άμεσα.

2.2.3 Pipeline

Το pipeline του MicroBlaze χωρίζεται σε τρία στάδια:

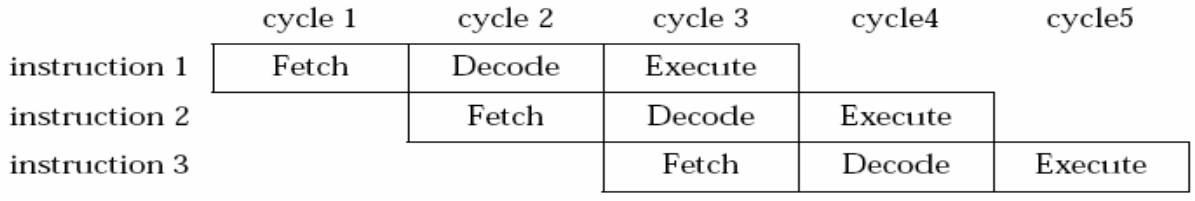
- Ανάκληση της εντολής
- Αποκωδικοποίηση της εντολής
- Εκτέλεση της εντολής

Γενικά κάθε στάδιο χρειάζεται έναν κύκλο ρολογιού για να ολοκληρωθεί, επομένως χρειάζονται τρεις κύκλοι ρολογιού, χωρίς να υπολογίζονται οι καθυστερήσεις ή οι αναβολές, για να εκτελεστεί μία εντολή.



Σχήμα 2.2 – Κάθε εντολή εκτελείται σε τρεις κύκλους

Στο παράλληλο pipeline του MicroBlaze κάθε στάδιο είναι ενεργό σε κάθε κύκλο ρολογιού. Τρεις εντολές είναι δυνατόν να εκτελούνται ταυτόχρονα, μία σε κάθε ένα από τα τρία στάδια του pipeline. Παρά το ότι χρειάζονται τρεις κύκλοι για την εκτέλεση μιας εντολής, κάθε στάδιο του pipeline μπορεί να είναι απασχολημένο με διαφορετική εντολή. Σε ένα κύκλο ρολογιού γίνεται ανάκληση μιας νέας εντολής, μία άλλη αποκωδικοποιείται και μία τρίτη ολοκληρώνει την εκτέλεσή της. Με τον τρόπο αυτό επιτυγχάνεται η εκτέλεση μιας εντολής ανά κύκλο ρολογιού.



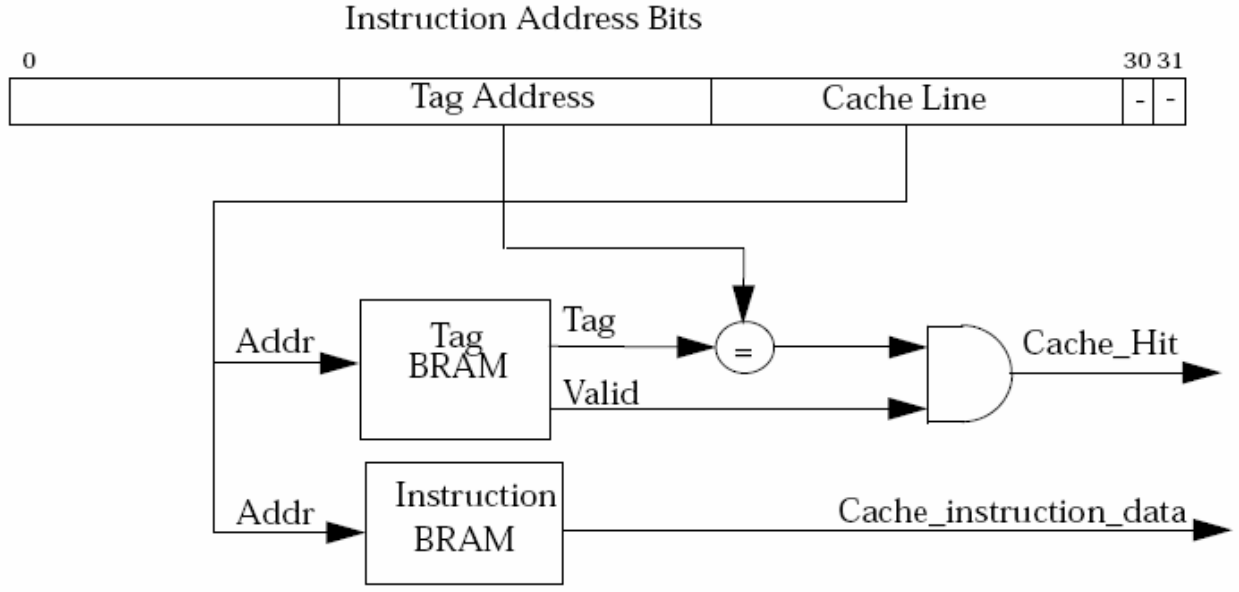
Σχήμα 2.3 – Παράλληλη εκτέλεση τριών εντολών

Όπως συμβαίνει με όλα τα pipeline των επεξεργαστών υπάρχουν εντολές που αλλάζουν την ροή του προγράμματος, εντολές διακλαδώσεων, και κατ' επέκταση τον ρυθμό εκτέλεσης των εντολών. Όταν μια εντολή διακλάδωσης βρεθεί στο στάδιο της εκτέλεσης, τότε η δουλειά που έχει γίνει στα προηγούμενα δύο στάδια δεν έχει καμία χρησιμότητα. Η δουλειά αυτή απορρίπτεται και το pipeline αδειάζει. Προκειμένου το pipeline να ξαναγεμίσει με τις σωστές εντολές, και δεδομένου ότι η εκτέλεση της εντολής διακλάδωσης χρειάζεται τρεις κύκλους ρολογιού, υπάρχει μία καθυστέρηση δύο κύκλων ρολογιού. Κάποιες εντολές διακλάδωσης καθιστούν δυνατό να μειωθεί ο χρόνος καθυστέρησης από δύο κύκλους σε έναν. Αυτό γίνεται με το να απορρίπτεται μόνο η εργασία που γίνεται στο πρώτο στάδιο του pipeline, ανάκληση

εντολής. Με τον τρόπο αυτό η εντολή που βρίσκεται μετά την εντολή διακλάδωσης, εκτελείται κανονικά και χάνεται μονάχα ένας κύκλος ρολογιού.

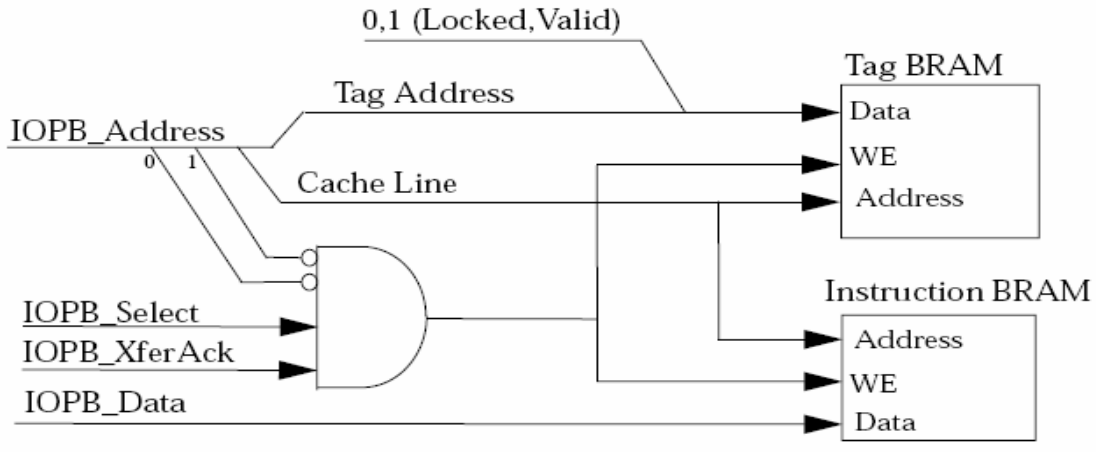
2.2.4 Γρήγορη Μνήμη

Ο MicroBlaze χρησιμοποιεί γρήγορη μνήμη, τόσο για τις εντολές, instruction cache, όσο και για τα δεδομένα, data cache. Η γρήγορη μνήμη για τις εντολές χρησιμοποιείται για καλύτερη επίδοση όταν υπάρχει εκτελέσιμος κώδικας που ανήκει σε μνήμες πέραν του εύρους διευθύνσεων του LMB. Στην περίπτωση αυτή ο χώρος της μνήμης χωρίζεται σε δύο μέρη, το ένα δύναται να εγγράφεται στην γρήγορη μνήμη , ενώ το άλλο, στο οποίο πρέπει να ανήκουν και οι διευθύνσεις του LMB, όχι. Το πρώτο τμήμα καθορίζεται από δύο παραμέτρους, C_ICACHE_BASEADDR και C_ICACHE_HIGHADDR. Όλες οι διευθύνσεις μέσα σε αυτό το εύρος χωρίζονται σε δύο τμήματα, στο τμήμα cache line και στο τμήμα tag address. Τα μεγέθη των δύο τμημάτων καθορίζονται από τον χρήστη, ενώ το τμήμα μεταξύ του bit 0 και του tag address αγνοείται και τα bit 30 και 31 είναι δεσμευμένα. Το μέγεθος του cache line είναι μεταξύ 10 και 14 bit, που μεταφράζεται σε μέγεθος γρήγορης μνήμης που κυμαίνεται μεταξύ 4 Kbytes και 64 Kbytes. Δεν υπάρχει περιορισμός στο μέγεθος του tag address.



Σχήμα 2.4 – Η οργάνωση της instruction cache

Στο στάδιο της ανάκλησης της εντολής, ο MicroBlaze γράφει την διεύθυνση της επόμενης εντολής στον αντίστοιχο διάδρομο δεδομένων και περιμένει το σήμα επιβεβαίωσης. Για οικονομία χρόνου το αίτημα της ανάκλησης γίνεται ταυτόχρονα και στον LMB και στον OPB. Αν το σήμα επιβεβαίωσης έρθει από τον LMB στον επόμενο κύκλο, τότε ακυρώνεται η πρόσβαση στον OPB. Κάθε φορά η γρήγορη μνήμη των εντολών ανιχνεύει εάν η διεύθυνση της εντολής ανήκει στο εύρος των εντολών που δύνανται να εγγράφονται στην γρήγορη μνήμη. Αν κάτι τέτοιο δεν συμβαίνει, τότε η γρήγορη μνήμη αφήνει την διεκπεραίωση της αίτησης στους LMB ή OPB. Αν ισχύει το αντίθετο τότε η γρήγορη μνήμη ελέγχει αν έχει την εντολή που ζητείται. Αυτό συμβαίνει όταν το tag address της ζητούμενης εντολής είναι το ίδιο με αυτό που έχει η γρήγορη μνήμη, αντιστοιχίζοντας το cache line, και εάν το bit εγκυρότητας είναι ενεργοποιημένο. Εάν η εντολή βρίσκεται στην γρήγορη μνήμη, τότε θα στείλει το σήμα επιβεβαίωσης στον MicroBlaze και μαζί την εντολή. Εάν όμως δεν την έχει θα περιμένει έως ότου η αίτηση διεκπεραιωθεί από τον OPB, και έπειτα θα ενημερώσει το περιεχόμενό της με τις νέες πληροφορίες.



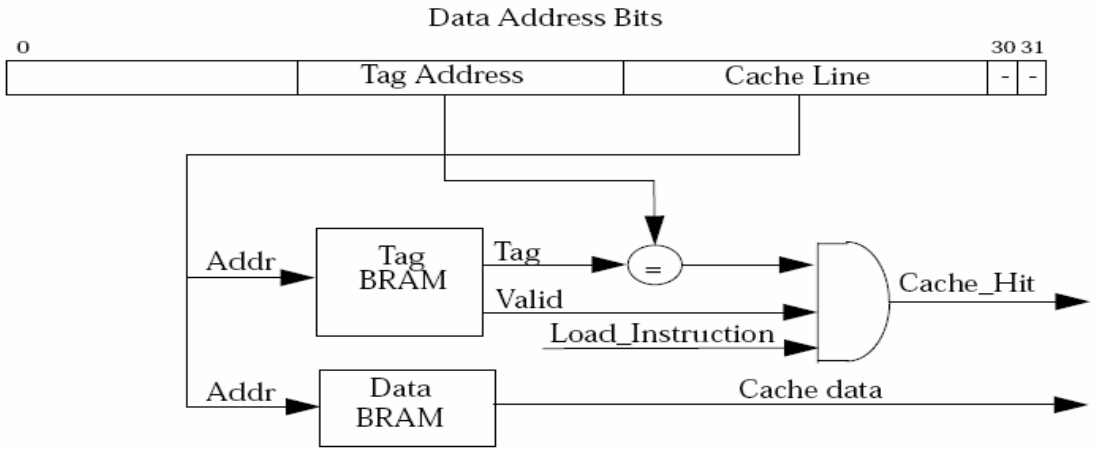
Σχήμα 2.5 – Διαδικασία εγγραφής στην γρήγορη μνήμη

Η ενεργοποίηση ή μη της γρήγορης μνήμης, για τις εντολές, ελέγχεται από το bit 26 του MSR. Τα δεδομένα της γρήγορης μνήμης διατηρούνται ανέπαφα μετά την απενεργοποίησή της, παρ' όλ' αυτά υπάρχει η δυνατότητα μέσω της εντολής

WIC Ra, Rb

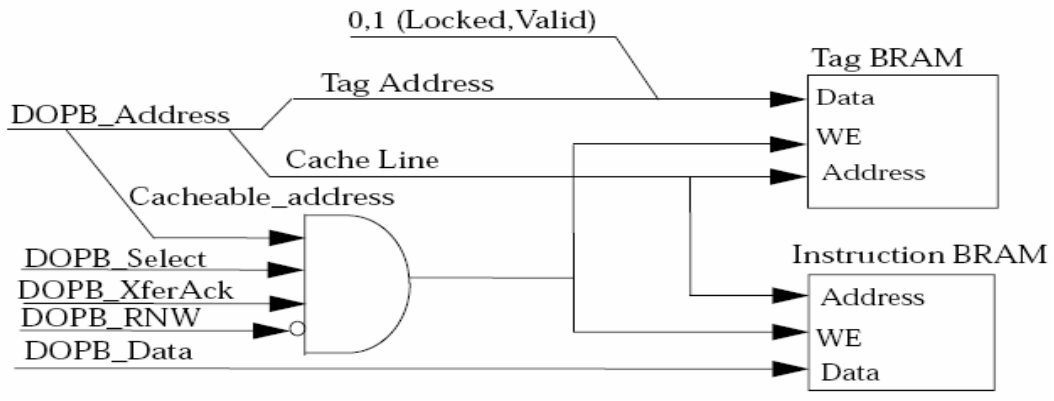
να αλλάξει το περιεχόμενο της γρήγορης μνήμης, στην διάρκεια μόνο που αυτή είναι απενεργοποιημένη. Ο καταχωρητής Rb περιέχει την εντολή. Ο καταχωρητής Ra περιέχει τα τμήματα cache line και tag address, καθώς και το bit εγκυρότητας, Ra(31), με το bit κλειδώματος, RA(30). Το bit κλειδώματος, επιτρέπει την μόνιμη παραμονή μιας εντολής στην γρήγορη μνήμη, με αποτέλεσμα τον εγγυημένο χρόνο εκτέλεσής της. Η συγκεκριμένη τεχνική όμως έχει ως αποτέλεσμα την μείωση του μεγέθους της γρήγορης μνήμης και κατ' επέκταση τον αριθμό των επιτυχημένων αναζητήσεων στην γρήγορη μνήμη (hits). Για το λόγο αυτό είναι προτιμότερο οι εντολές αυτού του τύπου να βρίσκονται στην μνήμη του LMB.

Η οργάνωση και η λειτουργία της γρήγορης μνήμης για τα δεδομένα είναι σχεδόν η ίδια με αυτήν της γρήγορης μνήμης για τις εντολές. Και σε αυτήν την περίπτωση η μνήμη χωρίζεται σε δύο τμήματα, εκ των οποίων το ένα, που οριοθετείται από τις παραμέτρους C_DCACHE_BASEADDR και C_DCACHE_HIGHADDR, περιλαμβάνει διευθύνσεις δεδομένων που είναι δυνατό να βρίσκονται στην γρήγορη μνήμη. Στο τμήμα αυτό δεν πρέπει να περιλαμβάνονται οι διευθύνσεις που αντιστοιχούν στην μνήμη του LMB. Οι διευθύνσεις χωρίζονται σε δύο τμήματα, tag address και cache line, και το μέγεθος της γρήγορης μνήμης κυμαίνεται από 4 Kbytes έως 64 Kbytes.



Σχήμα 2.6 – Η οργάνωση της data cache

Όταν εκτελείται μία εντολή αποθήκευσης δεδομένων, η διαδικασία ακολουθείται κανονικά με την διαφορά ότι εάν η διεύθυνση αποθήκευσης ανήκει στο τμήμα που αναφέρθηκε παραπάνω, τότε γίνεται και ενημέρωση των δεδομένων στην γρήγορη μνήμη. Στην περίπτωση που ο MicroBlaze εκτελεί μια εντολή φόρτωσης δεδομένων, πρώτα ελέγχεται εάν η διεύθυνση ανήκει στην ομάδα των διευθύνσεων που είναι δυνατό να βρίσκονται στην κύρια μνήμη, και μετά ελέγχεται εάν το ζητούμενο δεδομένο βρίσκεται στην γρήγορη μνήμη. Εάν η γρήγορη μνήμη διαθέτει το δεδομένο τότε στέλνει ένα σήμα επιβεβαίωσης στον MicroBlaze και μαζί τα δεδομένα. Αντιθέτως εάν η γρήγορη μνήμη δεν έχει το δεδομένο, αφήνει την διεκπεραίωση της φόρτωσης στον OPB.



Σχήμα 2.7 – Διαδικασία εγγραφής στην γρήγορη μνήμη

Η ενεργοποίηση και η απενεργοποίηση της γρήγορης μνήμης δεδομένων ελέγχονται από το bit 24 του MSR. Τα δεδομένα διατηρούνται και μετά την απενεργοποίηση της γρήγορης μνήμης, ενώ τον ρόλο της εντολής WIC, που χρησιμοποιείται για την γρήγορη μνήμη εντολών, παίζει η

WDC Ra, Rb

τα ορίσματα της οποίας περιέχουν ότι και τα αντίστοιχα της WIC με την διαφορά ότι ο καταχωρητής Rb περιέχει δεδομένα και όχι εντολές.

2.2.5 Διεπαφή FSL (Fast Simplex Link)

Ο MicroBlaze διαθέτει 16 διεπαφές FSL, οκτώ εισόδου και οκτώ εξόδου. Οι διεπαφές αυτές είναι αποκλειστικές συνδέσεις σημείου προς σημείο μεταξύ του MicroBlaze και των περιφερειακών. Το μήκος του διαδρόμου δεδομένων του FSL είναι 32 bit, τα οποία μπορεί να είναι είτε εντολές ελέγχου, είτε δεδομένα. Ένα ξεχωριστό bit καταδεικνύει πότε μία λέξη που εκπέμπεται ή λαμβάνεται, είναι δεδομένο ή εντολή ελέγχου.

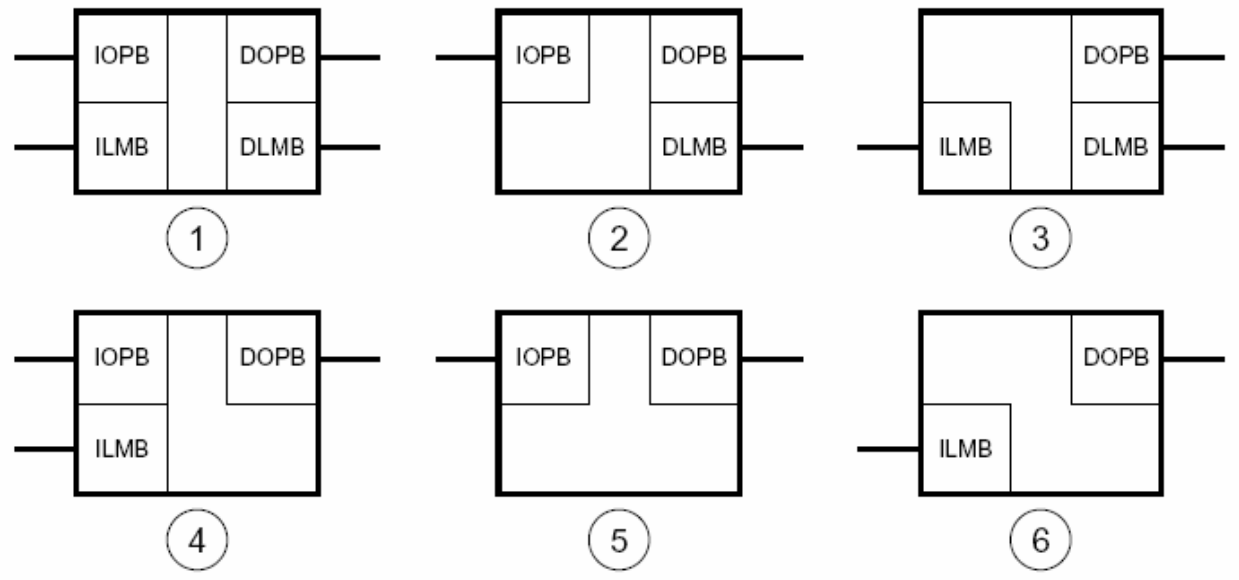
Ο MicroBlaze διαβάζει από μια είσοδο FSL με τέσσερις εντολές, δύο για τα δεδομένα και δύο για τις εντολές ελέγχου. Η εντολή get παγώνει το pipeline του MicroBlaze μέχρις ότου δεδομένα να γίνουν διαθέσιμα στην είσοδο της FSL σύνδεσης. Η εντολή nget επίσης περιμένει δεδομένα στην είσοδο της FSL σύνδεσης αλλά, δεν μπλοκάρει το pipeline. Αντί για αυτό θέτει την τιμή ‘0’ στο bit 29 του MSR όταν τα δεδομένα είναι διαθέσιμα ώστε να διαβαστούν. Και στις δύο εντολές αν επιχειρηθεί να περαστούν εντολές ελέγχου και όχι δεδομένα ενεργοποιείται το bit λάθους του FSL, που αντιστοιχεί στο bit 27 του MSR. Αντίστοιχες ιδιότητες και λειτουργίες με τις εντολές get και nget, έχουν οι εντολές cget και ncget, με την μόνη διαφορά ότι αφορούν αποκλειστικά εντολές ελέγχου και όχι δεδομένα. Οι τέσσερις εντολές ανάγνωσης από την διεπαφή FSL ολοκληρώνονται σε δύο κύκλους ρολογιού από την στιγμή που τα δεδομένα είναι διαθέσιμα.

Ανάλογη είναι και η λογική των εντολών που χρησιμοποιούνται για την εγγραφή μέσω μιας σύνδεσης FSL. Υπάρχουν δύο εντολές για την εγγραφή δεδομένων, η put που γράφει μπλοκάροντας το pipeline του MicroBlaze μέχρις ότου τα δεδομένα να μπορέσουν να προωθηθούν, και η nput που δεν μπλοκάρει το pipeline αλλά θέτει την τιμή '0' στο bit 29 του MSR όταν η εγγραφή ολοκληρωθεί επιτυχώς. Και οι δύο εντολές χειρίζονται μονάχα δεδομένα. Για τις εντολές ελέγχου χρησιμοποιούνται οι εντολές cput, που μπλοκάρει το pipeline, και η ncput που δεν το μπλοκάρει και χρησιμοποιεί και αυτή το bit 29 του καταχωρητή MSR. Ομοίως οι δύο παραπάνω εντολές χειρίζονται αποκλειστικά εντολές ελέγχου, ενώ όλες οι εντολές εγγραφής χρειάζονται δύο κύκλους ρολογιού για να ολοκληρωθούν από την στιγμή που τα δεδομένα για εγγραφή είναι διαθέσιμα.

2.2.6 Διάδρομοι Δεδομένων του MicroBlaze

Οι δύο κύριοι διάδρομοι δεδομένων του MicroBlaze, είναι ο Local Memory Bus (LMB), ο οποίος παρέχει πρόσβαση σε μία dual-port μνήμη BRAM που είναι προσπελάσιμη σε ένα κύκλο ρολογιού, και ο On-chip Peripheral Bus (OPB), που παρέχει σύνδεση με εσωτερικά και εξωτερικά περιφερειακά και μνήμη. Οι LMB και OPB χωρίζονται περαιτέρω στους DLMB, Data interface LMB, DOPB, Data interface OPB, που χειρίζονται μόνο δεδομένα και στους ILMB, Instruction interface LMB, IOPB, Instruction interface OPB, που χειρίζονται μόνο εντολές. Εδώ πρέπει να τονιστεί ότι όλα τα περιφερειακά συνδέονται στον DOPB. Εκτός από τους LMB και OPB ο MicroBlaze διαθέτει και οκτώ κύριες (Master) και οκτώ εξαρτημένες (Slave) διεπαφές, για συνδέσεις FSL.

Η δομή του MicroBlaze όσον αφορά τους διαδρόμους δεδομένων είναι δυνατόν να διαμορφωθεί με έξι διαφορετικούς τρόπους, όπως φαίνεται στο σχήμα 2.8 και στον πίνακα 2.3. Σε αυτούς τους έξι τρόπους δεν συνυπολογίζεται η χρησιμοποίηση ή μη των συνδέσεων FSL.



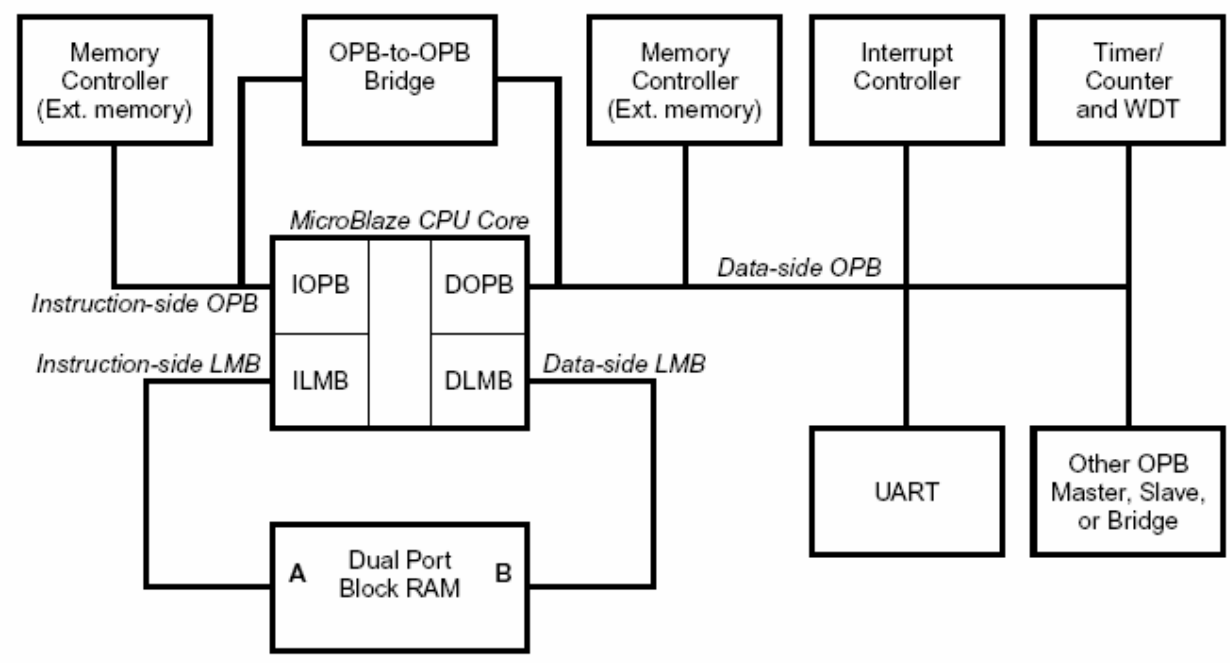
Σχήμα 2.8 – Οι έξι τρόποι διαμόρφωσης των διαδρόμων δεδομένων του MicroBlaze

Η επιλογή κάθε ενός από τους έξι τρόπους διαμόρφωσης εξαρτάται από το μέγεθος του εκτελέσιμου κώδικα, την ανάγκη σε χώρο για αποθήκευση δεδομένων και στην απαίτηση για γρήγορη πρόσβαση στην εσωτερική μνήμη BRAM.

Πίνακας 2.3 – Οι έξι διαμορφώσεις των διαδρόμων δεδομένων του MicroBlaze

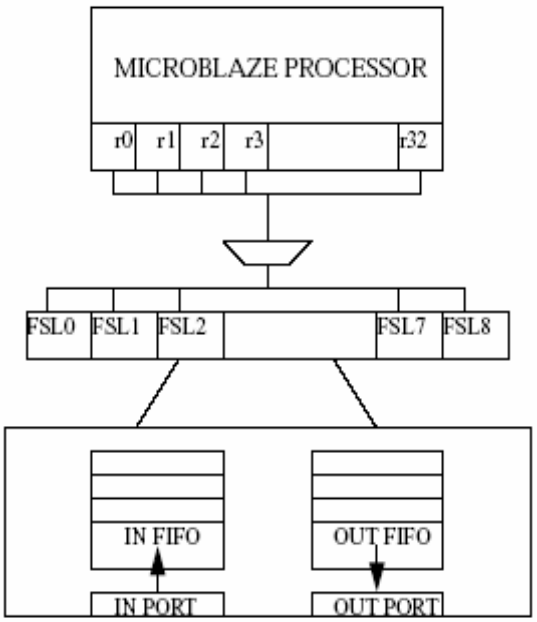
	Διαμόρφωση	Μέγιστη Συχνότητα	Υποστηριζόμενο Μοντέλο Μνήμης	Τυπικές Εφαρμογές
1	IOPB+ILMB+DOPB+DLMB	110	Μεγάλη εξωτερική μνήμη Γρήγορη εσωτερική μνήμη	Για εφαρμογές που χρειάζονται μνήμη μεγαλύτερη από αυτή που προσφέρει η BRAM. MPEG αποκωδικοποιητές, ελεγκτές επικοινωνιών
2	IOPB+DOPB+DLMB	125	Μεγάλη εξωτερική μνήμη Γρήγορη εσωτερική μνήμη για δεδομένα (BRAM)	Παρόμοιες με την περίπτωση 1 Επιτυγχάνεται μεγαλύτερη συχνότητα αλλά η λειτουργία είναι πιο αργή λόγω απουσίας ILMB
3	ILMB+DOPB+DLMB	125	Μεγάλη εξωτερική μνήμη για δεδομένα Γρήγορη εσωτερική μνήμη (BRAM)	Για εφαρμογές που αρκούνται στην μνήμη BRAM για τον εκτελέσιμο κώδικα Μικρές ή μεσαίες μηχανές καταστάσεων
4	IOPB+ILMB+DOPB	110	Μεγάλη εξωτερική μνήμη Γρήγορη εσωτερική μνήμη για εντολές (BRAM)	Παρόμοιες με την περίπτωση 1
5	IOPB+DOPB	125	Μεγάλη εξωτερική μνήμη	Παρόμοιες με την περίπτωση 2
6	ILMB+DOPB	125	Μεγάλη εξωτερική μνήμη για δεδομένα Γρήγορη εσωτερική μνήμη για εντολές (BRAM)	Εφαρμογές όπου η BRAM είναι αρκετή για τον εκτελέσιμο κώδικα, αλλά όχι για τα δεδομένα Μικροί ελεγκτές, μικρές ή μεσαίες μηχανές καταστάσεων

Το σχήμα 2.9 απεικονίζει την πρώτη περίπτωση διαμόρφωσης, από την οποία βέβαια απορρέουν και οι υπόλοιπες.



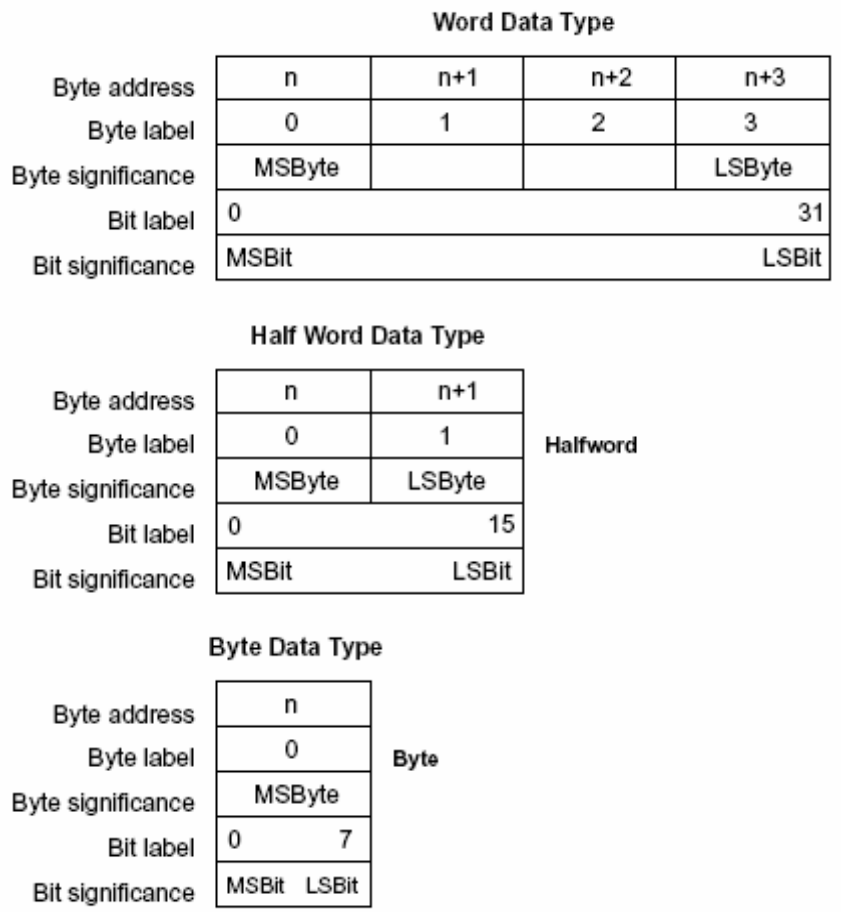
Σχήμα 2.9 – Η πρώτη περίπτωση διαμόρφωσης

Όπως αναφέρθηκε παραπάνω είναι δυνατόν σε κάθε έναν από τους έξι τρόπους διαμόρφωσης να υπάρχουν και συνδέσεις FSL μεταξύ του MicroBlaze και των περιφερειακών, που καθιστούν την επικοινωνία ταχύτερη και είναι ιδανικές για εφαρμογές επεξεργασίας σημάτων και ελέγχου δικτύων.



Σχήμα 2.10 – Η δομή των συνδέσεων FSL

Οι διάδρομοι δεδομένων του MicroBlaze μεταφέρουν δεδομένα τριών τύπων (σε παρένθεση οι αντιστοιχίσεις αυτών με assembly και C), word (data8 και char), half word (data16 και short, pointer) και byte (data32 και int, long int, enum, pointer). Οι τύποι δεδομένων και η συμβάσεις ονοματολογίας που χρησιμοποιούνται από τους διαδρόμους δεδομένων του MicroBlaze φαίνονται στο σχήμα 2.11.



Σχήμα 2.11 – Οι τύποι δεδομένων του MicroBlaze

Πηγές

1. Xilinx: “MicroBlaze Processor Reference Guide”

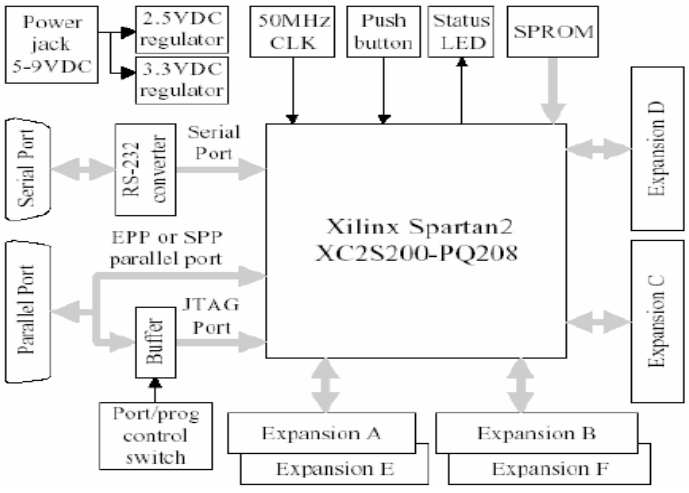
3. Αναπτυξιακές Κάρτες

3.1 Η κάρτα Digilab 2

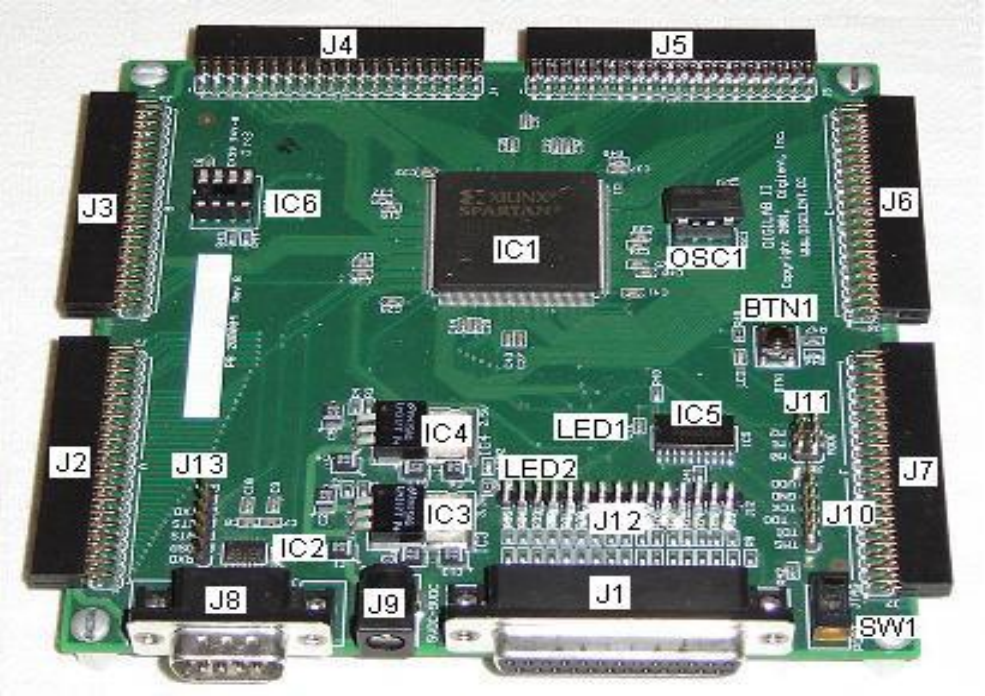
3.1.1 Κύρια μέρη και χαρακτηριστικά

Στο σχήμα 3.1 φαίνεται ένα διάγραμμα της κάρτας Digilab2, της Digilent, η οποία περιλαμβάνει τα παρακάτω μέρη (σε παρένθεση το όνομα της θέσης τους στην κάρτα και στο σχήμα 3.2):

- Το FPGA Spartan-II της Xilinx, που περιέχεται στο πακέτο XC2S200-PQ208 (IC1)
  - 200.000 πύλες, για λογική και RAM
  - 1.176 CLBs
  - Δεκατέσσερα μπλοκ RAMs των 4K-bit
  - Μέχρι 284 σήματα εισόδου / εξόδου, καθορισμένα από τον χρήστη
- Υποδοχή τροφοδοσίας 5-9V (J9) και ρυθμιστές τάσης LM317 για 3.3V (IC3) και 2.5V (IC4)
- Ταλαντωτή 50 MHz (OSC1)
- Παράλληλη θύρα (J1)
  - Ακροδέκτες ελέγχου για την παράλληλη θύρα (J12)
- Σειριακή θύρα RS-232 (J8)
  - Οδήγηση πέντε σημάτων, εκτός της γείωσης
  - Μετατροπέα τάσης MAX3386E, της MAXIM για την σειριακή θύρα (IC2)
  - Ακροδέκτες ελέγχου για την σειριακή θύρα (J13)
- Ένα LED συνδεδεμένο με το FPGA (LED1)
- LED ένδειξης λειτουργίας συνδεδεμένο με την τάση 2.5 V (LED2)
- Έναν πιεστικό διακόπτη συνδεδεμένο με το FPGA (BTN1)
- Έξι θύρες επέκτασης των 40 ακροδεκτών, A (J2), B (J3), C (J4), D (J5), E (J6), και F (J7)
- Υποδοχή για ολοκληρωμένο κύκλωμα σειριακής μνήμης SPROM (IC6)
- Έναν απομονωτή τριών καταστάσεων 74HC244 για τα σήματα προγραμματισμού του FPGA (IC5)
- Έναν διακόπτη για την εναλλαγή στον τρόπο προγραμματισμού του FPGA (SW1)
- Ακροδέκτες ελέγχου για την επιλογή του τρόπου προγραμματισμού του FPGA (J11)
- Ακροδέκτες προγραμματισμού JTAG (J10)



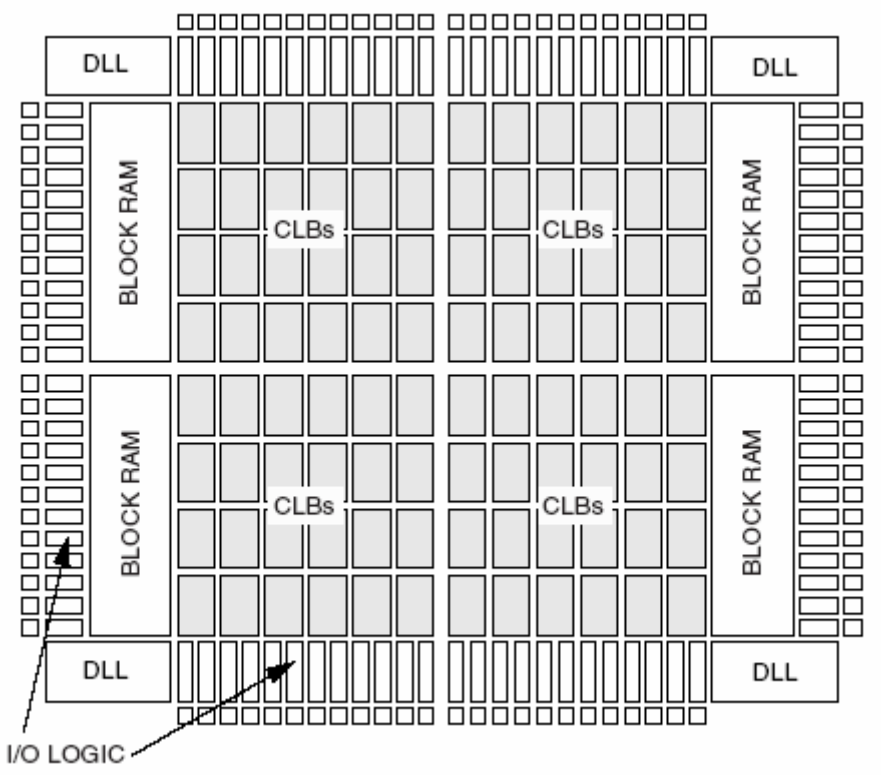
Σχήμα 3.1 – Σχεδιάγραμμα της κάρτας Digilab 2



Σχήμα 3.2 – Η κάρτα Digilab 2

3.1.2 Αναλυτική Περιγραφή

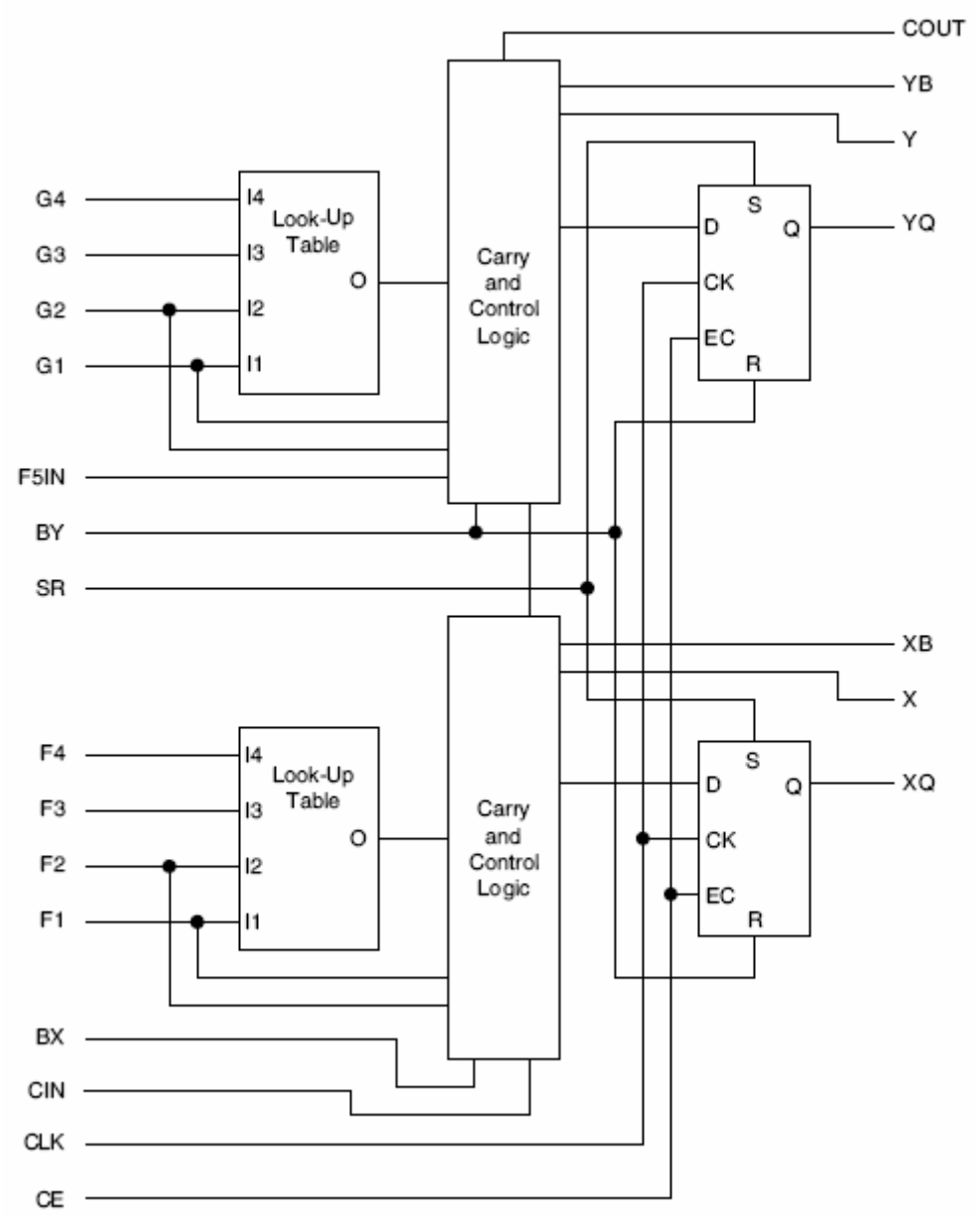
3.1.2.1 FPGA Spartan-II XC2S200-PQ208



Σχήμα 3.3 – Το FPGA XC2S15 της οικογένειας Spartan-II

Τα FPGAs, Field Programmable Logic Arrays, της οικογένειας Spartan-II βασίζονται σε τέσσερα θεμελιώδη μέρη:

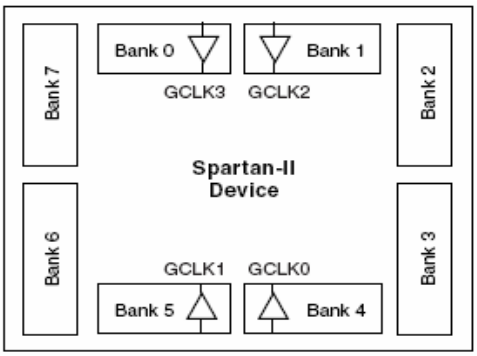
- Στα CLBs, Configurable Logic Blocks, διατάξεις που περιέχουν προγραμματιζόμενα κυκλώματα συνδυαστικής λογικής (Look-Up Table), και D Flip-Flops. Τα CLBs αποτελούνται από δύο όμοια slices, όπως αυτά του σχήματος 3.4. Βασίζονται σε δομές που λειτουργούν ως RAM, ROM, ή καταχωρητές ολίσθησης. Επιπλέον υπάρχουν ειδικά κυκλώματα για γρήγορη διάδοση κρατουμένου σε αριθμητικές πράξεις και τρικατάστατοι απομονωτές για τις εξόδους κάθε CLB. Η διάταξή τους μέσα στο FPGA είναι σε πίνακα, όπως αυτός του σχήματος 3.3 που είναι μεγέθους 8x12, ενώ ο πίνακας του XC2S200 που χρησιμοποιήθηκε στις εφαρμογές έχει μέγεθος 28x42, δηλαδή περιέχει συνολικά 1176 CLBs που αντιστοιχούν σε περίπου 200.000 πύλες.



Σχήμα 3.4 – Διάγραμμα ενός slice από τα δύο πανομοιότυπα που περιέχονται σε κάθε CLB

- Στα blocks εισόδου / εξόδου, Input/Output Blocks (IOB), που ελέγχουν την ροή των δεδομένων μεταξύ των ακροδεκτών I/O του FPGA και της εσωτερικής λογικής. Περιέχουν

απομονωτές εισόδου, εξόδου, και ελέγχου τρικατάστατης λογικής. Το XC2S200 δύναται να υποστηρίξει 284 ακροδέκτες, αλλά η συσκευασία στην οποία περιέχεται στο Digilab 2, PQ208, επιτρέπει μόνο 140. Τα IOBs χωρίζονται σε οκτώ ομάδες, οι οποίες μπορούν να λειτουργούν σε διαφορετικό πρότυπο εισόδου / εξόδου, αλλά και πάλι η συσκευασία που χρησιμοποιήθηκε ελαττώνει τις χρήσεις του FPGA, ενοποιώντας όλα τα IOBs σε μία ομάδα.



Σχήμα 3.5 – Οι οκτώ ομάδες των IOBs

- Στα block RAM για την αποθήκευση δεδομένων, μεγέθους 4 Kbit. Τα blocks είναι dual-port και διατάσσονται σε στήλες, με το ύψος τους να αντιστοιχεί σε τέσσερα CLB. Το πλάτος της κάθε πόρτας είναι μεταβλητό με την αναλογία μεταξύ διευθύνσεων και δεδομένων να κυμαίνεται όπως στον πίνακα 3.1.

Πίνακας 3.1 – Αναλογία διευθύνσεων και δεδομένων ενός block RAM

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

- Στους βρόχους κλειδώματος φάσης για την παραγωγή ρολογιού χωρίς καθυστέρηση, καθώς και για την παραγωγή νέου ρολογιού σε διπλάσια ή μικρότερες συχνότητες με συντελεστές, για την διαίρεση συχνότητας τους 1.5 , 2 , 2.5 , 3 , 4, 5 , 8 και 16. Όλα τα FPGA της οικογένειας Spartan-II, περιέχουν τέσσερις DLL, που μπορούν να συνδεθούν σε σειρά με άλλον έναν και να προκύψει ρολόι τετραπλάσιας συχνότητας από αυτό της εισόδου.

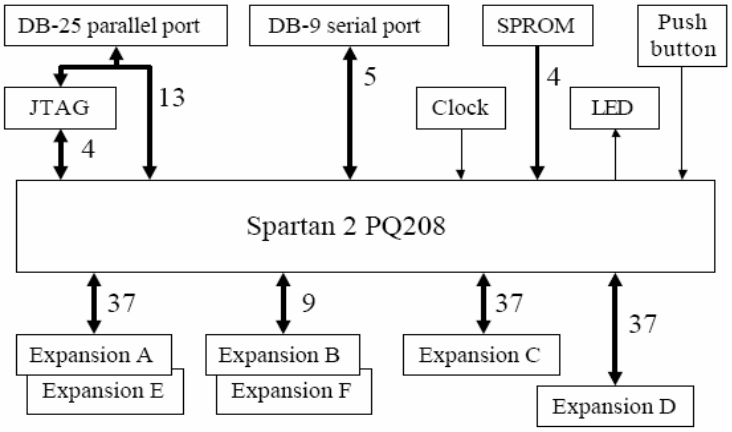
Μια σύνοψη των ιδιοτήτων των FPGAs της οικογένειας Spartan-II γίνεται στον πίνακα 3.2, ενώ στο σχήμα 3.6 και στον πίνακα 3.3 φαίνονται οι συνδέσεις του Spartan-II XC2S200-PQ208 στο Digilab 2 και οι λειτουργίες των ακροδεκτών του.

Πίνακας 3.2 – Οι ιδιότητες των FPGAs της οικογένειας Spartan-II

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O (1)	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

Πίνακας 3.3 – Οι συνδέσεις των ακροδεκτών του FPGA με το Digilab 2

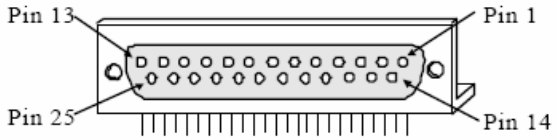
Pin #	Function	Pin #	Function	Pin #	Function	Pin #	Function
1	**GND**	53	**VCCO**	105	**VCCO**	157	**TDO**
2	**TMS**	54	**M2**	106	**PROG**	158	**GND**
3	PWT	55	**GND**	107	**INIT**	159	**TDI**
4	PINT	56	**MODE**	108	D20	160	C21
5	PD7	57	A14	109	D17	161	C20
6	PD6	58	A13	110	D18	162	C19
7	PD5	59	A12	111	D15	163	C18
8	PD4	60	A11	112	D16	164	C17
9	PD3	61	A10	113	D13	165	C16
10	PD2	62	A9	114	D14	166	C15
11	**GND**	63	A8	115	D11	167	C14
12	**VCCO**	64	**GND**	116	**GND**	168	C13
13	**VCCINT**	65	**VCCO**	117	**VCCO**	169	**GND**
14	PD1	66	**VCCINT**	118	**VCCINT**	170	**VCCO**
15	PD0	67	A7	119	D12	171	**VCCINT**
16	A40	68	A6	120	D9	172	C12
17	A39	69	A5	121	D10	173	C11
18	A38	70	A4	122	D7	174	C10
19	**GND**	71	LED1	123	D8	175	C9
20	A37	72	**GND**	124	**GND**	176	C8
21	A36	73	D39	125	D5	177	**GND**
22	A35	74	D40	126	D6	178	C7
23	A34	75	D37	127	D4	179	C6
24	A33	76	**VCCINT**	128	**VCCINT**	180	C5
25	**GND**	77	BTN1*	129	C40	181	C4
26	**VCCO**	78	**VCCO**	130	**VCCO**	182	B12*
27	A32	79	**GND**	131	**GND**	183	**GND**
28	**VCCINT**	80	CLK1*	132	C39	184	**VCCO**
29	A31	81	D38	133	C38	185	B11*
30	A30	82	D35	134	C37	186	**VCCINT**
31	A29	83	D36	135	C36	187	B10
32	**GND**	84	D33	136	C35	188	B9
33	A28	85	**GND**	137	**GND**	189	B8
34	A27	86	D34	138	C34	190	**GND**
35	A26	87	D31	139	C33	191	B7
36	A25	88	D32	140	C32	192	B6
37	A24	89	D29	141	C31	193	B5
38	**VCCINT**	90	D30	142	C30	194	B4
39	**VCCO**	91	**VCCINT**	143	**VCCINT**	195	RTS
40	**GND**	92	**VCCO**	144	**VCCO**	196	**VCCINT**
41	A23	93	**GND**	145	**GND**	197	**VCCO**
42	A22	94	D29	146	C29	198	**GND**
43	A21	95	D28	147	C28	199	CTS
44	A20	96	D25	148	C27	200	DSR
45	A19	97	D26	149	C26	201	TXD
46	A18	98	D23	150	C25	202	RXD
47	A17	99	D24	151	C24	203	PRS
48	A16	100	D21	152	C23	204	PAS
49	A15	101	D22	153	**DIN**	205	PDS
50	**M1**	102	D14	154	C22	206	PWE
51	**GND**	103	**GND**	155	**CCLK**	207	**TCK**
52	**MO**	104	**DONE**	156	**VCCO**	208	**VCCO**
* uses GCLK pin							



Σχήμα 3.6 – Οι συνδέσεις του Spartan-II με το Digilab 2

3.1.2.2 Παράλληλη θύρα

Η κάρτα Digilab 2 περιέχει μια παράλληλη θύρα 25 ακροδεκτών, που χρησιμοποιεί τον connector DB25, η οποία χρησιμοποιείται για τον προγραμματισμό του FPGA με το πρωτόκολλο JTAG ή για την μετακίνηση δεδομένων μεταξύ του FPGA και άλλων συσκευών με το πρωτόκολλο EPP (Enhanced Parallel Port). Η επιλογή μεταξύ των δύο λειτουργιών γίνεται μέσω του διακόπτη SW1. Όταν βρίσκεται στην θέση JTAG, τότε επιτρέπεται ο προγραμματισμός μέσω της παράλληλης θύρας, ενώ όταν βρίσκεται στην θέση PORT, τότε το κύκλωμα για τον προγραμματισμό απενεργοποιείται και η θύρα λειτουργία αποκλειστικά για μεταφορά δεδομένων με το πρωτόκολλο EPP. Η ταχύτητα μεταφοράς δεδομένων μπορεί να φτάσει έως τα 2 MB/sec. Στον πίνακα 3.4 αναγράφονται τα σήματα που οδηγεί η παράλληλη θύρα και η λειτουργία τους.



Σχήμα 3.7 Ο connector DB25 της παράλληλης θύρας

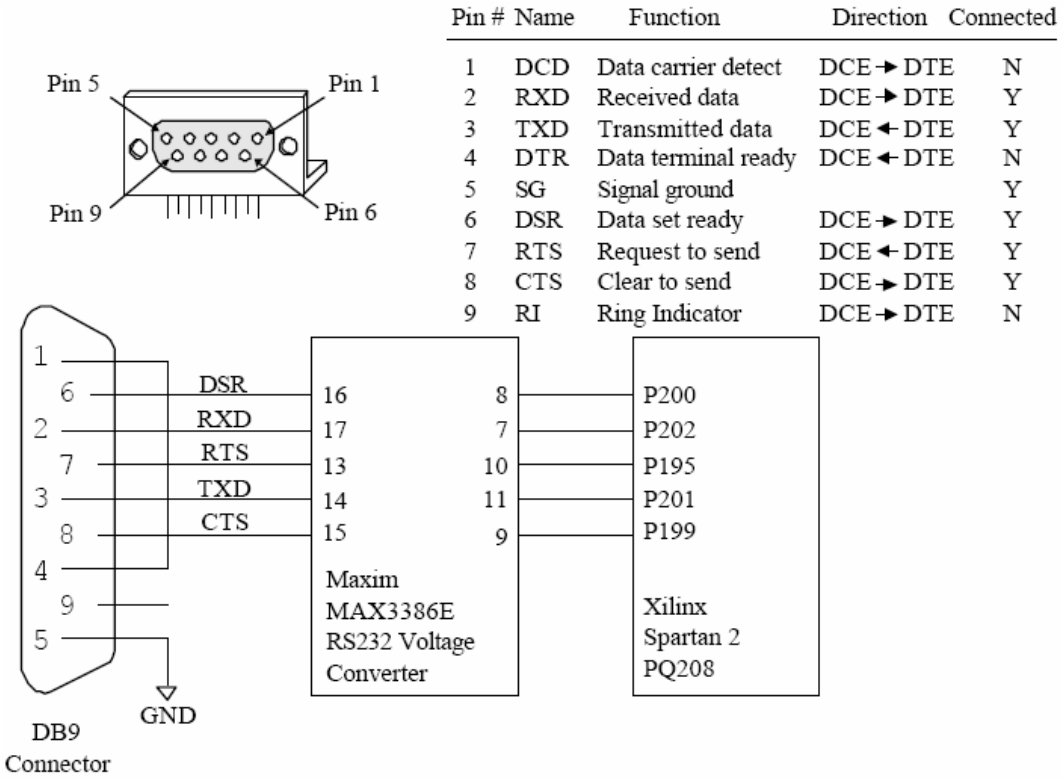
Πίνακας 3.4 – Τα σήματα της παράλληλης θύρας

Ακροδέκτης	Σήμα EPP	Λειτουργία
1	Write Enable (O)	Λογικό '0' για ανάγνωση, λογικό '1' για εγγραφή
2-9	Data Bus (B)	Γραμμές δεδομένων διπλής κατεύθυνσης
10	Interrupt (I)	Είσοδος διακοπής/επιβεβαίωσης
11	Wait (I)	Χειραψία διαδρόμου, επιβεβαίωση με λογικό '0'
12	Εφεδρικό	Αχρησιμοποίητο
13	Εφεδρικό	Αχρησιμοποίητο
14	Data Strobe (O)	Στο λογικό '0' για έγκυρα δεδομένα
15	Εφεδρικό	Αχρησιμοποίητο
16	Reset (O)	Μηδενισμός/αρχικοποίηση στο λογικό '0'
17	Address Strobe (O)	Στο λογικό '0' για έγκυρη διεύθυνση
18-25	GND	Γείωση

(O) = Σήμα εξόδου
(I) = Σήμα εισόδου
(B) = Σήμα διπλής κατεύθυνσης

3.1.2.3 Σειριακή θύρα

Η σειριακή θύρα χρησιμοποιεί τον connector DB9 και οδηγεί πέντε σήματα για την επίτευξη επικοινωνίας. Συνδέεται στο FPGA μέσω του ολοκληρωμένου κυκλώματος MAX3386E, της MAXIM, το οποίο αλλάζει τα επίπεδα τάσης από 0 V και 3.3 V, σε +6 V και -6 V, ώστε να είναι συμβατά με το πρωτόκολλο RS-232. Το συνολικό κύκλωμα επιτρέπει στη σειριακή θύρα να λειτουργήσει σωστά μέχρι τα 250 Kbps. Στο σχήμα 3.8 φαίνονται τα σήματα και οι συνδέσεις του κυκλώματος της σειριακής θύρας.



Σχήμα 3.8 – Τα σήματα της σειριακής θύρας και οι συνδέσεις τους

3.1.2.4 Ταλαντωτής

Ο ταλαντωτής στη θέση OSC1 λειτουργεί στα 50 MHz και είναι συνδεδεμένος με τον ακροδέκτη GCK0, πρώτη είσοδος ρολογιού, του FPGA. Με την χρήση των βρόχων κλειδώματος φάσης (DLL) του FPGA, η συχνότητα λειτουργίας μπορεί να φτάσει τα 200 MHz. Με την αντικατάσταση του συγκεκριμένου ταλαντωτή από άλλους με συχνότητες λειτουργίας από 32 KHz μέχρι 100 MHz, είναι δυνατόν να παραχθούν συχνότητες λειτουργίας από ένα μεγάλο εύρος τιμών.

3.1.2.5 Πιεστικός διακόπτης και LED

Η κάρτα Digilab 2, διαθέτει έναν πιεστικό διακόπτη και ένα LED για βασικές λειτουργίες κατάστασης και ελέγχου που υλοποιούνται δίχως την χρήση κάποιας περιφερειακής κάρτας. Αξίζει να σημειωθεί ότι ο πιεστικός διακόπτης είναι συνδεδεμένος με τον ακροδέκτη GCK1 του FPGA, που αποτελεί την δεύτερη είσοδο ρολογιού. Συνήθως ο πιεστικός διακόπτης χρησιμοποιείται για λειτουργίες μηδενισμού / αρχικοποίησης, ενώ το LED ως ένδειξη σωστού προγραμματισμού ή καλής λειτουργίας.

3.1.2.6 Μέθοδοι διαμόρφωσης του FPGA

Η κάρτα Digilab 2 προσφέρει δύο τρόπους διαμόρφωσης του FPGA. Ο πρώτος χρησιμοποιεί την παράλληλη θύρα και το πρωτόκολλο JTAG. Για να ακολουθηθεί αυτή η μέθοδος θα πρέπει ο διακόπτης SW1 να βρίσκεται στην θέση JTAG και οι ακροδέκτες ελέγχου J12 να είναι ανοιχτοκυκλωμένοι. Η διαμόρφωση γίνεται μέσω κατάλληλου προγράμματος της Xilinx (iMPACT). Αξίζει να αναφερθεί ότι μετά το πέρας της διαμόρφωσης ο διακόπτης SW1 μπορεί να μεταφερθεί στην θέση PORT και η παράλληλη θύρα να χρησιμοποιηθεί αποκλειστικά για μεταφορά δεδομένων.

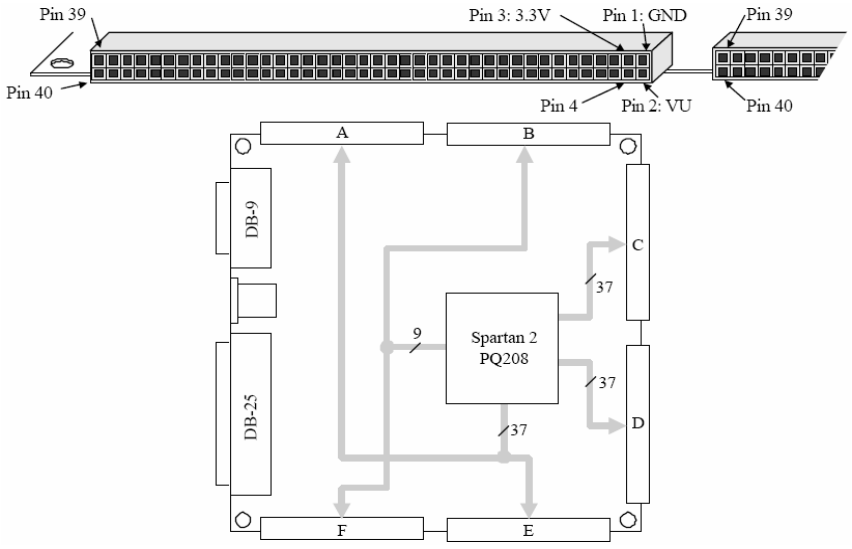
Ο άλλος τρόπος διαμόρφωσης του FPGA χρησιμοποιεί μία εξωτερική μνήμη SPROM οκτώ ακροδεκτών, στην οποία είναι ήδη αποθηκευμένη η πληροφορία διαμόρφωσης και η οποία τοποθετείται στην θέση IC6. Η επιλογή της μεθόδου αυτής γίνεται βραχυκυκλώνοντας τους ακροδέκτες ελέγχου J12 και μεταφέροντας τον διακόπτη SW1 στη θέση PORT. Η διαμόρφωση ξεκινά όταν εφαρμοστεί τάση στην κάρτα.

3.1.2.7 Τροφοδοσία

Το Digilab χρειάζεται διπλή τροφοδοσία. Το FPGA δουλεύει εσωτερικά στα 2.5 V, ενώ οι ακροδέκτες εισόδου / εξόδου στα 3.3 V. Οι τάσεις αυτές παρέχονται από τους ρυθμιστές τάσεις LM317 (IC3 και IC4), που δέχονται ως είσοδο, τροφοδοσία συνεχούς ρεύματος 5-9 V από την υποδοχή J9.

3.1.2.8 Θύρες επέκτασης

Οι έξι θύρες επέκτασης της κάρτας Digilab 2, συνδέονται με 122 ακροδέκτες του FPGA, όπως φαίνεται στο σχήμα 5. Οι συνδέσεις για τις θύρες A και E, καθώς και για τις B και F, είναι κοινές. Όλες οι θύρες έχουν τον ακροδέκτη 1 συνδεδεμένο με την γείωση, τον 3 με τα 3.3 V, και τον 2 με την τροφοδοσία των 5-9 V. Με τον τρόπο αυτό οι κάρτες που συνδέονται στις θύρες μπορούν είτε να χρησιμοποιήσουν απευθείας τα 3.3 V, είτε δικό τους ρυθμιστή, με είσοδο τα 5-9 V, όπως κάνει η κάρτα Digilab DIO 2, της οποίας η περιγραφή ακολουθεί. Στον πίνακα 3 αναγράφονται οι ακροδέκτες των θυρών επέκτασης και οι συνδέσεις τους με το FPGA, όπου αυτές υπάρχουν.



Σχήμα 3.9 – Οι θύρες επέκτασης και οι συνδέσεις τους

Πίνακας 3.5 – Οι ακροδέκτες των θυρών επέκτασης και οι συνδέσεις τους με το FPGA

A&E connector			B&F connector			C connector			D connector		
Pin	Signal	S-II pin	Pin	Signal	S-II pin	Pin	Signal	S-II pin	Pin	Signal	S-II pin
1	GND	-	1	GND	-	1	GND	-	1	GND	-
2	VU	-	2	VU	-	2	VU	-	2	VU	-
3	VDD33	-	3	VDD33	-	3	VDD33	-	3	VDD33	-
4	A4	70	4	B4	194	4	C4	181	4	D4	127
5	A5	69	5	B5	193	5	C5	180	5	D5	125
6	A6	68	6	B6	192	6	C6	179	6	D6	126
7	A7	67	7	B7	191	7	C7	178	7	D7	122
8	A8	63	8	B8	189	8	C8	176	8	D8	123
9	A9	62	9	B9	188	9	C9	175	9	D9	120
10	A10	61	10	B10	187	10	C10	174	10	D10	121
11	A11	60	11	B11	185*	11	C11	173	11	D11	115
12	A12	59	12	B12	182*	12	C12	172	12	D12	119
13	A13	58	13	B13	-	13	C13	168	13	D13	113
14	A14	57	14	B14	-	14	C14	167	14	D14	114
15	A15	49	15	B15	-	15	C15	166	15	D15	111
16	A16	48	16	B16	-	16	C16	165	16	D16	112
17	A17	47	17	B17	-	17	C17	164	17	D17	109
18	A18	46	18	NC	-	18	C18	163	18	D18	110
19	A19	45	19	NC	-	19	C19	162	19	D19	102
20	A20	44	20	NC	-	20	C20	161	20	D20	108
21	A21	43	21	NC	-	21	C21	160	21	D21	100
22	A22	42	22	NC	-	22	C22	154	22	D22	101
23	A23	41	23	NC	-	23	C23	152	23	D23	98
24	A24	37	24	NC	-	24	C24	151	24	D24	99
25	A25	36	25	NC	-	25	C25	150	25	D25	96
26	A26	35	26	NC	-	26	C26	149	26	D26	97
27	A27	34	27	NC	-	27	C27	148	27	D27	94
28	A28	33	28	NC	-	28	C28	147	28	D28	95
29	A29	31	29	NC	-	29	C29	146	29	D29	89
30	A30	30	30	NC	-	30	C30	142	30	D30	90
31	A31	29	31	NC	-	31	C31	141	31	D31	87
32	A32	27	32	NC	-	32	C32	140	32	D32	88
33	A33	24	33	NC	-	33	C33	139	33	D33	84
34	A34	23	34	NC	-	34	C34	138	34	D34	86
35	A35	22	35	NC	-	35	C35	136	35	D35	82
36	A36	21	36	NC	-	36	C36	135	36	D36	83
37	A37	20	37	NC	-	37	C37	134	37	D37	75
38	A38	18	38	NC	-	38	C38	133	38	D38	81
39	A39	17	39	NC	-	39	C39	132	39	D39	73
40	A40	16	40	NC	-	40	C40	129	40	D40	74

* uses GCLK pin

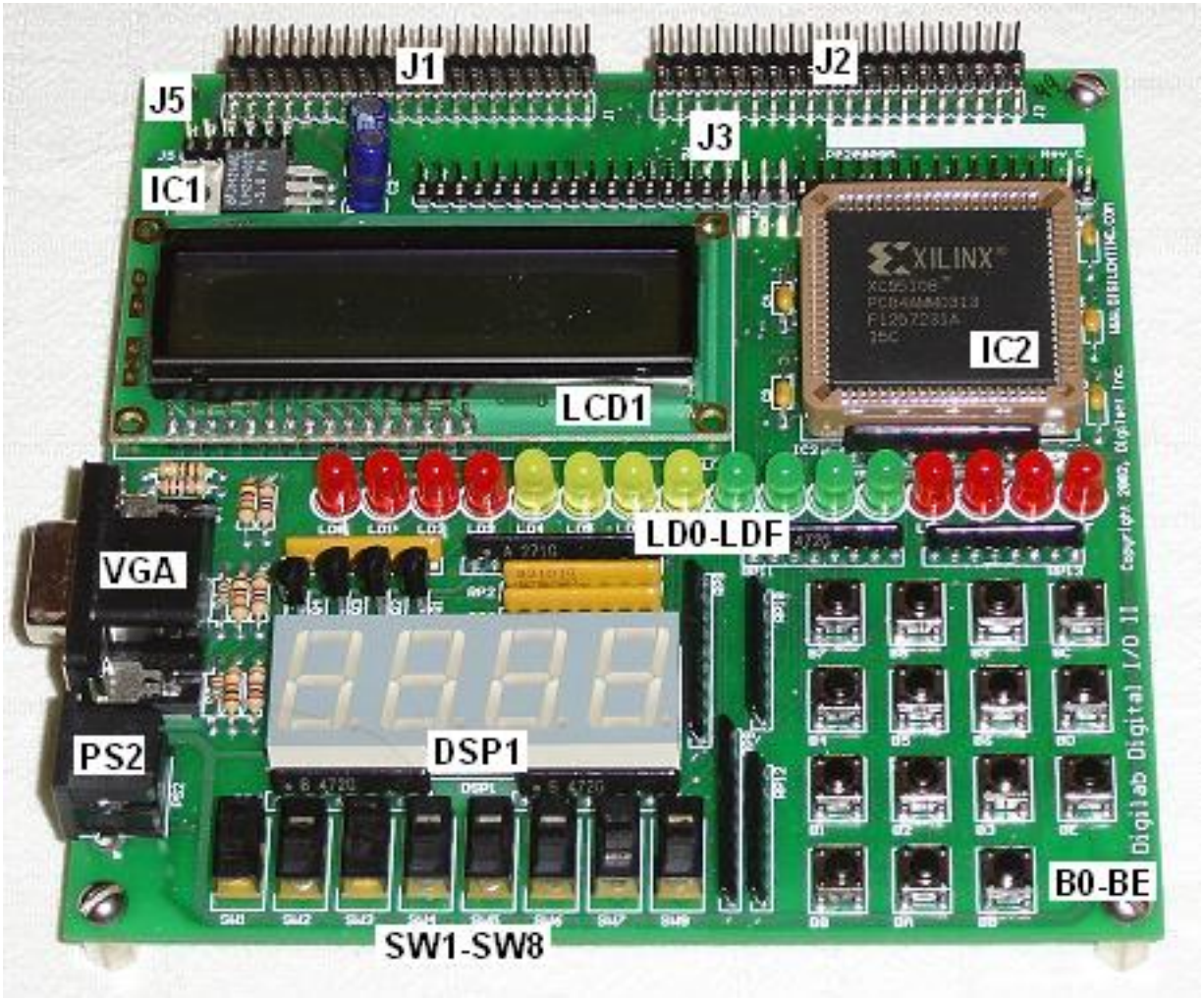
3.2 Η κάρτα Digilab Digital I/O 2 (DIO2)

3.2.1 Κύρια μέρη και χαρακτηριστικά

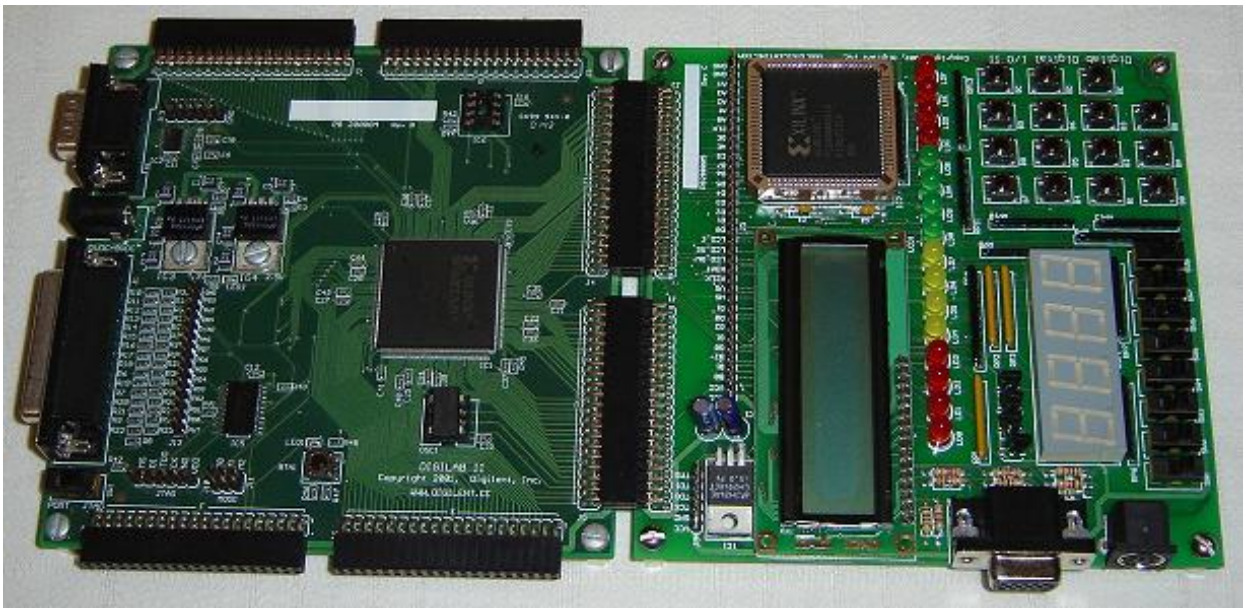
Στο σχήμα 3.10 φαίνεται η κάρτα επέκτασης Digilab Digital I/O 2 (DIO2), της Digilent. Η κάρτα DIO2 συνδέεται στις θύρες επέκτασης C και D, όπως φαίνεται στο σχήμα 3.11, προκειμένου να υλοποιηθούν βασικές λειτουργίες εισόδου / εξόδου. Στην κάρτα DIO2

περιλαμβάνονται τα ακόλουθα μέρη (σε παρένθεση το όνομα της θέσης τους στην κάρτα και στο σχήμα 3.10):

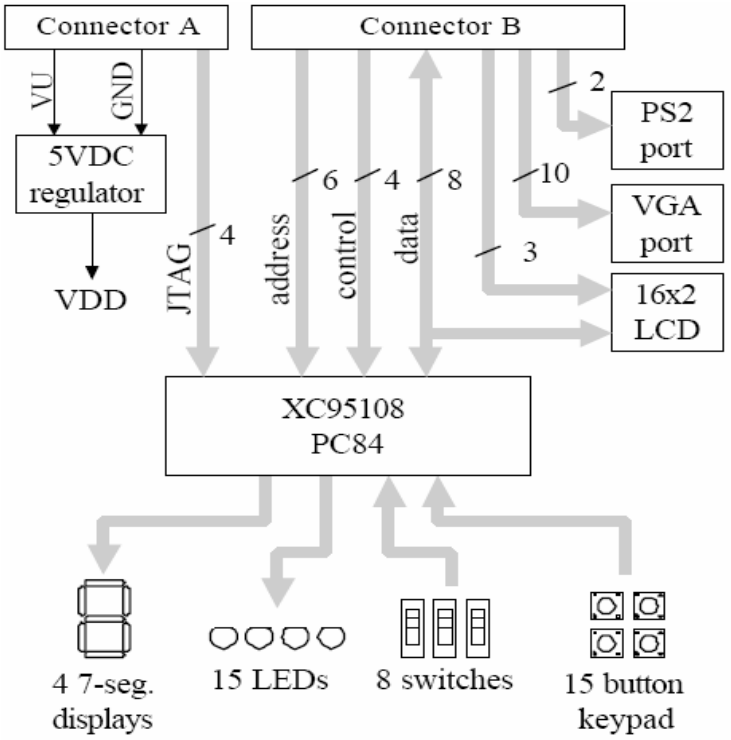
- Ένα ολοκληρωμένο Complex Programmable Logic Device, CPLD, της Xilinx και συγκεκριμένα το XC95108-PC84 (IC2)
- Ένας ρυθμιστή τάσης LM2940 για 5 V DC (IC1)
- Δύο connectors των 40 ακροδεκτών, οι A (J1) και B (J2), για σύνδεση σε θύρα επέκτασης
- Μία θύρα εξόδου VGA με χρώματα βάθους 8 bit (VGA)
- Μία θύρα τύπου PS/2 για ποντίκι / πληκτρολόγιο (PS2)
- Μία οθόνη υγρών κρυστάλλων, Liquid Crystal Display (LCD), 16x2 χαρακτήρων με ενσωματωμένο ελεγκτή (LCD1)
- Μία οθόνη με τέσσερα ψηφία των επτά τμημάτων (DSP1)
- Δεκαέξι LED, χωρισμένα σε ομάδες χρωμάτων των τεσσάρων LED (LD0-LDF)
- Δεκαπέντε πιεστικοί διακόπτες (B0-BE)
- Οκτώ διακόπτες (SW1-SW8)
- Ακροδέκτες ελέγχου για τα σήματα που προέρχονται από το Digilab 2 (J3) και για τα σήματα JTAG (J5)



Σχήμα 3.10 – Η κάρτα DIO2



Σχήμα 3.11 – Σύνδεση των καρτών Digilab 2 και DIO2



Σχήμα 3.12 – Διάγραμμα της κάρτας DIO2

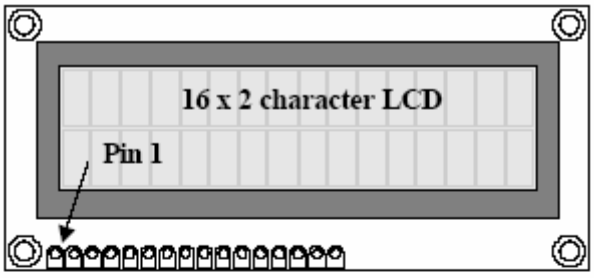
3.2.2 Αναλυτική περιγραφή

3.2.2.1 Οθόνη υγρών κρυστάλλων (LCD)

Η LCD οθόνη της κάρτας DIO2 είναι η RC1602D της Okaya. Είναι τοποθετημένη σε ξεχωριστή πλακέτα μαζί με τον ελεγκτή της, KS0066U της Samsung, που οδηγεί τα σήματα ελέγχου και δεδομένων που προορίζονται για αυτήν. Η πλακέτα έχει δεκαέξι ακροδέκτες, εκ των οποίων οι τρεις (1-3) που τροφοδοτούν με τάση τον ελεγκτή και την οθόνη συνδέονται απευθείας με την τροφοδοσία. Η οθόνη διαθέτει εσωτερικό LED οπίσθιου φωτισμού της

οθόνης, όμως οι δύο ακροδέκτες που το οδηγούν είναι ασύνδετοι στην κάρτα DIO2 και συνεπώς δεν μπορεί να ανάψει. Οκτώ (7-14) από τους δεκαέξι ακροδέκτες συνδέονται με τον διάδρομο δεδομένων που συνδέει την DIO2 με το Digilab 2 (D0-D7), και οι τρεις εναπομείναντες ακροδέκτες, συνδέονται με τα σήματα ελέγχου R/W (5), RS (4) και E (6) που έρχονται απευθείας από το FPGA της Digilab 2.

Η οθόνη LCD χρησιμοποιεί κωδικούς χαρακτήρων ASCII 32. Ο ελεγκτής έχει προ-αποθηκευμένους σε ROM γεννήτρια χαρακτήρων, Character Generator ROM (CGROM), 208 χαρακτήρες σε πρότυπο απεικόνισης 5x8 ψηφίδων (pixels), ενώ μπορεί να αποθηκεύσει οκτώ ορισμένους από τον χρήστη χαρακτήρες, σε πρότυπο 5x8, σε μία μνήμη RAM, Character Generator RAM (CGRAM). Παρότι η οθόνη έχει δυνατότητα απεικόνισης 16x2 (χαρακτήρες x γραμμές), ο ελεγκτής έχει μια μνήμη οθόνης, Display Data RAM (DDRAM), η οποία αποθηκεύει 80 κωδικούς χαρακτήρων σε διάταξη 40x2. Οι κωδικοί που αποθηκεύονται στην DDRAM είναι δείκτες στην CGROM ή στην CGRAM και για την πρώτη γραμμή οι διευθύνσεις των χαρακτήρων είναι από 00H έως 27H, ενώ για την δεύτερη γραμμή είναι από 40H έως 67H. Παρότι δεν μπορούν να εμφανιστούν όλοι οι χαρακτήρες κάθε γραμμής της μνήμης DDRAM ταυτόχρονα, οι εμφανιζόμενες γραμμές μπορούν να ολισθαίνουν είτε αριστερά είτε δεξιά, θέτοντας την κατάλληλη εντολή στον καταχωρητή εντολών IR (Instruction Register), όπως φαίνεται στο σχήμα 3.13.



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Διευθύνσεις της DDRAM που εμφανίζονται

01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

Διευθύνσεις της DDRAM που εμφανίζονται μετά από αριστερή ολίσθηση

27	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
67	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E

Διευθύνσεις της DDRAM που εμφανίζονται μετά από δεξιά ολίσθηση

Σχήμα 3.13 – Η εμφάνιση του LCD με βάση τις διευθύνσεις της DDRAM και των εντολών ολίσθησης

Η ολίσθηση είναι μία από τις εντολές που υποστηρίζει ο ελεγκτής της LCD οθόνης. Οι εντολές σχηματίζονται από τα σήματα ελέγχου, RS και R/W, και από τον διάδρομο δεδομένων D0-D7. Όλες οι εντολές αναγράφονται στον πίνακα 3.6 που ακολουθεί.

Πίνακας 3.6 – Οι εντολές που απευθύνονται στον ελεγκτή της οθόνης LCD

Εντολή	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Μέγιστος χρόνος εκτέλεσης
Καθαρισμός οθόνης	0	0	0	0	0	0	0	0	0	1	1520μs
Επιστροφή στην αρχική θέση	0	0	0	0	0	0	0	0	1	X	1520μs
Καθορισμός τρόπου εισόδου	0	0	0	0	0	0	0	1	D/I	C/D	37μs
Έλεγχος οθόνης	0	0	0	0	0	0	1	D	C	B	37μs
Ολίσθηση δρομέα ή οθόνης	0	0	0	0	0	1	C/D	R/L	X	X	37μs
Καθορισμός τρόπου λειτουργίας	0	0	0	0	1	DL	N	F	X	X	37μs
Καθορισμός διεύθυνσης CGRAM	0	0	0	1	$ADDR_{CGRAM}$						37μs
Καθορισμός διεύθυνσης DDRAM	0	0	1	$ADDR_{DDRAM}$							37μs
Ανάγνωση BF και διεύθυνσης	0	1	BF	ADDR							1μs
Εγγραφή δεδομένων	1	0	DATA								37μs
Ανάγνωση δεδομένων	1	1	DATA								37μs

Σύμβολο	Επεξήγηση
D/I	Αυτόματη αλλαγή διεύθυνσης : Αύξηση (1) / Μείωση (0)
C/D	Δρομέας (1) / Οθόνη (0)
D	Οθόνη : αναμμένη (1) / Οθόνη σβηστή (0)
C	Δρομέας σε μορφή υπογράμμιση : ενεργοποιημένος (1) / απενεργοποιημένος (0)
B	Δρομέας σε μορφή χαρακτήρα που αναβοσβήνει : ενεργοποιημένος (1) / απενεργοποιημένος (0)
R/L	Ολίσθηση : Δεξιά (1) / Αριστερά (0)
DL	Διάδρομος δεδομένων : 8-bit (1) / 4-bit (0)
N	Αριθμός γραμμών στην οθόνη : Δύο (1) / Μία (0)
F	Μέγεθος Font : 5x11 (1) / 5x8 (0)
BF	Σημαία : «απασχολημένο» (1) / «έτοιμο» (0)

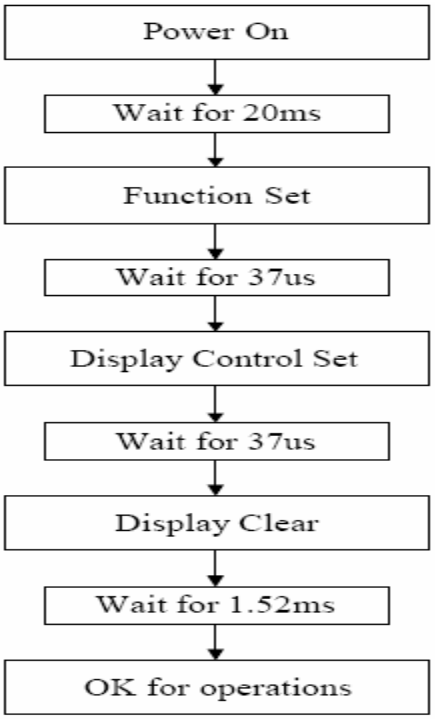
Όταν δίνεται τάση στην τροφοδοσία, ο ελεγκτής KS0066U αρχικοποιείται αυτόματα ως εξής:

- Εκτελείται η εντολή καθαρισμού της οθόνης, δηλαδή γράφεται ο κενός χαρακτήρας, κωδικός ASCII 32, σε όλες τις θέσεις μνήμης της DDRAM
- Εκτελείται η εντολή καθορισμού τρόπου λειτουργίας με επιλογές για διάδρομο δεδομένων 8 bit, οθόνη μίας γραμμής και πρότυπο απεικόνισης 5x8 ψηφίδες
- Εκτελείται η εντολή ελέγχου οθόνης με όλες τις επιλογές, οθόνη, δρομέας σε μορφή υπογράμμισης και δρομέας σε μορφή χαρακτήρα που αναβοσβήνει, απενεργοποιημένες
- Εκτελείται η εντολή καθορισμού τρόπου εισόδου με επιλογές να αυξάνει αυτόματα η διεύθυνση μετά την εισαγωγή νέου χαρακτήρα, να μετακινείται ο δρομέας και να μην πραγματοποιείται ολίσθηση της οθόνης

Σε όλη την παραπάνω διαδικασία η σημαία BF (απασχολημένο) παραμένει ενεργή, ενώ μόλις ολοκληρωθεί η αρχικοποίηση η σημαία αυτή μηδενίζεται. Για να αλλάξει η παραπάνω διαμόρφωση της οθόνης LCD, θα πρέπει να ακολουθηθούν τα ακόλουθα βήματα, που περιέχουν και διαστήματα καθυστέρησης:

- Μετά από διάστημα τουλάχιστον 20 ms μπορεί να δοθεί η εντολή του τρόπου λειτουργίας (function set), ώστε να καθοριστεί το πλάτος του διαδρόμου δεδομένων, ο αριθμός των γραμμών και το πρότυπο απεικόνισης (font)
- Έπειτα χρειάζεται άλλη μία καθυστέρηση των 37us προτού δοθεί η εντολή ελέγχου οθόνης (display control set), που εκκινεί την οθόνη, θέτει τον δρομέα σε μορφή υπογράμμισης ή όχι, καθώς και αν θα συμπεριφέρεται σαν χαρακτήρας που αναβοσβήνει ή όχι.
- Μετά από ακόμα 37us μπορεί να δοθεί η εντολή καθαρισμού της οθόνης (display clear). Προτού να είναι δυνατόν να γραφούν δεδομένα στην DDRAM ή να εκτελεστεί κάποια από

τις υπόλοιπες εντολές, χρειάζεται ένα διάστημα περίπου 1.52ms ή έως ότου η σημαία BF απενεργοποιηθεί.



Σχήμα 3.14 – Η απαραίτητη διαδικασία κατά την εκκίνηση

Το σήμα ελέγχου E, αν και δεν εμφανίζεται σε καμία από τις εντολές ελέγχου, είναι απαραίτητο για την μεταφορά πληροφορίας μέσω του διαδρόμου δεδομένων. Όταν έχει την λογική τιμή ‘1’, και δεδομένου ότι και το R/W έχει την λογική τιμή ‘1’, επιτρέπει την αποστολή πληροφορίας μέσω του διαδρόμου δεδομένων, ενώ όταν έχει την λογική τιμή ‘1’, και το R/W την λογική τιμή ‘0’, επιτρέπει την εγγραφή στη μνήμη. Οι χρονικοί περιορισμοί για την λειτουργία του ελεγκτή αναγράφονται στον πίνακα 3.7

Πίνακας 3.7 – Οι χρονικοί περιορισμοί στην λειτουργία του ελεγκτή

Ελάχιστη περίοδος σήματος E	500 ns
Ελάχιστος χρόνος υψηλής στάθμης παλμού του σήματος E	230 ns
Μέγιστος χρόνος ανόδου / καθόδου για το σήμα E	20 ns
Ελάχιστος χρόνος προετοιμασίας σημάτων RS, R/W, DB0-7 πριν την πτώση του σήματος E	80 ns
Ελάχιστος χρόνος συγκράτησης σημάτων RS, R/W, DB0-7 μετά την πτώση του σήματος E	10 ns
Μέγιστη καθυστέρηση δεδομένων εξόδου (μετά την άνοδο του σήματος E)	120 ns
Ελάχιστος χρόνος διατήρησης δεδομένων εξόδου (μετά την πτώση του σήματος E)	5 ns

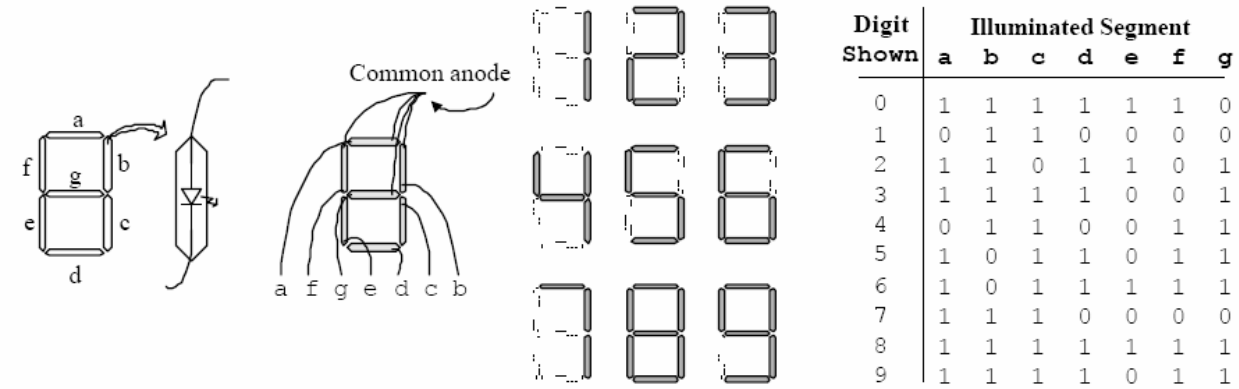
3.2.2.2 LEDs

Η κάρτα DIO2 διαθέτει δεκαέξι LEDs χωρισμένα σε τέσσερις διαδοχικές χρωματιστές ομάδες, κόκκινο, κίτρινο, πράσινο, κόκκινο, των τεσσάρων LED. Οι άνοδοι των LED είναι συνδεδεμένοι με την τροφοδοσία μέσω αντιστάσεων των 270 Ω, ενώ οι κάθοδοι οδηγούνται με σήματα που προέρχονται από το CPLD. Τα σήματα αυτά οδηγούνται από καταχωρητές που αντιστοιχούν στις διευθύνσεις 04H και 05H. Οι έξοδοι των καταχωρητών αντιστρέφονται πριν φτάσουν στις καθόδους, συνεπώς δεν υπάρχει η ανάγκη να χρησιμοποιηθεί αρνητική λογική, αν

και καθοδηγούμε καθόδους, και το λογικό ‘1’ ανάβει τα LED. Αναλυτικά ο τρόπος ελέγχου των LEDs, μέσω των διευθύνσεων 04H και 05H, αναφέρεται στην περιγραφή του CPLD.

3.2.2.3 Οθόνη με τέσσερα ψηφία των επτά τμημάτων

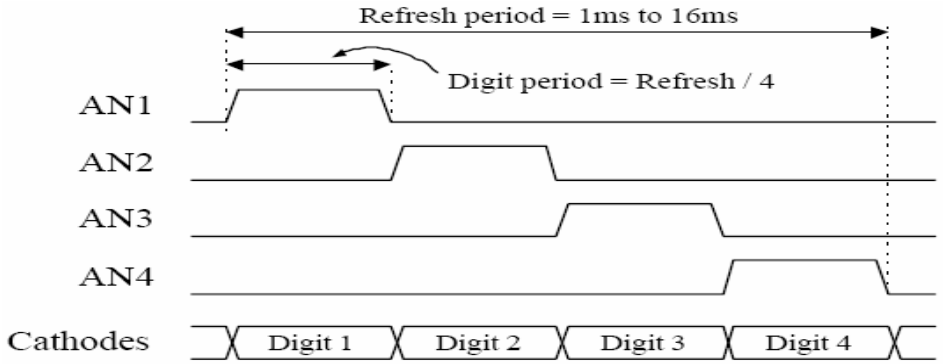
Η κάρτα DIO2 διαθέτει μία οθόνη επτά τμημάτων που αποτελείται από τέσσερα ψηφία. Κάθε ψηφίο έχει ακροδέκτη κοινής ανόδου για όλα τα τμήματα που το αποτελούν (σήματα AN1, AN2, AN3 και AN4), ενώ τα τμήματα της ίδιας θέσης όλων των ψηφίων έχουν κοινές καθόδους (σήματα CA, CB, CC, CD, CE, CF, CG και DP).



Digit Shown	Illuminated Segment a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Σχήμα 3.15 – Οι συνδέσεις των τμημάτων και οι σχηματισμοί των ψηφίων

Με αυτόν τον τρόπο σύνδεσης η εμφάνιση όλων των ψηφίων επιτυγχάνεται οδηγώντας διαδοχικά την κοινή άνοδο του κάθε ψηφίου σε λογικό ‘1’ και ταυτόχρονα τις επιθυμητές καθόδους των τμημάτων που πρέπει να ανάψουν σε λογικό ‘0’. Η ακολουθία αυτή πρέπει να επαναλαμβάνεται συνεχώς σε συχνότητα τουλάχιστον 60Hz ώστε να μην είναι αντιληπτή από το ανθρώπινο μάτι, φροντίζοντας παράλληλα να υπάρχει συγχρονισμός στην οδήγηση των ανόδων και των καθόδων. Η παραπάνω διαδικασία είναι ήδη υλοποιημένη σε έναν ελεγκτή του CPLD και το μόνο που απαιτείται είναι ένα ρολόι εισόδου τουλάχιστον 240Hz. Ο ελεγκτής οδηγεί την οθόνη με βάση τις τιμές που έχει στους καταχωρητές των οκτώ bit που είναι υλοποιημένοι στο εσωτερικό του. Οι διευθύνσεις των δύο καταχωρητών είναι 06H και 07H και σε αυτούς αποθηκεύονται τέσσερις αριθμοί των τεσσάρων bit, που προορίζονται για εμφάνιση.

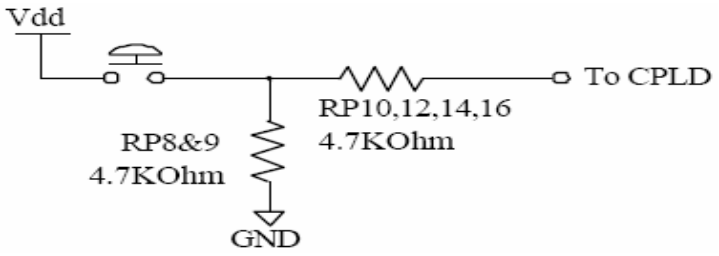


Σχήμα 3.16 – Διάγραμμα χρονισμού της οθόνης επτά τμημάτων

Δυστυχώς, ο ελεγκτής δίνει την δυνατότητα εμφάνισης μόνο των δεκαεξαδικών ψηφίων 0 έως F, δεν οδηγεί τις υποδιαστολές και δεν δίνει την δυνατότητα να μην εμφανίζονται και τα τέσσερα ψηφία ταυτόχρονα.

3.2.2.4 Πιεστικοί Διακόπτες

Οι δεκαπέντε πιεστικοί διακόπτες της κάρτας DIO2 έχουν την μια τους άκρη συνδεδεμένη με την τροφοδοσία και την άλλη με την γη μέσω αντίστασης των 4.7 ΚΩ. Αντίστοιχα σήματα από τους ακροδέκτες του CPLD (BTN0-BTNE), συνδέονται ανάμεσα στον διακόπτη και την αντίσταση. Με τον τρόπο αυτό όσο οι διακόπτες είναι ελεύθεροι, οι ακροδέκτες συνδέονται με την γείωση, δηλαδή λογικό '0', ενώ 'όταν πατηθούν συνδέονται με την τροφοδοσία, δηλαδή λογικό '1'.

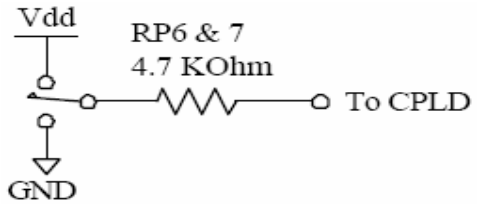


Σχήμα 3.17 – Το κύκλωμα των πιεστικών διακόπτών

Η κατάσταση των πιεστικών διακοπτών μπορεί να διαβαστεί από τις θέσεις μνήμης 00H και 01H CPLD, όπου ενώ για τα LEDs και την οθόνη των επτά τμημάτων χρησιμοποιούνταν καταχωρητές, εδώ διαβάζονται απευθείας τα σήματα.

3.2.2.5 Διακόπτες

Η κάρτα DIO2 διαθέτει οκτώ διακόπτες. Οι διακόπτες έχουν τρεις ακροδέκτες, εκ των οποίων οι δύο είναι συνδεδεμένοι με την τροφοδοσία και την γείωση αντίστοιχα, ενώ ο τρίτος συνδέεται με το αντίστοιχο σήμα του CPLD (SW1-SW8) μέσω μιας αντίστασης 4.7 ΚΩ.



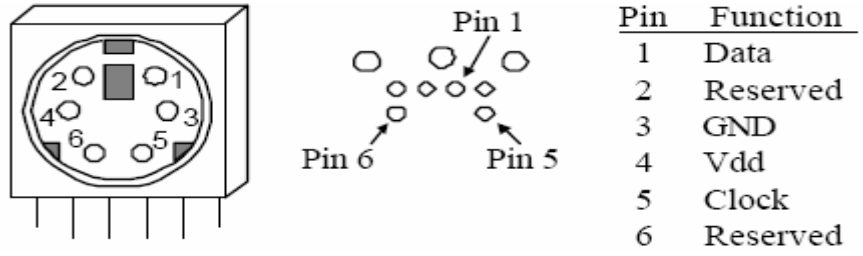
Σχήμα 3.18 – Το κύκλωμα των διακοπτών

Τα σήματα των διακοπτών μπορούν να διαβαστούν από το CPLD, μέσω των διευθύνσεων 02H και 03H (και οι δύο διευθύνσεις αντιστοιχούν στα ίδια σήματα).

3.2.2.6 Θύρα PS/2

Η κάρτα DIO2 διαθέτει έναν connector έξι ακροδεκτών που μπορεί να φιλοξενήσει ένα ποντίκι ή ένα πληκτρολόγιο τύπου PS/2. Τόσο το ποντίκι όσο και το πληκτρολόγιο χρησιμοποιούν διπλό σειριακό διάδρομο δεδομένων, που περιέχει ρολόι και δεδομένα, για να επικοινωνήσουν με τη συσκευή που τα φιλοξενεί. Και οι δύο οδηγούν το διάδρομο με ίδιους χρονισμούς σήματος και χρησιμοποιούν λέξεις των έντεκα bit που περιέχουν ένα bit εκκίνησης, ένα bit λήξης και ένα bit περιττής ισοτιμίας. Παρόλα αυτά τα πακέτα δεδομένων είναι οργανωμένα διαφορετικά για το πληκτρολόγιο και για το ποντίκι, ενώ η διασύνδεση του

πληκτρολογίου επιτρέπει την μεταφορά δεδομένων και προς τις δύο κατευθύνσεις (ώστε η συσκευή που το φιλοξενεί να μπορεί να φωτίσει τα LED κατάστασης στο πληκτρολόγιο).

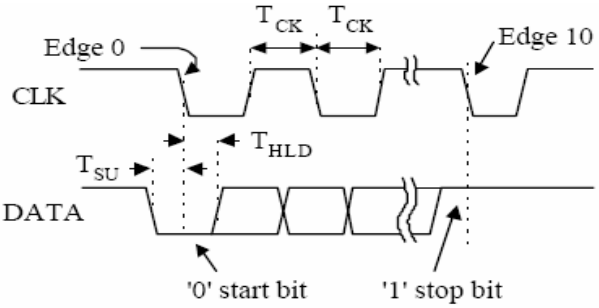


Pin	Function
1	Data
2	Reserved
3	GND
4	Vdd
5	Clock
6	Reserved

Σχήμα 3.19 – Ο connector της θύρας PS/2 και οι ακροδέκτες του

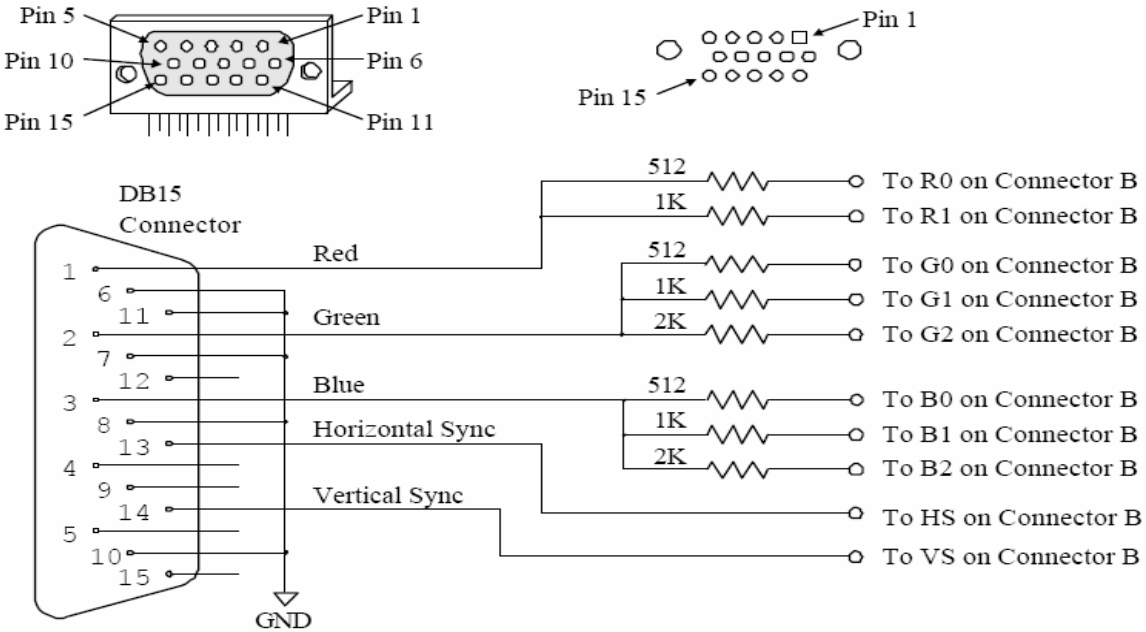
Οι χρονισμοί του διαδρόμου δεδομένων φαίνονται στο σχήμα 3.20. Τα σήματα του ρολογιού και των δεδομένων οδηγούνται μόνο όταν συμβαίνουν μεταφορές δεδομένων και σε κάθε άλλη περίπτωση παραμένουν στην άεργη κατάσταση οδηγώντας το λογικό '1'. Οι χρονισμοί καθορίζουν τις απαιτήσεις του σήματος για την επικοινωνία του ποντικιού με την συσκευή και την διπλής κατεύθυνσης επικοινωνία του πληκτρολογίου.

Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us



Σχήμα 3.20 – Οι χρονισμοί των σημάτων για τον διάδρομο δεδομένων της θύρας PS/2

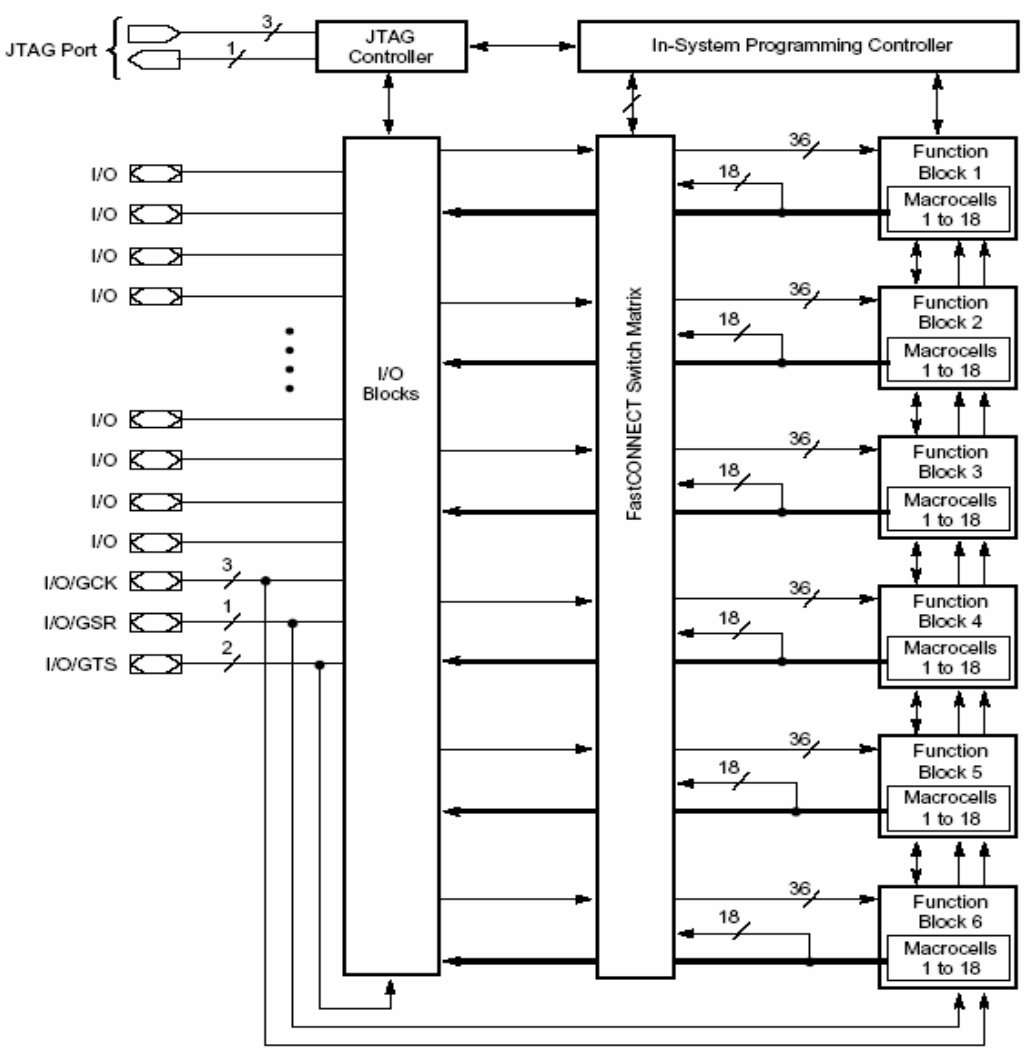
3.2.2.7 Θύρα VGA



Σχήμα 3.21 – Ο connector DB15 και τα σήματα της θύρας VGA

Η θύρα εξόδου VGA της κάρτας DIO2, χρησιμοποιεί τον connector DB15, των δεκαπέντε ακροδεκτών. Τα πέντε καθορισμένα σήματα VGA, κόκκινο (R), πράσινο (G), μπλε (B), ο οριζόντιος συγχρονισμός (HS) και ο κάθετος συγχρονισμός (VS) δρομολογούνται απευθείας από τον connector B στον connector της VGA, δίχως να περνάνε από το CPLD. Ένα δίκτυο αντιστάσεων χρησιμοποιείται για να παρέχει χρώματα βάθους οκτώ bit, με τρία bit για το μπλε, τρία bit για το πράσινο και δύο bit για το κόκκινο, λόγω του ότι το ανθρώπινο μάτι είναι λιγότερο ευαίσθητο στο κόκκινο. Το δίκτυο αντιστάσεων χρησιμοποιεί τον τερματισμό του καλωδίου VGA των 75Ω για να εξασφαλίσει ότι τα σήματα των χρωμάτων παραμένουν στο όριο των 0 έως 0,7V που ορίζει το πρωτόκολλο για την χρήση της VGA. Τα σήματα HS και VS είναι επιπέδου TTL και έρχονται απευθείας από το FPGA μέσω του connector B, όπως γίνεται αντιληπτό και από το σχήμα 3.21.

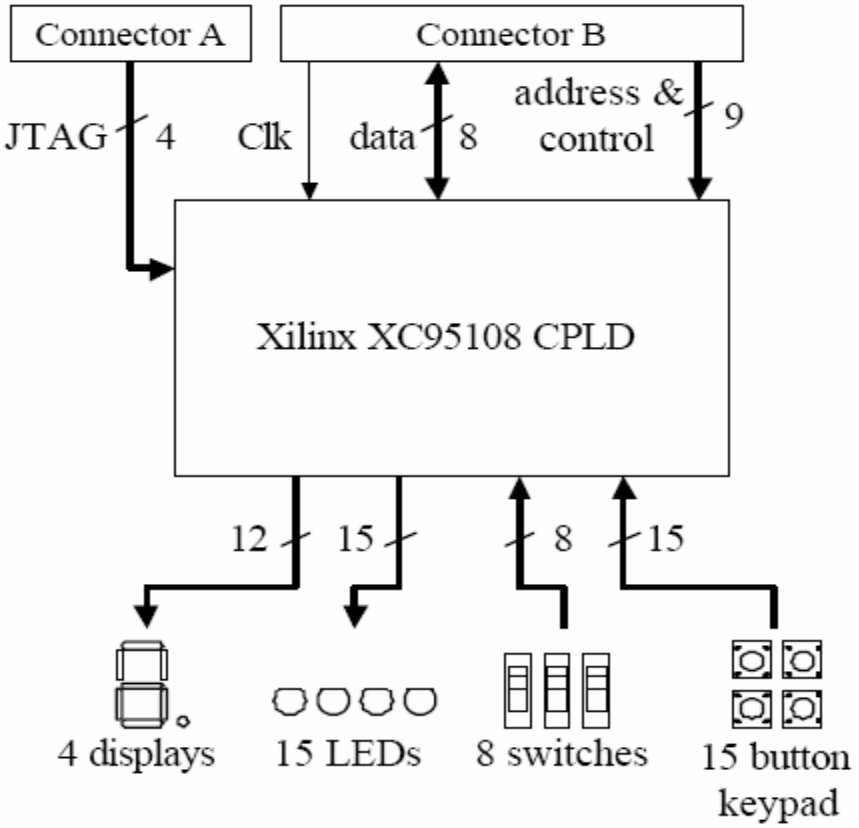
3.2.2.8 CPLD XC95108-PC84



Σχήμα 3.22 – Διάγραμμα του CPLD XC95108

Τα Complex Programmable Logic Devices (CPLD) της οικογένειας XC9500 της Xilinx, αποτελούνται από Input/Output Blocks (IOBs) και Function Blocks ή αλλιώς Simple Programmable Logic Devices (SPLD), όπως φαίνεται στο σχήμα 3.22. Κάθε Function Block χαρακτηρίζεται από τον αριθμό των εισόδων και των εξόδων του. Το XC95108 αποτελείται από έξι Function Blocks τύπου 36V18, δηλαδή με 36 εισόδους και 18 εξόδους το καθένα. Τα

Function Blocks αποτελούνται από macrocells, ένα για κάθε έξοδο, ενώ κάθε macrocell περιέχει ένα συνδυαστικό κομμάτι, ένα D Flip-Flop και έναν τρικατάστατο απομονωτή. Η έξοδος του macrocell μπορεί να οδηγηθεί είτε από το συνδυαστικό τμήμα είτε από το D Flip-Flop, ανάλογα με το πώς θα προγραμματιστεί ένας πολυπλέκτης που υπάρχει ειδικά για αυτόν τον σκοπό. Για το XC95108 της κάρτας DIO2, η αναμενόμενη καθυστέρηση από ακροδέκτη εισόδου σε ακροδέκτη εξόδου είναι 15 ns.



Σχήμα 3.23 – Οι συνδέσεις του CPLD με τα υπόλοιπα μέρη της κάρτας DIO2

Όπως γίνεται αντιληπτό από το σχήμα 3.23 το CPLD δεν συνδέεται με τα σήματα της θύρας VGA, της θύρας PS/2 και της οθόνης LCD, τα οποία δρομολογούνται απευθείας από τον connector B στις συσκευές. Όλες οι υπόλοιπες συσκευές συνδέονται και ελέγχονται μέσω του CPLD. Το CPLD μπορεί να διαμορφωθεί μόνο με την μέθοδο JTAG. Η διαμόρφωση μπορεί να γίνει με ειδικό καλώδιο μέσω των ακροδεκτών 1, 3, 5 και 7 του connector A. Αντί για καλώδιο, μπορεί να χρησιμοποιηθεί μια κάρτα, όπως η Digilab 2, η οποία θα συνδεθεί σε αυτούς τους ακροδέκτες και θα υλοποιήσει μια JTAG θύρα.

Το CPLD διατηρεί την διαμόρφωσή του ακόμα και όταν διακοπεί η τροφοδοσία, ενώ για την πραγματοποίηση των εφαρμογών, είναι ήδη διαμορφωμένο ώστε να υλοποιεί ένα απλό κύκλωμα που πολυπλέκει τα σήματα εισόδου και εξόδου του CPLD σε ένα διάδρομο δεδομένων των οκτώ bit. Παράλληλα χρησιμοποιώντας των διάδρομο διευθύνσεων των έξι bit δίνεται η δυνατότητα εγγραφής σημάτων εξόδου στους καταχωρητές του CPLD και ανάγνωσης των σημάτων εισόδου σαν να ήταν θέσεις μνήμης RAM. Για την ενεργοποίηση των παραπάνω λειτουργιών θα πρέπει ένα σήμα ελέγχου, το cs, να οδηγηθεί στο λογικό ‘1’. Δύο άλλα σήματα ελέγχου, το oe και το we, χρησιμοποιούνται για την ανάγνωση και την εγγραφή. Για να γίνει ανάγνωση δεδομένων πρέπει το we να συγκρατείται στο λογικό ‘0’ και το oe στο λογικό ‘1’, ενώ για την διαδικασία της εγγραφής, θα πρέπει το oe να βρίσκεται στο λογικό ‘0’ και να εμφανιστεί αρνητική ακμή στο we. Οι χρονικοί περιορισμοί για την ορθή λειτουργία του CPLD φαίνονται στον πίνακα 3.8.

Πίνακας 3.8 – Οι χρονικοί περιορισμοί των σημάτων ελέγχου του CPLD

Ελάχιστη περίοδος του σήματος WE (ανάμεσα σε δυο ακμές εγγραφής)	30 ns
Ελάχιστος χρόνος προετοιμασίας δεδομένων πριν την ακμή εγγραφής	7 ns
Ελάχιστος χρόνος συγκράτησης δεδομένων μετά την ακμή εγγραφής	0 ns
Ελάχιστος χρόνος προετοιμασίας (άνοδος) του σήματος OE πριν την ανάγνωση	10 ns
Ελάχιστος χρόνος συγκράτησης (στο 1) του σήματος OE μετά την ανάγνωση	10 ns
Ελάχιστος χρόνος προετοιμασίας (κάθοδος) του σήματος OE πριν την ακμή εγγραφής	10 ns
Ελάχιστος χρόνος συγκράτησης (στο 0) του σήματος OE μετά την ακμή εγγραφής	10 ns
Ελάχιστος χρόνος προετοιμασίας (κάθοδος) του σήματος WE πριν την ανάγνωση	10 ns
Ελάχιστος χρόνος συγκράτησης (στο 0) του σήματος WE μετά την ανάγνωση	2 ns
Ελάχιστος χρόνος προετοιμασίας (άνοδος) του σήματος CS πριν την ανάγνωση	15 ns
Ελάχιστος χρόνος συγκράτησης (στο 1) του σήματος CS μετά την ανάγνωση	2 ns

Όπως έχει ήδη αναφερθεί γίνεται χρήση του χώρου διευθύνσεων 00H-07H, προκειμένου να ελεγχθούν τα σήματα των συσκευών που συνδέονται με το CPLD. Οι διευθύνσεις των έξι bit που αντιστοιχούν σε κάθε συσκευή αναγράφονται στον πίνακα 3.9.

Πίνακας 3.9 – Ο χάρτης διευθύνσεων για τις συσκευές εισόδου / εξόδου

Διεύθυνση	bit7	bit6	bit5	bit4	bit3	bit2	bit1	Bit0	Περιγραφή
0	BTN7	BTN6	BTN5	BTN4	BTN3	BTN2	BTN1	BTN0	Πιεστικοί διακόπτες 7 - 0
1	0	BTNE	BTND	BTNC	BTNB	BTNA	BTN9	BTN8	Πιεστικοί διακόπτες 14 – 8
2	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1	Διακόπτες
3	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1	Διακόπτες (αντίγραφο)
4	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0	LED 7 - 0
5	LDF	LDE	LDD	LDC	LDB	LDA	LD9	LD8	LED 15 - 8
6	D2B3	D2B2	D2B1	D2B0	D1B3	D1B2	D1B1	D1B0	Οθόνη 7 τμημάτων, ψηφία 2 και 1
7	D4B3	D4B2	D4B1	D4B0	D3B3	D3B2	D3B1	D3B0	Οθόνη 7 τμημάτων, ψηφία 4 και 3

Τέλος, εκτός από την κανονική λειτουργία που αναφέρθηκε έως τώρα, υπάρχει και η δοκιμαστική λειτουργία του CPLD, όπου ο έλεγχος των συσκευών εισόδου / εξόδου περνάει στον χρήστη. Η λειτουργία αυτή τίθεται σε εφαρμογή όταν ενώ οι διακόπτες SW1 και SW8 βρίσκονται στο λογικό ‘1’ και ‘0’ αντίστοιχα, κρατηθούν πατημένοι οι πιεστικοί διακόπτες B7 και BE και κατόπιν πατηθεί και ο διακόπτης B0. Στην δοκιμαστική λειτουργία όλοι οι άνοδοι των ψηφίων της οθόνης επτά τμημάτων οδηγούνται στο λογικό 1, δηλαδή εμφανίζονται όλα τα ψηφία, ενώ οι κάθοδοι των τμημάτων ελέγχονται από τους διακόπτες με τον SW8 να οδηγεί την υποδιαστολή (DP). Επίσης στην ίδια λειτουργία κάθε LED ελέγχεται από τον πιεστικό διακόπτη με το αντίστοιχο νούμερο, εκτός από το LDF που οδηγείται παράλληλα με το LDE, μιας και δεν υπάρχει BF, και το LD0, που οδηγείται από τον B1 παράλληλα με το LED1. Η έξοδος από την δοκιμαστική λειτουργία γίνεται με το πάτημα του πιεστικού διακόπτη B0.

Πληροφορίες για τους ακροδέκτες του CPLD βρίσκονται στον πίνακα 10 που βρίσκεται στο τέλος της περιγραφής της κάρτας DIO2.

3.2.2.9 Τροφοδοσία

Η κάρτα DIO2 τροφοδοτείται από έναν ρυθμιστή τάσης LM2940 που παράγει 5 V, ο οποίος με την σειρά του τροφοδοτείται από τους ακροδέκτες 39 και 40 του connector A. Στους ακροδέκτες αυτούς συνδέονται οι ακροδέκτες 1 και 2 αντίστοιχα του connector C του Digilab 2, δηλαδή η απευθείας τροφοδοσία και η γείωση. Λόγω του ότι η DIO2 μπορεί να φτάσει σε κατανάλωση τα 350 mA, με όλα τα LED αναμμένα, η απευθείας τροφοδοσία, από το Digilab 2, θα πρέπει να είναι 5.5-8 V

3.2.2.10 Connectors

Οι connectors A και B της κάρτας DIO2 συνδέονται με τους connector D και C του Digilab 2 αντίστοιχα. Ο connector A λαμβάνει τα σήματα για τον προγραμματισμό JTAG και την τροφοδοσία του CPLD, ενώ τα υπόλοιπα σήματα, ρολόι, σήματα ελέγχου και δεδομένα συσκευών, συνδέονται μέσω του connector B. Η αντιστοίχιση των σημάτων με τους ακροδέκτες τόσο για τους connectors της DIO2, όσο και για το CPLD, την κάρτα Digilab 2 και το FPGA αυτής, φαίνεται συγκεντρωτικά στον πίνακα 3.10.

Πίνακας 3.10 – Η αντιστοίχηση των σημάτων με τους ακροδέκτες των καρτών και των στοιχείων τους

	Ακροδέκτης FPGA	Ακροδέκτης στο Digilab 2	Ακροδέκτης στο DIO2	Ακροδέκτης CPLD	Όνομα σήματος στην κάρτα	Επεξήγηση
1	80	-	-	-	CLK1	Κύριο ρολόι του FPGA (GCLK0)
2	77	-	-	-	BTN1	Πιεστικός διακόπτης του Digilab 2 (GCLK1)
3	71	-	-	-	LED1	LED του Digilab 2
4	179	C06	B36	-	-	6ο bit διεύθυνσης, ασύνδετο
5	180	C05	B35	75	A5	Διάδρομος διευθύνσεων, bit 5
6	176	C08	B34	77	A4	Διάδρομος διευθύνσεων, bit 4
7	178	C07	B33	79	A3	Διάδρομος διευθύνσεων, bit 3
8	174	C10	B32	80	A2	Διάδρομος διευθύνσεων, bit 2
9	175	C09	B31	81	A1	Διάδρομος διευθύνσεων, bit 1
10	172	C12	B30	83	A0	Διάδρομος διευθύνσεων, bit 0
11	173	C11	B29	10	BOCI / CLK	Κύριο ρολόι του CPLD (GCK2)
12						
13	168	C13	B27	76	OE	Επίτρεψη εξόδου του CPLD
14	165	C16	B26	9	WE	Ρολόι εγγραφής στο CPLD (GCK1)
15	166	C15	B25	13	CS	Σήμα επιλογής του CPLD
16	163	C18	B24	1	D7	Διάδρομος δεδομένων, bit 7
17	164	C17	B23	2	D6	Διάδρομος δεδομένων, bit 6
18	161	C20	B22	3	D5	Διάδρομος δεδομένων, bit 5
19	162	C19	B21	4	D4	Διάδρομος δεδομένων, bit 4
20	154	C22	B20	5	D3	Διάδρομος δεδομένων, bit 3
21	160	C21	B19	6	D2	Διάδρομος δεδομένων, bit 2
22	151	C24	B18	7	D1	Διάδρομος δεδομένων, bit 1
23	152	C23	B17	11	D0	Διάδρομος δεδομένων, bit 0
24						
25	150	C25	B15	-	LCD_E	Επίτρεψη εξόδου / Ρολόι εγγραφής
26	147	C28	B14	-	LCD_RS	Επιλογή καταχωρητή
27	148	C27	B13	-	LCD_RW	Επιλογή ανάγνωσης / εγγραφής
28	146	C29	B11	-	KCLK	Ρολόι PS2
29	142	C30	B12	-	KDAT	Δεδομένα PS2
30	140	C32	B10	-	HS	Οριζόντιος συγχρονισμός
31	141	C31	B09	-	VS	Κατακόρυφος συγχρονισμός
32	138	C34	B08	-	R0	Κόκκινο χρώμα, bit 1 (MSB)
33	139	C33	B07	-	R1	Κόκκινο χρώμα, bit 0 (LSB)
34	134	C37	B03	-	G0	Πράσινο χρώμα, bit 2 (MSB)
35	135	C36	B06	-	G1	Πράσινο χρώμα, bit 1
36	136	C35	B05	-	G2	Πράσινο χρώμα, bit 0 (LSB)
37	129	C40	B02	-	B0	Μπλε χρώμα, bit 2 (MSB)
38	132	C39	B01	-	B1	Μπλε χρώμα, bit 1
39	133	C38	B04	-	B2	Μπλε χρώμα, bit 0 (LSB)
40						
41	201	-	-	-	TXD	Σειριακά δεδομένα εκπομπής
42	202	-	-	-	RXD	Σειριακά δεδομένα λήψης
43	200	-	-	-	DSR	Σήμα ύπαρξης συσκευής
44	199	-	-	-	CTS	Αποδοχή αίτησης εκπομπής
45	195	-	-	-	RTS	Αίτηση για εκπομπή δεδομένων
46	15	-	-	-	PD0	Δεδομένα παράλληλης θύρας, bit 0
47	14	-	-	-	PD1	Δεδομένα παράλληλης θύρας, bit 1
48	10	-	-	-	PD2	Δεδομένα παράλληλης θύρας, bit 2

49	9	-	-	-	PD3	Δεδομένα παράλληλης θύρας, bit 3
50	8	-	-	-	PD4	Δεδομένα παράλληλης θύρας, bit 4
51	7	-	-	-	PD5	Δεδομένα παράλληλης θύρας, bit 5
52	6	-	-	-	PD6	Δεδομένα παράλληλης θύρας, bit 6
53	5	-	-	-	PD7	Δεδομένα παράλληλης θύρας, bit 7
54	4	-	-	-	PINT	Σήμα διακοπής
55	3	-	-	-	PWT	Σήμα αναμονής
56	206	-	-	-	PWE	Επίτρεψη εγγραφής
57	205	-	-	-	PDS	Σήμα μανδάλωσης δεδομένων
58	204	-	-	-	PAS	Σήμα μανδάλωσης διευθύνσεων
59	203	-	-	-	PRS	Σήμα μηδενισμού
60						
61	-	-	-	82	LD0	Κάθοδος LED LD0
62	-	-	-	12	LD1	Κάθοδος LED LD1 (GCK3)
63	-	-	-	14	LD2	Κάθοδος LED LD2
64	-	-	-	15	LD3	Κάθοδος LED LD3
65	-	-	-	17	LD4	Κάθοδος LED LD4
66	-	-	-	18	LD5	Κάθοδος LED LD5
67	-	-	-	19	LD6	Κάθοδος LED LD6
68	-	-	-	20	LD7	Κάθοδος LED LD7
69	-	-	-	63	LD8	Κάθοδος LED LD8
70	-	-	-	69	LD9	Κάθοδος LED LD9
71	-	-	-	67	LDA	Κάθοδος LED LDA
72	-	-	-	68	LDB	Κάθοδος LED LDB
73	-	-	-	70	LDC	Κάθοδος LED LDC
74	-	-	-	71	LDD	Κάθοδος LED LDD
75	-	-	-	72	LDE	Κάθοδος LED LDE
76	-	-	-	74	LDF	Κάθοδος LED LDF
77						
78	-	-	-	84	BTN0	Πιεστικός διακόπτης BTN0
79	-	-	-	47	BTN1	Πιεστικός διακόπτης BTN1
80	-	-	-	66	BTN2	Πιεστικός διακόπτης BTN2
81	-	-	-	52	BTN3	Πιεστικός διακόπτης BTN3
82	-	-	-	55	BTN4	Πιεστικός διακόπτης BTN4
83	-	-	-	48	BTN5	Πιεστικός διακόπτης BTN5
84	-	-	-	50	BTN6	Πιεστικός διακόπτης BTN6
85	-	-	-	57	BTN7	Πιεστικός διακόπτης BTN7
86	-	-	-	53	BTN8	Πιεστικός διακόπτης BTN8
87	-	-	-	58	BTN9	Πιεστικός διακόπτης BTN9
88	-	-	-	61	BTNA	Πιεστικός διακόπτης BTNA
89	-	-	-	56	BTNB	Πιεστικός διακόπτης BTNB
90	-	-	-	65	BTNC	Πιεστικός διακόπτης BTNC
91	-	-	-	62	BTND	Πιεστικός διακόπτης BTND
92	-	-	-	54	BTNE	Πιεστικός διακόπτης BTNE
93						
94	-	-	-	34	SW1	Διακόπτης SW1
95	-	-	-	40	SW2	Διακόπτης SW2
96	-	-	-	39	SW3	Διακόπτης SW3
97	-	-	-	41	SW4	Διακόπτης SW4
98	-	-	-	43	SW5	Διακόπτης SW5
99	-	-	-	45	SW6	Διακόπτης SW6
100	-	-	-	44	SW7	Διακόπτης SW7
101	-	-	-	46	SW8	Διακόπτης SW8
102						
103	-	-	-	21	AN1	Άνοδος ψηφίου 1 του 7seg

104	-	-	-	23	AN2	Άνοδος ψηφίου 2 του 7seg
105	-	-	-	24	AN3	Άνοδος ψηφίου 3 του 7seg
106	-	-	-	25	AN4	Άνοδος ψηφίου 4 του 7seg
107	-	-	-	35	CA	Κάθοδος τμήματος A του 7seg
108	-	-	-	33	CB	Κάθοδος τμήματος B του 7seg
109	-	-	-	26	CC	Κάθοδος τμήματος C του 7seg
110	-	-	-	51	CD	Κάθοδος τμήματος D του 7seg
111	-	-	-	36	CE	Κάθοδος τμήματος E του 7seg
112	-	-	-	32	CF	Κάθοδος τμήματος F του 7seg
113	-	-	-	31	CG	Κάθοδος τμήματος G του 7seg
114	-	-	-	37	DP	Κάθοδος υποδιαστολής του 7seg
115						
116	52	-	-	-	M0	Μέθοδος προγραμματισμού FPGA, bit 0
117	50	-	-	-	M1	Μέθοδος προγραμματισμού FPGA, bit 1
118	54	-	-	-	M2	Μέθοδος προγραμματισμού FPGA, bit 2
119						
120	207	-	-	-	TCLK	Ρολόι JTAG του FPGA
121	2	-	-	-	TMS	Επιλογή λειτουργίας JTAG του FPGA
122	159	-	-	-	TDI	Δεδομένα εισόδου JTAG του FPGA
123	157	-	-	-	TDO	Δεδομένα εξόδου JTAG του FPGA
124						
125	106	-	-	-	PROG	Σήμα αρχής προγραμματισμού
126	107	-	-	-	INIT	Ένδειξη διαγραφής προγραμματισμού
127	155	-	-	-	CCLK	Ρολόι προγραμματισμού από SROM
128	153	-	-	-	DIN	Δεδομένα εισόδου από SROM
129	104	-	-	-	DONE	Ένδειξη τέλους προγραμματισμού
130						
131	73	D39	A01	30	TCLK	Ρολόι JTAG του CPLD
132	84	D33	A07	29	TMS	Επιλογή λειτουργίας JTAG του CPLD
133	82	D35	A05	28	TDI	Δεδομένα εισόδου JTAG του CPLD
134	75	D37	A03	59	TDO	Δεδομένα εξόδου JTAG του CPLD
135						
136	70	A04	-	-	-	Ακροδέκτης 4 του connector A
137	69	A05	-	-	-	Ακροδέκτης 5 του connector A
138	68	A06	-	-	-	Ακροδέκτης 6 του connector A
139	67	A07	-	-	-	Ακροδέκτης 7 του connector A
140	63	A08	-	-	-	Ακροδέκτης 8 του connector A
151	62	A09	-	-	-	Ακροδέκτης 9 του connector A
152	61	A10	-	-	-	Ακροδέκτης 10 του connector A
153	60	A11	-	-	-	Ακροδέκτης 11 του connector A
154	59	A12	-	-	-	Ακροδέκτης 12 του connector A
155	58	A13	-	-	-	Ακροδέκτης 13 του connector A
156	57	A14	-	-	-	Ακροδέκτης 14 του connector A
157	49	A15	-	-	-	Ακροδέκτης 15 του connector A
158	48	A16	-	-	-	Ακροδέκτης 16 του connector A
159	47	A17	-	-	-	Ακροδέκτης 17 του connector A
160	46	A18	-	-	-	Ακροδέκτης 18 του connector A
161	45	A19	-	-	-	Ακροδέκτης 19 του connector A
162	44	A20	-	-	-	Ακροδέκτης 20 του connector A
163	43	A21	-	-	-	Ακροδέκτης 21 του connector A
164	42	A22	-	-	-	Ακροδέκτης 22 του connector A
165	41	A23	-	-	-	Ακροδέκτης 23 του connector A
166	37	A24	-	-	-	Ακροδέκτης 24 του connector A
167	36	A25	-	-	-	Ακροδέκτης 25 του connector A
168	35	A26	-	-	-	Ακροδέκτης 26 του connector A

169	34	A27	-	-	-	Ακροδέκτης 27 του connector A
170	33	A28	-	-	-	Ακροδέκτης 28 του connector A
171	31	A29	-	-	-	Ακροδέκτης 29 του connector A
172	30	A30	-	-	-	Ακροδέκτης 30 του connector A
173	29	A31	-	-	-	Ακροδέκτης 31 του connector A
174	27	A32	-	-	-	Ακροδέκτης 32 του connector A
175	24	A33	-	-	-	Ακροδέκτης 33 του connector A
176	23	A34	-	-	-	Ακροδέκτης 34 του connector A
177	22	A35	-	-	-	Ακροδέκτης 35 του connector A
178	21	A36	-	-	-	Ακροδέκτης 36 του connector A
179	20	A37	-	-	-	Ακροδέκτης 37 του connector A
180	18	A38	-	-	-	Ακροδέκτης 38 του connector A
181	17	A39	-	-	-	Ακροδέκτης 39 του connector A
182	16	A40	-	-	-	Ακροδέκτης 40 του connector A
183						
184	194	B04	-	-	-	Ακροδέκτης 4 του connector B
185	193	B05	-	-	-	Ακροδέκτης 5 του connector B
186	192	B06	-	-	-	Ακροδέκτης 6 του connector B
187	191	B07	-	-	-	Ακροδέκτης 7 του connector B
188	189	B08	-	-	-	Ακροδέκτης 8 του connector B
189	188	B09	-	-	-	Ακροδέκτης 9 του connector B
190	187	B10	-	-	-	Ακροδέκτης 10 του connector B
191	185	B11	-	-	-	Ακροδέκτης 11 του connector B (GCK3)
192	182	B12	-	-	-	Ακροδέκτης 12 του connector B (GCK2)
193						
194	181	C04	B38	-	-	Ακροδέκτης 4 του connector C
195	167	C14	B28	-	-	Ακροδέκτης 14 του connector C
196	149	C26	B16	-	-	Ακροδέκτης 26 του connector C
197				-	-	
198	127	D04	A38	-	-	Ακροδέκτης 4 του connector D
199	125	D05	A35	-	-	Ακροδέκτης 5 του connector D
200	126	D06	A36	-	-	Ακροδέκτης 6 του connector D
201	122	D07	A33	-	-	Ακροδέκτης 7 του connector D
202	123	D08	A34	-	-	Ακροδέκτης 8 του connector D
203	120	D09	A31	-	-	Ακροδέκτης 9 του connector D
204	121	D10	A32	-	-	Ακροδέκτης 10 του connector D
205	115	D11	A29	-	-	Ακροδέκτης 11 του connector D
206	119	D12	A30	-	-	Ακροδέκτης 12 του connector D
207	113	D13	A27	-	-	Ακροδέκτης 13 του connector D
208	114	D14	A28	-	-	Ακροδέκτης 14 του connector D
209	111	D15	A25	-	-	Ακροδέκτης 15 του connector D
210	112	D16	A26	-	-	Ακροδέκτης 16 του connector D
211	109	D17	A23	-	-	Ακροδέκτης 17 του connector D
212	110	D18	A24	-	-	Ακροδέκτης 18 του connector D
213	102	D19	A21	-	-	Ακροδέκτης 19 του connector D
214	108	D20	A22	-	-	Ακροδέκτης 20 του connector D
215	100	D21	A19	-	-	Ακροδέκτης 21 του connector D
216	101	D22	A29	-	-	Ακροδέκτης 22 του connector D
217	98	D23	A17	-	-	Ακροδέκτης 23 του connector D
218	99	D24	A18	-	-	Ακροδέκτης 24 του connector D
219	96	D25	A15	-	-	Ακροδέκτης 25 του connector D
220	97	D26	A16	-	-	Ακροδέκτης 26 του connector D
221	94	D27	A13	-	-	Ακροδέκτης 27 του connector D
222	95	D28	A14	-	-	Ακροδέκτης 28 του connector D
223	89	D29	A11	-	-	Ακροδέκτης 29 του connector D

224	90	D30	A12	-	-	Ακροδέκτης 30 του connector D
225	87	D31	A09	-	-	Ακροδέκτης 31 του connector D
226	88	D32	A10	-	-	Ακροδέκτης 32 του connector D
227	86	D34	A08	-	-	Ακροδέκτης 34 του connector D
228	83	D36	A06	-	-	Ακροδέκτης 36 του connector D
229	81	D38	A04	-	-	Ακροδέκτης 38 του connector D
230	74	D40	A02	-	-	Ακροδέκτης 40 του connector D

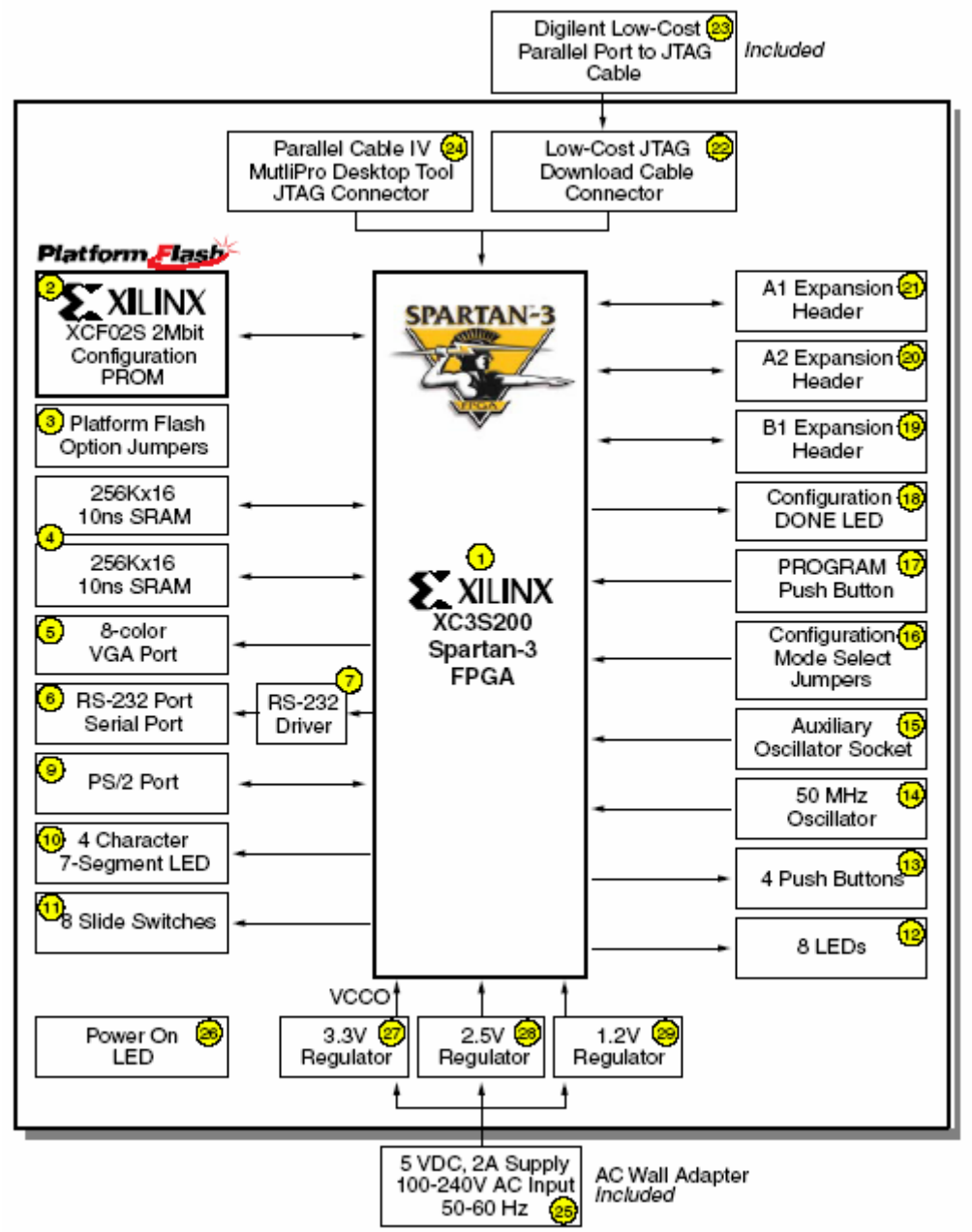
3.3 Η κάρτα Spartan-3 Starter Kit

3.3.1 Κύρια μέρη και χαρακτηριστικά

Στο σχήμα 3.24 φαίνεται ένα διάγραμμα της κάρτας Spartan-3 Starter Kit, της Xilinx, η οποία περιλαμβάνει τα παρακάτω μέρη (σε παρένθεση οι θέσεις τους στα σχήματα 3.24, 3.25, και 3.26):

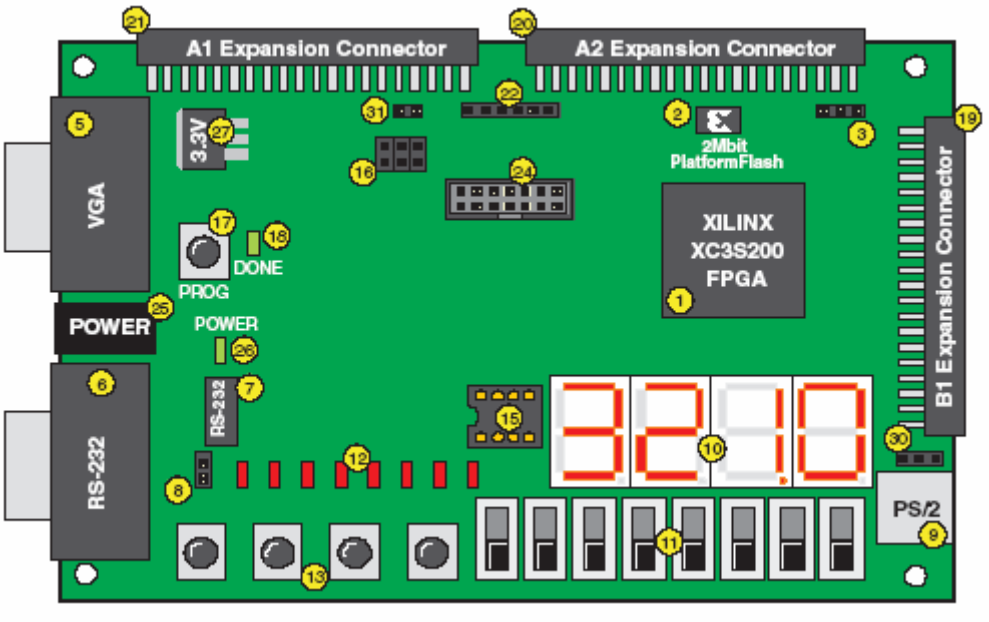
- Το FPGA Spartan-3 της Xilinx, που περιέχεται στο πακέτο XC3S200-FT256 (1)
  - 4.320 logic cells
  - Δώδεκα μπλοκ RAM των 18K-bit
  - Δώδεκα hardware πολλαπλασιαστές 18x18
  - Τέσσερις Digital Clock Managers (DCM), για την παραγωγή ρολογιού συχνότητας διαφορετικής την ονομαστική της κάρτας
  - Μέχρι 173 σήματα εισόδου εξόδου, καθορισμένα από τον χρήστη
- Μία μνήμη PROM, στο ολοκληρωμένο Xilinx XCF02S Platform Flash, μεγέθους 2Mbit (2)
  - Δυνατότητα αποθήκευσης 1Mbit αμετάβλητων δεδομένων ή κώδικα εφαρμογής, μετά την σύνθεση στο FPGA
  - Ακροδέκτες ελέγχου που επιτρέπουν την ανάγνωση δεδομένων PROM ή της διαμόρφωσης του FPGA από άλλες πηγές (3)
- 1M-byte γρήγορης ασύγχρονης SRAM (βρίσκεται στην κάτω πλευρά της κάρτας) (4)
  - Δύο ολοκληρωμένα SRAM 256Kx16
  - Μεταβλητή αρχιτεκτονική μνήμης
    - Μία ενιαία SRAM 256Kx32, ιδανική για τον MicroBlaze
    - Δύο ανεξάρτητες SRAM 256Kx16
  - Ξεχωριστό σήμα επιλογής για κάθε συσκευή μνήμης
  - Ξεχωριστό σήμα επίτρεψης / εγκυρότητας για κάθε byte
- Μία θύρα εξόδου VGA 3-bit και 8 χρωμάτων (5)
- Σειριακή θύρα RS-232, 9 ακροδεκτών (6)
  - Σύνδεση μέσω θηλυκού connector (DB9)
  - Ξεχωριστό οδηγό για την σειριακή θύρα (7)
  - Εφεδρικούς δοκιμαστικούς δίαυλους λήψης και εκπομπής RS-232 (8)
- Μία θύρα τύπου PS/2 για ποντίκι / πληκτρολόγιο (9)
- Μία πλήρως ελεγχόμενη οθόνη τεσσάρων χαρακτήρων των επτά τμημάτων (10)
- Οκτώ διακόπτες (11)
- Οκτώ ξεχωριστές εξόδους σε LED (12)
- Τέσσερις πιεστικούς διακόπτες (13)
- Ταλαντωτή 50 MHz (στην κάτω όψη της κάρτας) (14)
- Υποδοχή για βοηθητικό ταλαντωτή (15)
- Επιλογή τύπου διαμόρφωσης του FPGA μέσω ακροδεκτών ελέγχου (16)
- Πιεστικό διακόπτη για την επαναφορά του FPGA στην ονομαστική του λειτουργία (17)

- LED που υποδεικνύει ότι το FPGA έχει προγραμματιστεί επιτυχώς (18)
- Τρεις θύρες επέκτασης των 40 ακροδεκτών που επιτρέπει την σύνδεση με συμβατές κάρτες (19,20,21)
- Μία θύρα JTAG (22) συμβατή με το καλώδιο προγραμματισμού / εκσφαλμάτωσης που συνδέεται στην σειριακή θύρα ενός προσωπικού υπολογιστή (23)
- Μία θύρα JTAG προγραμματισμού / εκσφαλμάτωσης συμβατή με τα Xilinx Parallel Cable IV και MultiPRO Desktop Tool (24)
- Υποδοχή τροφοδοσίας 5V (25)
- LED ένδειξης λειτουργίας (26)
- Ρυθμιστές τάσης για 3.3V (27), 2.5V (28) και 1.2V (29)

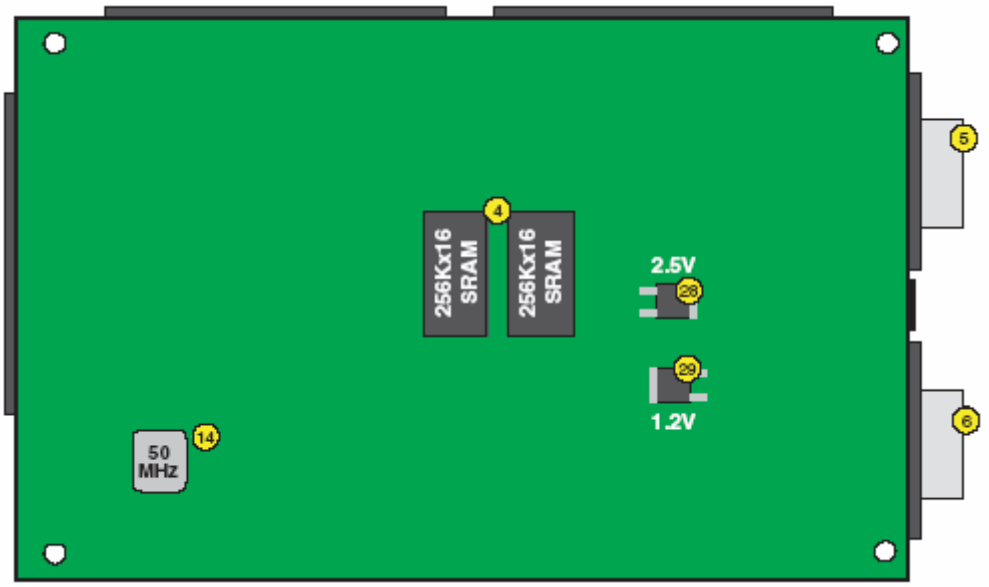


Σχήμα 3.24 – Διάγραμμα της κάρτας Xilinx Spartan-3 Starter Kit

Στα σχήματα 3.25 και 3.26, που ακολουθούν, φαίνονται οι θέσεις των παραπάνω στοιχείων της κάρτας, στην πάνω και κάτω πλευρά αυτής αντίστοιχα.



Σχήμα 3.25 – Η πάνω όψη της κάρτας Xilinx Spartan-3 Starter Kit



Σχήμα 3.26 - Η κάτω όψη της κάρτας Xilinx Spartan-3 Starter Kit

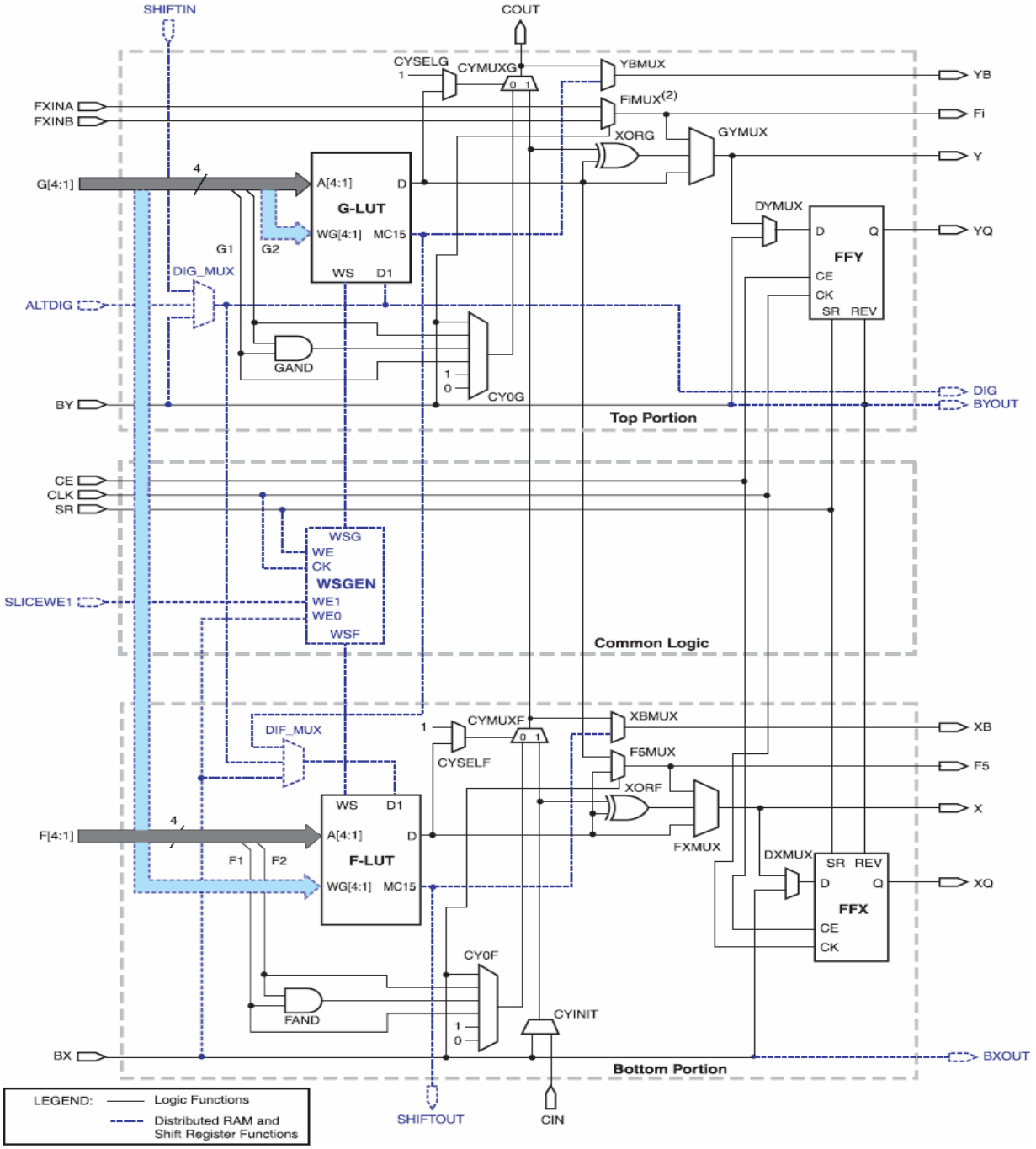
3.3.2 Αναλυτική περιγραφή

3.3.2.1 FPGA Spartan-3 XC3S200-FT256

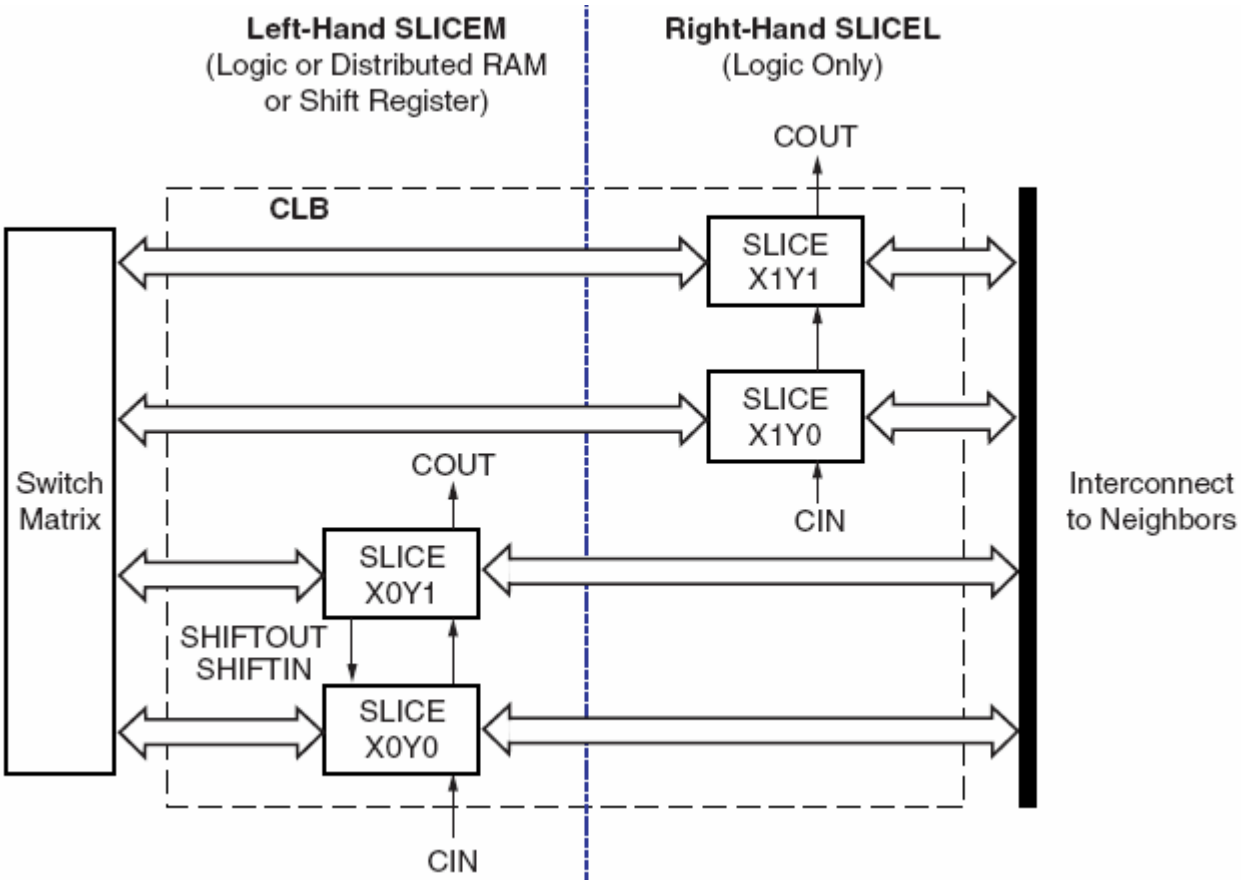
Τα FPGAs, Field Programmable Logic Arrays, της οικογένειας Spartan-3 βασίζονται σε πέντε θεμελιώδη μέρη:

- Στα CLBs, Configurable Logic Blocks, που αποτελούν τα προγραμματιζόμενα κυκλώματα που υλοποιούν τόσο αποθήκευση δεδομένων όσο και συνδυαστική λογική. Κάθε CLB αποτελείται από τέσσερα slices, τα οποία χωρίζονται σε δύο ζευγάρια. Το μεγαλύτερο μέρος του FPGA αποτελείται από έναν πίνακα από CLBs, που στην συσκευή XC3S200 είναι μεγέθους 24 γραμμών και 20 στηλών. Μέσα στο CLB τα slices ονομάζονται με τον εξής

τρόπο: το γράμμα X και ο αριθμός που ακολουθεί μετά δείχνει την στήλη στην οποία ανήκει το slice, μετρώντας από αριστερά προς τα δεξιά, ενώ το γράμμα Y και το νούμερο που ακολουθεί μετά δείχνει την γραμμή του slice, μετρώντας από κάτω προς τα πάνω, αλλά και την σχετική θέση αυτού στο ζευγάρι. Για να γίνει κατανοητή η διάταξη τα slices του σχήματος 3.28 ανήκουν στο κάτω αριστερά CLB. Σε κάθε CLB ο όρος 'left-hand' ή SLICEM χρησιμοποιείται για να δηλώσει το ζευγάρι των slices που έχει ζυγό αριθμό μετά το γράμμα X, και αντίστοιχα ο όρος 'right-hand' ή SLICEL δηλώνει το ζευγάρι των slices που έχει μονό αριθμό μετά το γράμμα Y. Όλα τα slices αποτελούνται από τα ίδια βασικά στοιχεία, πολυπλέκτες, καταχωρητές, αριθμητικές πύλες, που λειτουργούν ως RAM ή ROM. Παρόλα αυτά τα SLICEMs χρησιμοποιούνται για να υλοποιούν λογικά κυκλώματα ή καταχωρητές ολίσθησης και αποθήκευσης, με λειτουργία RAM ή αλλιώς Look-Up Table (LUT), ενώ τα SLICELs μόνο για να υλοποιούν λογικά κυκλώματα. Η συσκευή XC3S200 περιέχει 480 CLBs που αντιστοιχούν σε 200K πύλες.



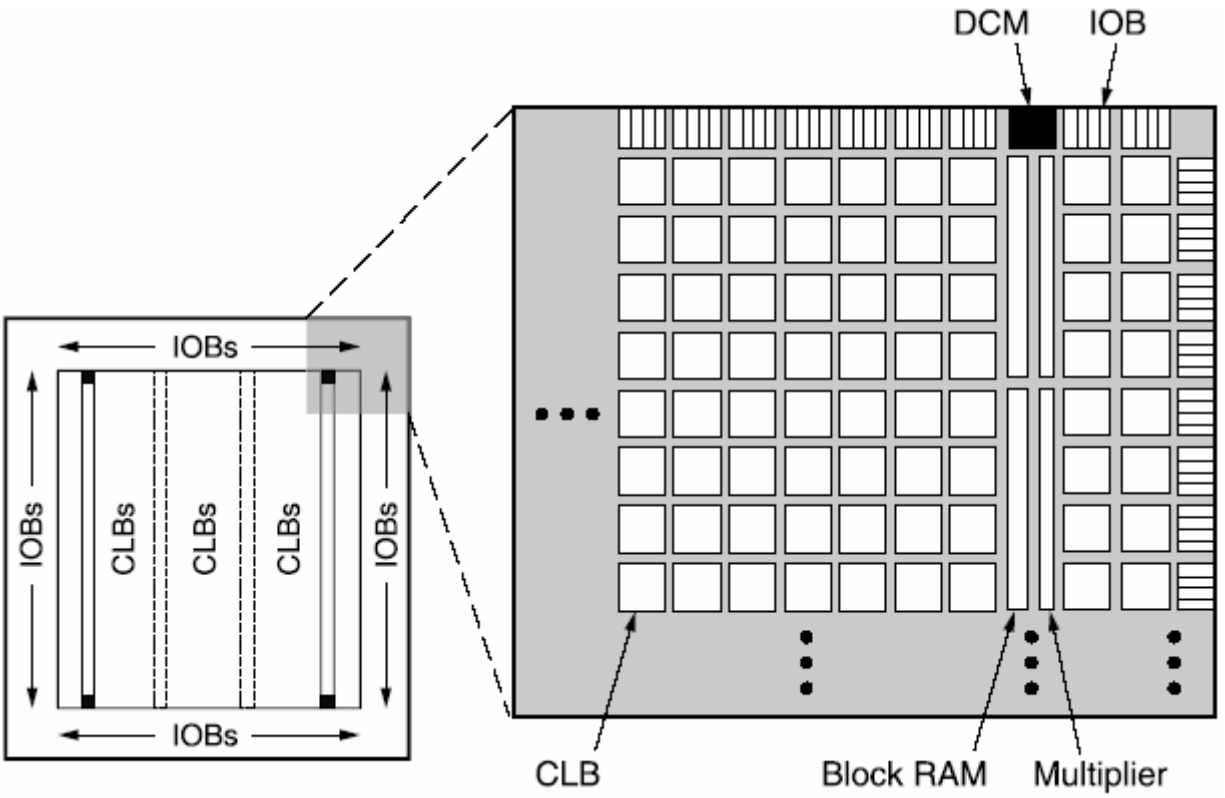
Σχήμα 3.27 – Απλοποιημένο διάγραμμα ενός SLICEM



Σχήμα 3.28 – Η διάταξη των slices μέσα στο CLB

- Στα blocks εισόδου / εξόδου, Input/Output Blocks (IOB), που ελέγχουν την ροή των δεδομένων μεταξύ των ακροδεκτών I/O και της εσωτερικής λογικής της συσκευής. Κάθε IOB υποστηρίζει ροή δεδομένων δύο κατευθύνσεων και τρικατάστατη λειτουργία. Τα IOBs περιβάλλουν τα CLBs όπως φαίνεται στο σχήμα 3.29, ενώ η συσκευή XC3S200 υποστηρίζει 173 σήματα I/O οριζόμενα από τον χρήστη.
- Στα block RAM για την αποθήκευση δεδομένων, μεγέθους 18 Kbit. Τα blocks είναι dual-port. Η συσκευή XC3S200 διαθέτει δώδεκα τέτοια block, επομένως 216 Kbit RAM. Τα blocks διατάσσονται σε στήλες και παρεμβάλλονται μεταξύ των CLBs, η συσκευή XC3S200 διαθέτει δύο τέτοιες στήλες.
- Στους hardware πολλαπλασιαστές των 18 bit. Οι είσοδοι των πολλαπλασιαστών αυτών δέχονται δεδομένα σε μορφή συμπληρώματος ως προς δύο, είτε προσημασμένους αριθμούς των 18 bit είτε μη προσημασμένους των 17 bit και συνδέονται εσωτερικά με ένα block RAM. Η λειτουργία του πολλαπλασιαστή μπορεί να είναι ασύγχρονη ή σύγχρονη, ενώ ο συνδυασμός του με άλλους πολλαπλασιαστές δίνει την δυνατότητα πράξεων με άνω των δύο συντελεστών και χειρισμό αριθμών μεγέθους μεγαλύτερου από 18bit.
- Στους DCMs, Digital Clock Manager, οι οποίοι εξαλείφουν τις ασυμμετρίες στις φάσεις σημάτων που διανύουν διαφορετικά μονοπάτια, παράγουν παλμούς ρολογιού πολλαπλάσιους ή υποπολλαπλάσιους αυτών που δέχονται ως είσοδο και έχουν την δυνατότητα να μετατοπίζουν την φάση των παλμών ρολογιού. Οι DCMs βρίσκονται στο τέλος των στηλών των block RAM.

Μια σύνοψη των ιδιοτήτων των FPGAs της οικογένειας Spartan-3 γίνεται στον πίνακα 3.11.



Σχήμα 3.29 – Η αρχιτεκτονική του FPGA Spartan-3

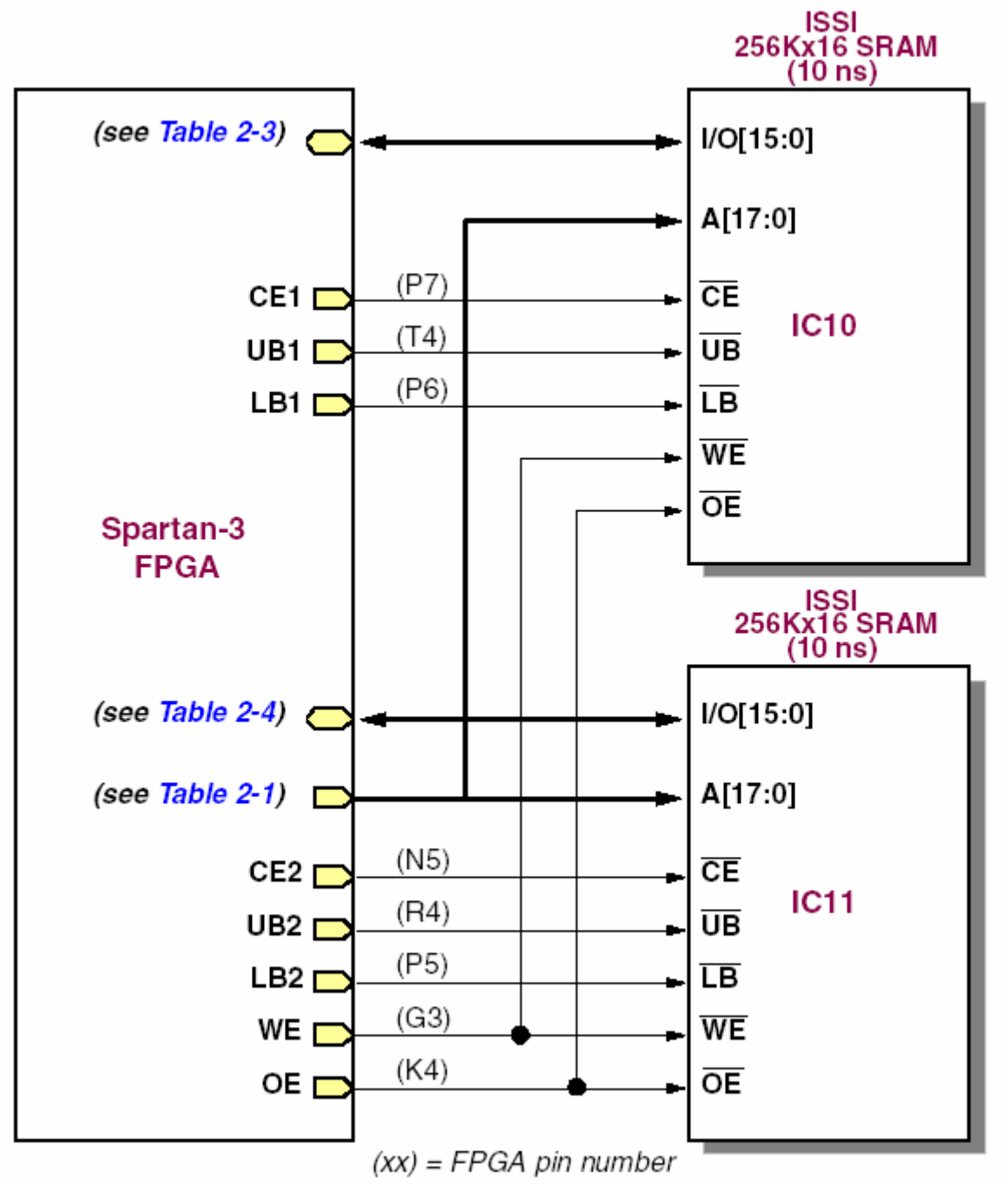
Πίνακας 3.11 – Οι κυριότερες ιδιότητες των FPGAs της οικογένειας Spartan-3

Device	System Gates	Logic Cells	CLB Array (One CLB = Four Slices)			Distributed RAM (bits)	Block RAM (bits)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	712	312
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	784	344

3.3.2.2 Γρήγορη, ασύγχρονη SRAM

Η κάρτα Spartan-3 Starter Kit διαθέτει ένα Mbyte γρήγορης, ασύγχρονης μνήμης SRAM, η οποία βρίσκεται στην κάτω όψη της κάρτας. Η μνήμη περιλαμβάνει δύο ολοκληρωμένα 256Kx16 ISSI (IS61LV25616AL-10T) καθυστέρησης 10ns. Τα δύο ολοκληρωμένα είναι δυνατόν είτε να σχηματίσουν μία μνήμη SRAM 256Kx32, είτε δύο ανεξάρτητες μνήμες 256Kx16. Επίσης μοιράζονται τα σήματα επίτρεψης εγγραφής (write-enable WE), επίτρεψης εξόδου (output-enable OE), και διευθύνσεων (A[17:0]). Αντιθέτως κάθε συσκευή μνήμης έχει το δικό της σήμα επιλογής (chip select enable CE) και ξεχωριστό σήμα επίτρεψης byte, το οποίο καθορίζει εάν το υψηλό byte (UB) και το χαμηλό byte (LB) της 16-bit λέξης δεδομένων της μνήμης είναι έγκυρα.

Η δομή μνήμης 256Kx16 είναι ιδεατή για τις εντολές του MicroBlaze. Εναλλακτικά προσφέρει αποθήκευση δεδομένων υψηλής πυκνότητας για μια σειρά από εφαρμογές, όπως ψηφιακή επεξεργασία σημάτων, μεγάλες ουρές δεδομένων (FIFO) και buffering γραφικών.



Σχήμα 3.30 – Σύνδεση των ολοκληρωμένων SRAM με το FPGA

Όπως αναφέρθηκε τα δύο ολοκληρωμένα μνήμης SRAM μοιράζονται τις ίδιες 18-bit γραμμές διευθύνσεων, όπως φαίνεται στον πίνακα 3.12. Τα σήματα διευθύνσεων συνδέονται επίσης με την θύρα επέκτασης A1.

Πίνακας 3.12 – Συνδέσεις ακροδεκτών των σημάτων διευθύνσεων των ολοκληρωμένων μνήμης SRAM

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

Εκτός από τα σήματα διευθύνσεων, τα ολοκληρωμένα μνήμης μοιράζονται και τα σήματα επίτρεψης εγγραφής (write-enable WE) και επίτρεψης εξόδου (output-enable OE), όπως φαίνεται στον πίνακα 3.13. Και τα σήματα αυτά συνδέονται με την θύρα επέκτασης A1.

Πίνακας 3.13 – Συνδέσεις ακροδεκτών για τα σήματα OE και WE των ολοκληρωμένων μνήμης SRAM

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

Τα σήματα δεδομένων επιλογής συσκευής και επίτρεψης / εγκυρότητας byte δεδομένων συνδέονται αποκλειστικά μεταξύ του FPGA και των ολοκληρωμένων μνήμης και είναι ξεχωριστά για κάθε ολοκληρωμένο (IC10 και IC11). Ο πίνακας 3 δείχνει τις συνδέσεις του FPGA με το IC10 και ο πίνακας 4 τις συνδέσεις με το IC11. Προκειμένου να απενεργοποιηθεί μία SRAM, αρκεί ο ακροδέκτης του αντίστοιχου σήματος επιλογής συσκευής να οδηγηθεί στο ‘1’ (High).

Πίνακας 3.14 – Συνδέσεις του ολοκληρωμένου μνήμης SRAM IC10

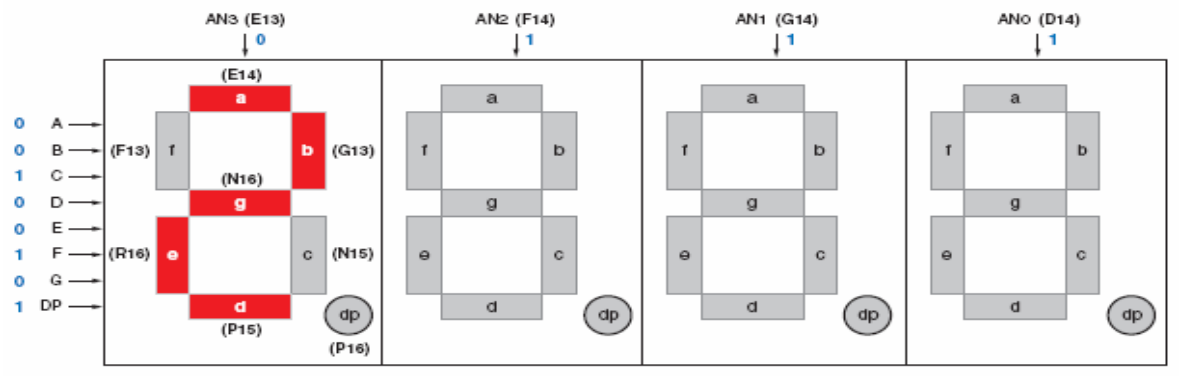
Signal	FPGA Pin
IO15	R1
IO14	P1
IO13	L2
IO12	J2
IO11	H1
IO10	F2
IO9	P8
IO8	D3
IO7	B1
IO6	C1
IO5	C2
IO4	R5
IO3	T5
IO2	R6
IO1	T8
IO0	N7
CE1 (chip enable IC10)	P7
UB1 (upper byte enable IC10)	T4
LB1 (lower byte enable IC10)	P6

Πίνακας 3.15 - Συνδέσεις του ολοκληρωμένου μνήμης SRAM IC11

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

3.3.2.3 Οθόνη επτά τμημάτων, τεσσάρων χαρακτήρων

Η κάρτα Spartan-3 Starter Kit διαθέτει μία οθόνη με τέσσερις χαρακτήρες των επτά τμημάτων η οποία ελέγχεται από τους ακροδέκτες του FPGA που οδηγούνται από σήματα I/O πλήρως ελεγχόμενα από τον χρήστη, όπως φαίνεται στο σχήμα 3.31. Κάθε χαρακτήρας μοιράζεται οκτώ κοινά σήματα που ελέγχουν το κάθε ένα από τα επτά LED των τμημάτων του και κάθε χαρακτήρας έχει μία ξεχωριστή είσοδο ανόδου.



Σχήμα 3.31 – Έλεγχος της οθόνης επτά τμημάτων

Ο αριθμός του ακροδέκτη του FPGA για κάθε λειτουργία της οθόνης φαίνεται στην παρένθεση του σχήματος 3.31. Προκειμένου να ανάψει το LED ενός συγκεκριμένου τμήματος, πρέπει να οδηγηθεί στο λογικό ‘0’ (LOW) το αντίστοιχο σήμα που το ελέγχει και το σήμα ανόδου του χαρακτήρα που επιθυμούμε. Στο παράδειγμα του σχήματος 3.31 στον αριστερότερο χαρακτήρα απεικονίζεται ο αριθμός ‘2’. Το σήμα AN3 έχει οδηγηθεί στο λογικό ‘0’ ‘ώστε να επιτρέψει τις εισόδους για τα LED των τμημάτων του αριστερότερου χαρακτήρα. Τα σήματα εισόδου A έως G και το DP, οδηγούν τα αντίστοιχα τμήματα τα οποία αποτελούν τον χαρακτήρα. Η τιμή λογικό ‘0’ ενεργοποιεί το LED του τμήματος και η τιμή λογικό ‘1’ το απενεργοποιεί. Έτσι η ανάθεση της τιμής λογικό ‘0’ στο σήμα A μεταφράζεται στην ενεργοποίηση του LED ‘a’. Οι άνοδοι των υπόλοιπων ψηφίων (AN2, AN1, AN0) οδηγούνται όλες στην τιμή λογικό ‘1’, και με αυτόν τον τρόπο αγνοούν τις εισόδους των σημάτων A έως G και DP.

Ο πίνακας 5 καταγράφει τις συνδέσεις του FPGA που οδηγούν τα ξεχωριστά LED που σχηματίζουν έναν χαρακτήρα επτά τμημάτων. Ο πίνακας 6 καταγράφει τις συνδέσεις του FPGA που οδηγούν κάθε έναν από τους τέσσερις χαρακτήρες. Τέλος ο πίνακας 7 καταγράφει τις αναθέσεις που πρέπει να γίνουν στα σήματα των τμημάτων για να απεικονίσουμε τους δεκαεξαδικούς χαρακτήρες.

Πίνακας 3.16 – Συνδέσεις του FPGA για την οδήγηση των τμημάτων (ενεργοποίηση στο λογικό ‘0’)

Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

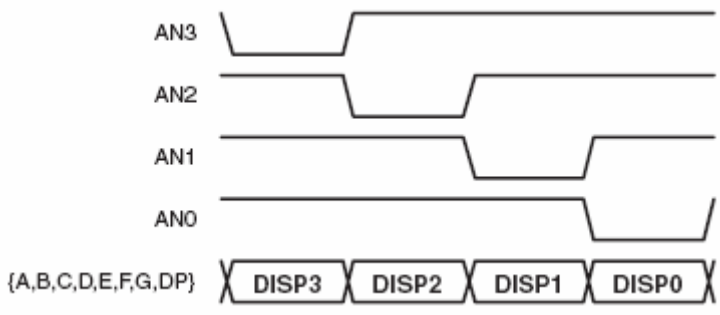
Πίνακας 3.17 – Συνδέσεις του FPGA για την οδήγηση των χαρακτήρων (ενεργοποίηση στο λογικό '0')

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14

Πίνακας 3.18 – Ανάθεση τιμών για την απεικόνιση δεκαεξαδικών χαρακτήρων

Character	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Προκειμένου να απεικονίζονται δεδομένα και στους τέσσερις χαρακτήρες θα πρέπει τα σήματα που ελέγχουν τα LED των τμημάτων να πολυπλεχθούν στο χρόνο, όπως φαίνεται στο σχήμα 3.32. Οι τιμές που οδηγούν τα σήματα των τμημάτων αλλάζουν ανάλογα με το ποιος χαρακτήρας επιθυμούμε να απεικονιστεί και ταυτόχρονα το σήμα ανόδου του αντίστοιχου χαρακτήρα οδηγείται στο λογικό '0'. Λόγω της γρήγορης εναλλαγής των χαρακτήρων ο ανθρώπινος εγκέφαλος αντιλαμβάνεται ότι και οι τέσσερις χαρακτήρες εμφανίζονται ταυτόχρονα.



Σχήμα 3.32 – Χρονική πολύπλεξη της απεικόνισης των χαρακτήρων

Η παραπάνω τεχνική της σάρωσης μειώνει τον απαιτούμενο αριθμό ακροδεκτών I/O για την οδήγηση και των τεσσάρων χαρακτήρων. Αν για κάθε τμήμα και των τεσσάρων χαρακτήρων

δεσμευόταν ένας ακροδέκτης, τότε θα χρειαζόντουσαν 32 ακροδέκτες για να οδηγηθεί μια οθόνη επτά τμημάτων και τεσσάρων χαρακτήρων. Η τεχνική της σάρωσης μειώνει τον αριθμό των απαιτούμενων ακροδεκτών I/O σε 12. Το μειονέκτημα της προσέγγισης αυτής είναι ότι θα πρέπει συνεχώς να τροφοδοτούνται δεδομένα, ένα μάλλον μικρό τίμημα μπροστά στους 20 κερδισμένους ακροδέκτες I/O.

3.3.2.4 Διακόπτες και LEDs

3.3.2.4.1 Διακόπτες

Η κάρτα Spartan-3 Starter Kit διαθέτει οκτώ διακόπτες. Οι διακόπτες φέρουν τις ετικέτες SW7 έως SW0 όπου SW7 ονομάζεται ο αριστερότερος διακόπτης και SW0 ο δεξιότερος. Οι διακόπτες συνδέονται με τους ακροδέκτες του FPGA που αναφέρονται στον πίνακα 3.19.

Πίνακας 3.19 – Οι ακροδέκτες του FPGA που συνδέονται με τους διακόπτες

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Όταν ένας διακόπτης βρίσκεται στην θέση ‘HI’ τότε ο ακροδέκτης του συνδέεται με το V_{CCO}, δηλαδή οδηγείται στην τιμή λογικό ‘1’. Αντίστοιχα όταν βρίσκεται στην θέση ‘LO’ τότε ο ακροδέκτης του συνδέεται με την γείωση του FPGA και οδηγείται στην τιμή λογικό ‘0’.

3.3.2.4.2 Πιεστικοί διακόπτες

Εκτός από τους παραπάνω διακόπτες η κάρτα διαθέτει και τέσσερις πιεστικούς διακόπτες. Οι ονομασίες των διακοπτών αυτών είναι BTN3 έως BTN0 όπου ο BTN3 είναι ο αριστερότερος διακόπτης και ο BTN0 ο δεξιότερος. Οι ακροδέκτες του FPGA με τους οποίους συνδέονται οι διακόπτες αναφέρονται στον πίνακα 3.20. Πιέζοντας έναν διακόπτη αναθέτεται στον αντίστοιχο ακροδέκτη η τιμή λογικό ‘1’.

Πίνακας 3.20 – Οι ακροδέκτες του FPGA που συνδέονται με τους πιεστικούς διακόπτες

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Ο αριστερότερος πιεστικός διακόπτης, BTN3, συνηθίζεται να χρησιμοποιείται σαν το σήμα reset (αρχικοποίηση) του συστήματος, το οποίο είναι ελεγχόμενο από τον χρήστη. Η λειτουργία του συγκεκριμένου διακόπτη δεν διαφέρει σε τίποτα σε σύγκριση με τους υπόλοιπους, απλά παίζει τον ρόλο του σήματος reset όταν αυτό κρίνεται απαραίτητο.

3.3.2.4.3 LED

Η κάρτα Spartan-3 Starter Kit διαθέτει οκτώ ανεξάρτητα LED ακριβώς επάνω από τους πιεστικούς διακόπτες. Οι ονομασίες των LED είναι LED7 έως LED0, όπου το LD7 είναι το

αριστερότερο και το LD0 το δεξιότερο LED αντίστοιχα. Οι συνδέσεις των LED με τους αντίστοιχους ακροδέκτες του FPGA αναφέρονται στον πίνακα 3.21.

Πίνακας 3.21 – Οι ακροδέκτες του FPGA που συνδέονται με τα LEDs

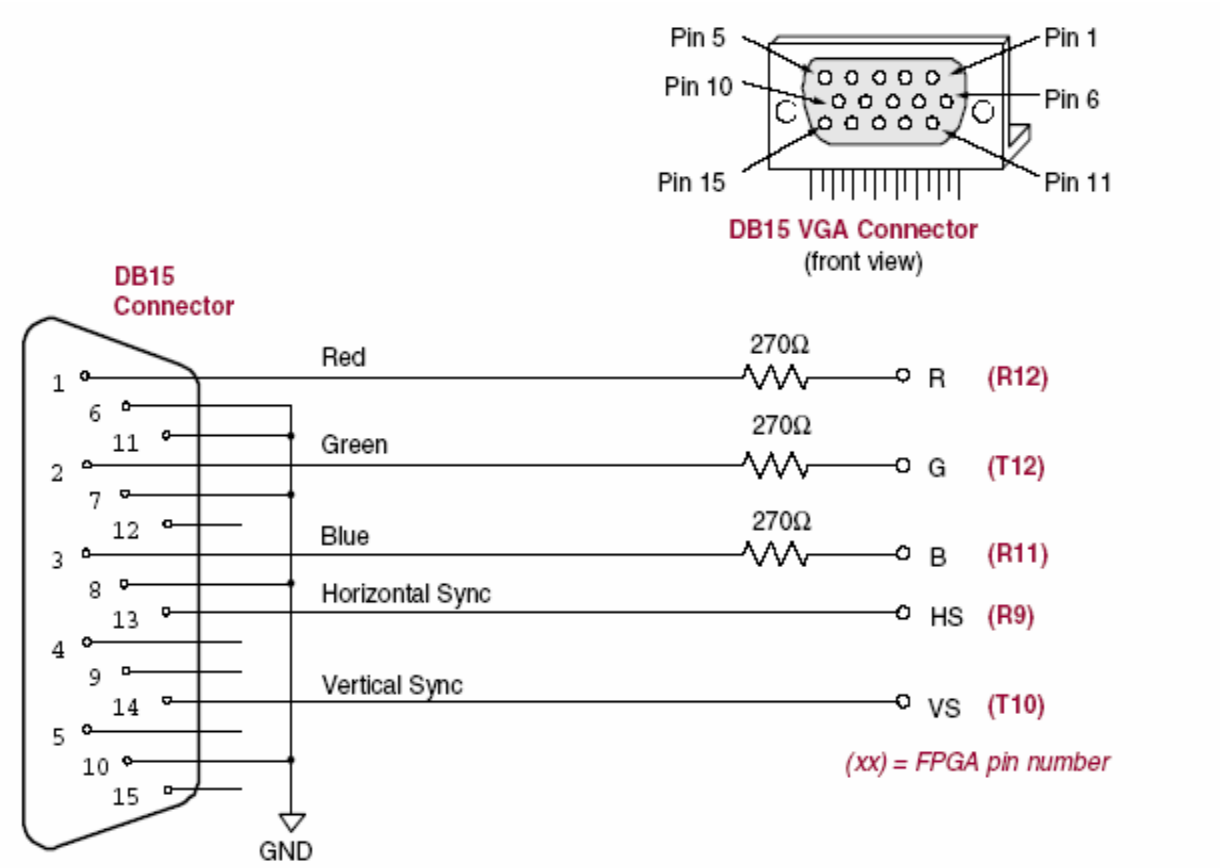
LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

Η κάθοδος του κάθε LED είναι συνδεδεμένη με την γείωση μέσω μιας αντίστασης 270Ω. Προκειμένου να ανάψει ένα LED το σήμα στον αντίστοιχο ακροδέκτη θα πρέπει να οδηγηθεί στην τιμή λογικό '1' (High), δηλαδή σε τιμή αντίθετη με αυτήν που χρειάζεται για το άναμμα των LED της οθόνης 7-segment.

3.3.2.5 Θύρα VGA

3.3.2.5.1 Περιγραφή της θύρας VGA

Η κάρτα Spartan-3 Starter Kit διαθέτει μια θύρα εξόδου για οθόνη VGA, μέσω του VGA connector DB15 που φαίνεται στο σχήμα 3.33.



Σχήμα 3.33 – Οι ακροδέκτες της θύρας VGA και οι συνδέσεις αυτών με το FPGA

Από το σχήμα γίνεται αντιληπτό ότι η κάρτα ελέγχει πέντε σήματα που αφορούν την θύρα VGA. Τα σήματα αυτά είναι το Κόκκινο (RED(R)), το Πράσινο (Green(G)), το Μπλε (Blue(B)), ο Οριζόντιος Συγχρονισμός (Horizontal Sync(HS)) και ο κάθετος συγχρονισμός

(Vertical Sync(VS)). Οι συνδέσεις των σημάτων αυτών με τους αντίστοιχους ακροδέκτες του FPGA αναφέρονται στον πίνακα 3.22.

Πίνακας 3.22 – Τα σήματα της θύρας VGA που ελέγχει η κάρτα και οι αντίστοιχοι ακροδέκτες

Signal	FPGA Pin
Red (R)	R12
Green (G)	T12
Blue (B)	R11
Horizontal Sync (HS)	R9
Vertical Sync (VS)	T10

Οι γραμμές των τριών χρωμάτων (R, G, B) είναι δυνατό να συνδυαστούν με τέτοιο τρόπο ώστε να προκύψουν χρώματα τριών bit, από ένα για τα R, G, B. Τα πιθανά χρώματα ανάλογα με τις τιμές που παίρνουν τα τρία bit συνοψίζονται στον πίνακα 3.23. Τα χρώματα με κατάλληλο δίκτυο αντιστάσεων μετατρέπονται σε σήματα εύρους 0 έως 0.7 V.

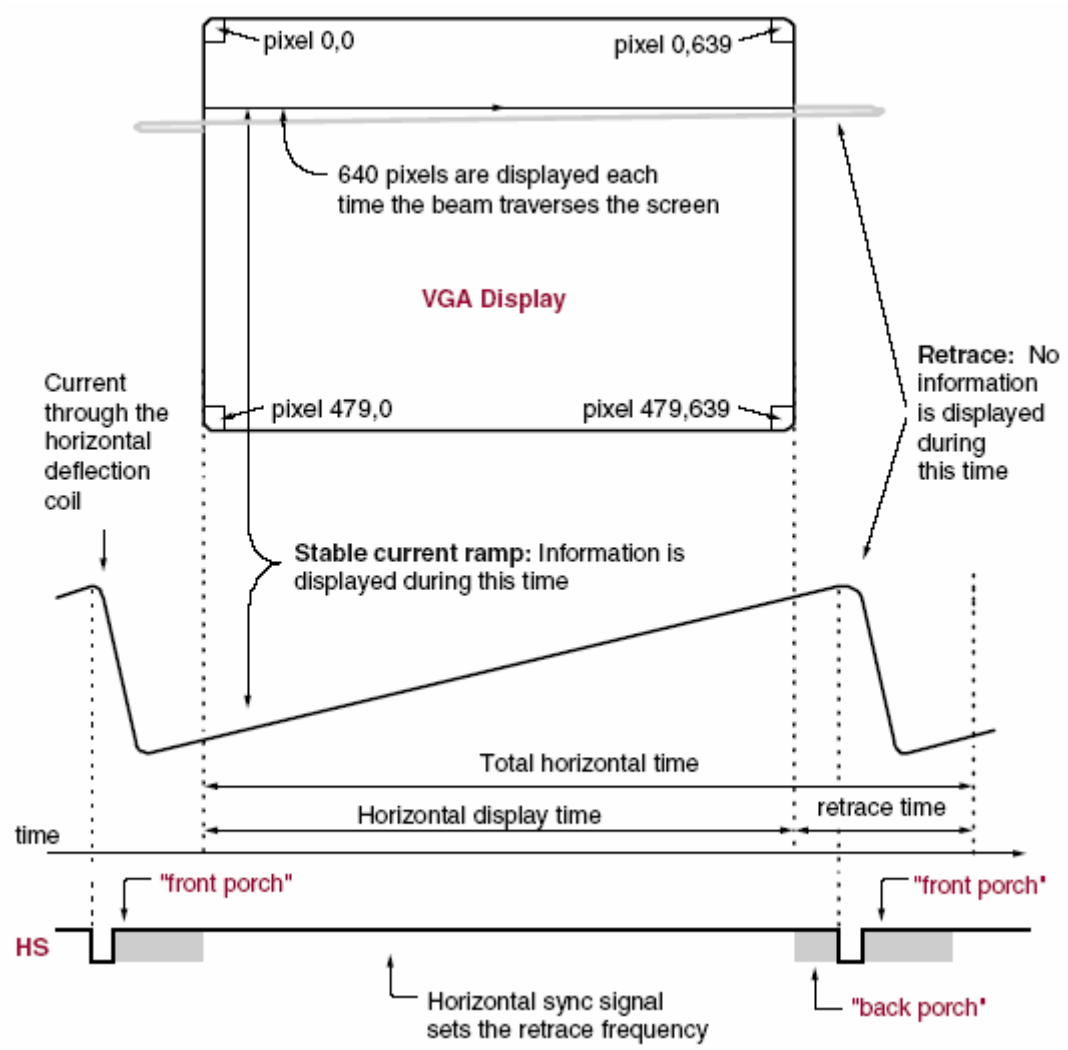
Πίνακας 3.23 – Κατασκευή των χρωμάτων τριών bit από τα R, G, B

Red (R)	Green (G)	Blue (B)	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

3.3.2.5.2 Παράδειγμα οδήγησης οθόνης συχνότητας 60Hz, ανάλυσης 640x480

Η οθόνη αποτελείται από οριζόντιες και κάθετες γραμμές pixel (εικονοστοιχεία), οι οριζόντιες έχουν 640 ενεργά pixel και οι κάθετες 480. Μια δέσμη ηλεκτρονίων κινείται από τα αριστερά προς τα δεξιά και από επάνω προς τα κάτω μεταδίδοντας την κατάλληλη πληροφορία προς απεικόνιση σε όλα τα pixel. Η μετάδοση της πληροφορίας γίνεται μόνο κατά την φορά που αναφέρθηκε προηγουμένως και επομένως κάποιος από τον χρόνο καθυστέρησης οφείλεται στον χρόνο που απαιτείται για την επιστροφή της δέσμης στην αρχική της θέση προκειμένου να ξεκινήσει η διαδικασία της ενημέρωσης των pixel. Το μέγεθος της δέσμης ηλεκτρονίων, η συχνότητα με την οποία η δέσμη σαρώνει την οθόνη και η συχνότητα με την οποία διαμορφώνεται η δέσμη, καθορίζουν την ανάλυση της οθόνης. Ο ελεγκτής της θύρας VGA καθορίζει την ανάλυση της οθόνης παρέχοντας τα κατάλληλα σήματα χρονισμού.

Συνήθως τα δεδομένα εικόνων και βίντεο προέρχονται από μία μνήμη όπου ένα ή περισσότερα byte αντιστοιχούν σε κάθε pixel. Η κάρτα Spartan-3 Starter Kit χρησιμοποιεί τρία bit για κάθε pixel, κατασκευάζοντας τα οκτώ πιθανά χρώματα. Ο ελεγκτής έχει έναν buffer όπου είναι αποθηκευμένη η πληροφορία των pixel και έναν δείκτη σε αυτόν που κινείται αντίστοιχα με την δέσμη που σαρώνει την οθόνη. Την στιγμή που η δέσμη κινείται προς ένα pixel ο ελεγκτής ανακτά και προωθεί στην έξοδο την αντίστοιχη πληροφορία.

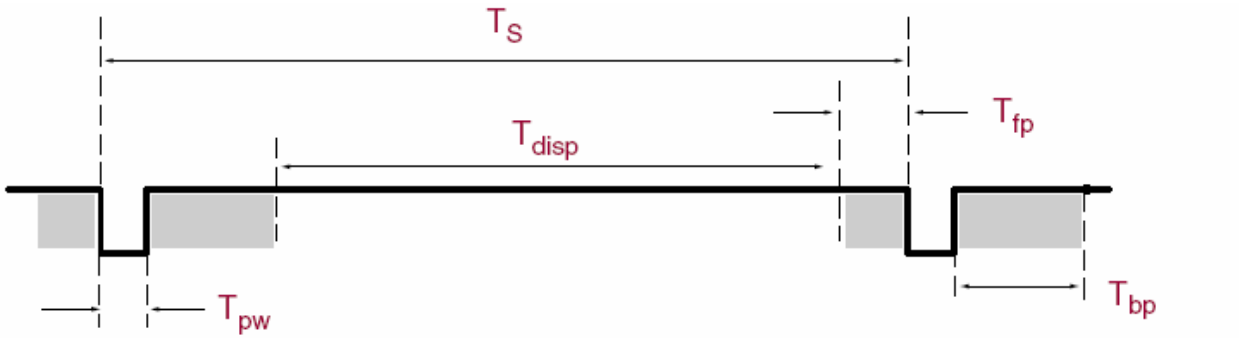


Σχήμα 3.34 – Παράδειγμα χρονισμού

Όπως φαίνεται στο σχήμα 3.34 τα σήματα HS και VS συνδυάζονται ώστε να μεταδίδεται σωστά η πληροφορία του κάθε pixel στον χρόνο που του αναλογεί. Το σήμα HS καθορίζει την συχνότητα με την οποία σαρώνεται μία οριζόντια γραμμή, ενώ το VS την συχνότητα με την οποία όλη η οθόνη σαρώνεται ξανά από την αρχή. Ο χρόνος στον οποίο η πληροφορία ενός pixel είναι δυνατό να μεταδοθεί καθορίζεται από έναν παλμό που αφορά τα pixel. Με την συχνότητα του παλμού αυτού στα 25MHz και του VS στα 60Hz ± 1, οι χρονισμοί των σημάτων που αφορούν τα HS, VS, και για ανάλυση οθόνης 640 γραμμών και 480 στηλών, φαίνονται στον πίνακα 3.24. Ο χρόνος T_{PW} είναι η διάρκεια του παλμού συγχρονισμού, και οι χρόνοι T_{FP} και T_{BP} διαστήματα πριν και μετά τον παλμό συγχρονισμού αντίστοιχα που έχουν προκύψει από παρατηρήσεις για διάφορους τύπους οθονών. Στην διάρκεια των δύο αυτών παλμών δεν είναι δυνατόν να απεικονιστεί πληροφορία.

Πίνακας 3.24 – Χρονισμοί των σημάτων HS, VS για ανάλυση 640x480

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 μs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 μs	640
T_{PW}	Pulse width	64 μs	1,600	2	3.84 μs	96
T_{FP}	Front porch	320 μs	8,000	10	640 ns	16
T_{BP}	Back porch	928 μs	23,200	29	1.92 μs	48



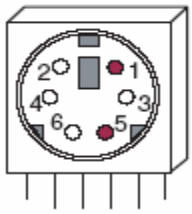
Σχήμα 3.35 – Κυματομορφή βασιζόμενη στους χρονισμούς του πίνακα 3.24

Ένας μετρητής που ελέγχεται από τον παλμό των pixel είναι αρκετός για να ελέγξει τους χρονισμούς των σημάτων HS. Ο μετρητής αυτός θα αναπαριστά την θέση του τρέχοντος pixel σε κάποια δεδομένη γραμμή. Ομοίως ένας μετρητής μπορεί να ελέγχει τα σήματα VS με την διαφορά ότι αυτός θα αυξάνεται μετά από κάθε παλμό HS. Ο μετρητής αυτός θα αναπαριστά την τρέχουσα γραμμή. Ο συνδυασμός των δύο αυτών μετρητών θα μπορούσε να σχηματίζει την διεύθυνση των αντίστοιχων δεδομένων στον buffer όπου είναι αποθηκευμένα.

3.3.2.6 Θύρα PS/2 για ποντίκι / πληκτρολόγιο

3.3.2.6.1 Περιγραφή της θύρας PS/2

Η κάρτα Spartan-3 Starter Kit περιλαμβάνει μία θύρα τύπου PS/2 για ποντίκι και πληκτρολόγιο, μέσω του connector των έξι ακροδεκτών που φαίνεται στο σχήμα 3.36. Στον πίνακα 3.25 αντιστοιχίζονται τα σήματα του connector με τους ακροδέκτες του FPGA. Όπως γίνεται αντιληπτό μόνο οι ακροδέκτες 1 και 5 του connector συνδέονται στο FPGA.



Σχήμα 3.36 – Ο connector PS/2 DIN

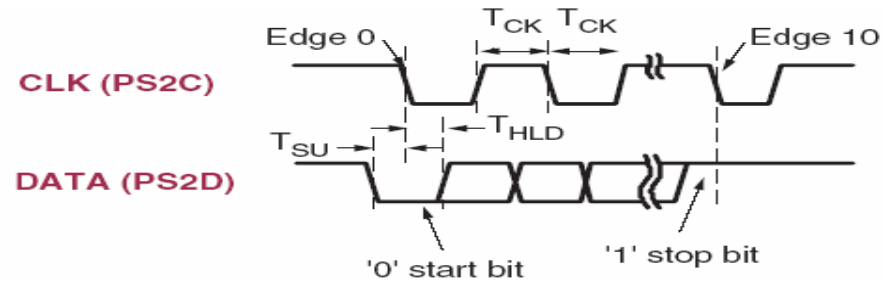
Πίνακας 3.25 – Οι συνδέσεις του connector PS/2 DIN με το FPGA

PS/2 DIN Pin	Signal	FPGA Pin
1	DATA (PS2D)	M15
2	Reserved	—
3	GND	GND
4	Voltage Supply	—
5	CLK (PS2C)	M16
6	Reserved	—

Ο σειριακός δίαυλος PS/2 περιλαμβάνει ρολόι και δεδομένα. Το πληκτρολόγιο και το ποντίκι χρησιμοποιούν πανομοιότυπους χρονισμούς σημάτων και λέξεις των 11 bit που περιλαμβάνουν bit έναρξης (start), τερματισμού (stop) και ισοτιμίας (parity). Παρά τις ομοιότητες όμως τα πακέτα δεδομένων οργανώνονται και ερμηνεύονται διαφορετικά, ανάλογα με το αν έχουμε σύνδεση με ποντίκι ή πληκτρολόγιο. Εκτός αυτού η διεπαφή του πληκτρολογίου επιτρέπει μεταφορά δεδομένων διπλής κατεύθυνσης προκειμένου να ελέγχονται κάποια LED κατάστασης στο πληκτρολόγιο. Οι χρονισμοί του διαδρόμου δεδομένων φαίνονται στον πίνακα 3.26 και το σχήμα 3.37. Το ρολόι και το σήμα των δεδομένων οδηγούνται μονάχα όταν έχουμε μεταφορά δεδομένων, σε αντίθετη περίπτωση διατηρούνται σε μία άεργο κατάσταση όπου εκπέμπεται η τιμή λογικό ‘1’. Όπως φαίνεται στο σχήμα 3.37, το πληκτρολόγιο ή το ποντίκι γράφει ένα bit στην γραμμή των δεδομένων όταν το σήμα του ρολογιού έχει την τιμή λογικό ‘1’ και ο δέκτης διαβάζει την γραμμή των δεδομένων όταν το σήμα του ρολογιού έχει την τιμή λογικό ‘0’.

Πίνακας 3.26 – Οι χρονισμοί του διαδρόμου PS/2

Symbol	Parameter	Min	Max
T_{CK}	Clock High or Low time	30 μs	50 μs
T_{SU}	Data-to-clock setup time	5 μs	25 μs
T_{HLD}	Clock-to-data hold time	5 μs	25 μs

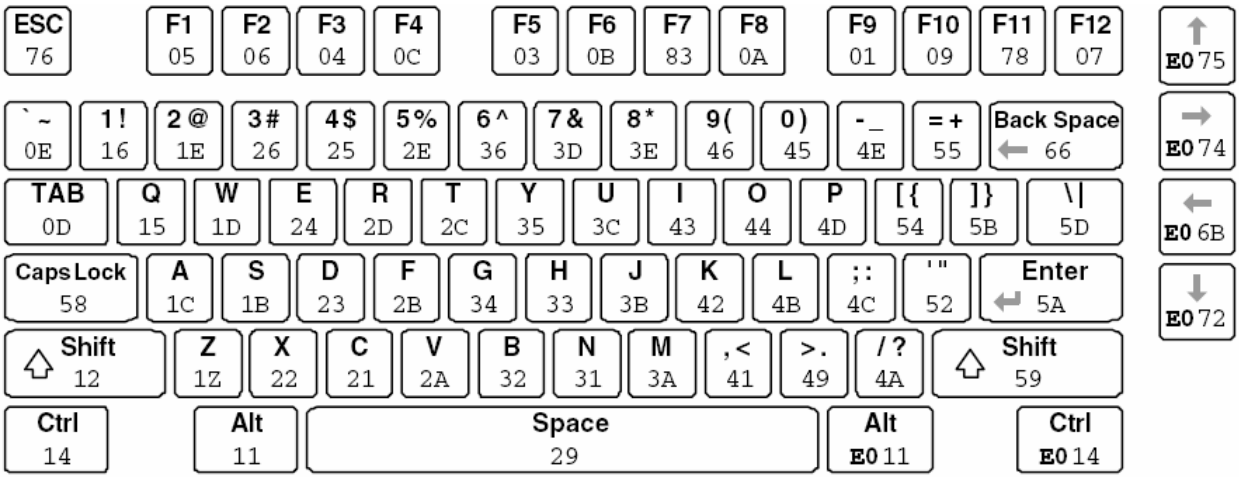


Σχήμα 3.37 – Κυματομορφές χρονισμού του διαδρόμου PS/2

3.3.2.6.2 Πληκτρολόγιο

Ένα πληκτρολόγιο τύπου PS/2 στέλνει έναν ξεχωριστό κωδικό σάρωσης για κάθε πλήκτρο του που πιέζεται. Οι κωδικοί σάρωσης των περισσότερων πλήκτρων φαίνονται στο σχήμα 3.38. Αν ένα πλήκτρο πιεστεί και κρατηθεί πιεσμένο, το πληκτρολόγιο επαναλαμβάνει την αποστολή του κωδικού κάθε 100 ms περίπου. Όταν ένα πλήκτρο αφήνεται το πληκτρολόγιο στέλνει τον κωδικό ‘F0’, ακολουθούμενο από τον κωδικό σάρωσης του πλήκτρου που είχε πιεστεί. Το πληκτρολόγιο στέλνει το ίδιο κωδικό, ανεξάρτητα από το αν υποστηρίζει διαφορετικούς

χαρακτήρες όταν είναι πατημένο το πλήκτρο ‘Shift’ και από το αν το πλήκτρο αυτό ήταν πατημένο ή όχι. Είναι δουλειά του δέκτη να αποφασίσει για το ποιόν χαρακτήρα τελικά έχει λάβει.



Σχήμα 3.38 – Οι κωδικοί σάρωσης του πληκτρολογίου

Κάποια πλήκτρα, extended keys, στέλνουν τον κωδικό ‘E0’ πριν από τον κωδικό σάρωσης του πλήκτρου και μάλιστα είναι δυνατόν να στείλουν πάνω από έναν κωδικό. Στην περίπτωση αυτή όταν το πλήκτρο αφήνεται, το πληκτρολόγιο στέλνει τον κωδικό ‘E0F0’, και έπειτα στέλνεται ο κωδικός σάρωσης του πλήκτρου ή των πλήκτρων που πιέστηκαν.

Όπως αναφέρθηκε και προηγουμένως είναι δυνατόν να λαμβάνει και το πληκτρολόγιο δεδομένα από το υπολογιστικό σύστημα στο οποίο είναι συνδεδεμένο. Μια λίστα με τις πιο συχνές εντολές προς το πληκτρολόγιο παρέχεται στον πίνακα 3.27.

Πίνακας 3.27 – Συνήθεις εντολές προς το πληκτρολόγιο

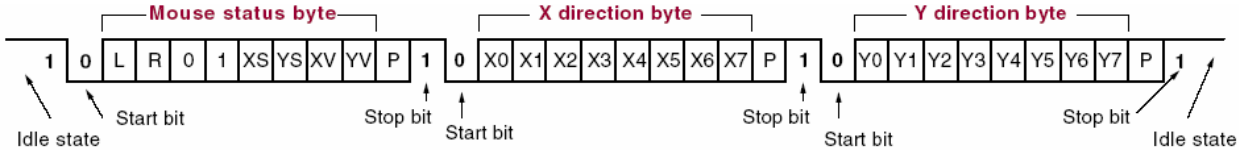
<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>ED</td><td>Ενεργοποίηση/Απενεργοποίηση των LED Num Lock, Caps Lock, Scroll Lock. Το πληκτρολόγιο βεβαιώνει την λήψη της εντολής 'ED' απαντώντας με τον κωδικό 'FA', μετά από τον οποίο το υπολογιστικό σύστημα στέλνει άλλο byte για να θέσει την κατάσταση του LED. Όταν το αντίστοιχο bit είναι 1 τότε το LED ενεργοποιείται.
<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="5">Ignored</td><td>Caps Lock</td><td>Num Lock</td><td>Scroll Lock</td></tr>
</table>
</td></tr>
<tr><td>EE</td><td>Αντηχώ (echo). λαμβάνοντας αυτή την εντολή το πληκτρολόγιο απαντά με τον ίδιο κωδικό σάρωσης 'EE'.</td></tr>
<tr><td>F3</td><td>Ορισμός του ρυθμού επανάληψης κωδικών σάρωσης. Το πληκτρολόγιο βεβαιώνει την λήψη της εντολής απαντώντας με τον κωδικό 'FA', ακολουθούμενο από ένα byte που ορίζει τον ρυθμό επανάληψης.</td></tr>
<tr><td>FE</td><td>Επανεκπομπή. Λαμβάνοντας αυτή την εντολή, το πληκτρολόγιο στέλνει ξανά τον τελευταίο κωδικό σάρωσης</td></tr>
<tr><td>FF</td><td>Reset. Αρχικοποιεί το πληκτρολόγιο.</td></tr>
</table>

Το πληκτρολόγιο στέλνει δεδομένα μόνο κατά την άεργη κατάσταση, όπου το ρολόι και το σήμα των δεδομένων έχουν τιμή λογικό ‘1’. Επειδή το υπολογιστικό σύστημα είναι ο κυρίαρχος του διαύλου, bus master, το πληκτρολόγιο ελέγχει τον δίαυλο προτού επιχειρήσει να στείλει δεδομένα, και εάν το ρολόι έχει τιμή λογικό ‘0’, τότε δεν μπορεί να στείλει έως ότου εισέλθει ο δίαυλος στην άεργη κατάσταση. Το πληκτρολόγιο στέλνει μία λέξη των 11 bit, που ξεκινάει με το bit έναρξης (start bit), με τιμή λογικό ‘0’, έπειτα ακολουθούν τα οκτώ bit του κωδικού

σάρωσης, με πρώτο το LSB, και τα δύο τελευταία bit είναι το bit της ισοτιμίας (parity bit) και το bit τερματισμού (stop bit), με τιμή λογικό ‘1’. Όσο το πληκτρολόγιο στέλνει δεδομένα γεννά 11 παλμούς ρολογιού με συχνότητα 20 με 30 kHz περίπου και τα δεδομένα είναι έγκυρα στην ακμή μετάβασης από το λογικό ‘1’ στο λογικό ‘0’ όπως φαίνεται στο σχήμα 3.37.

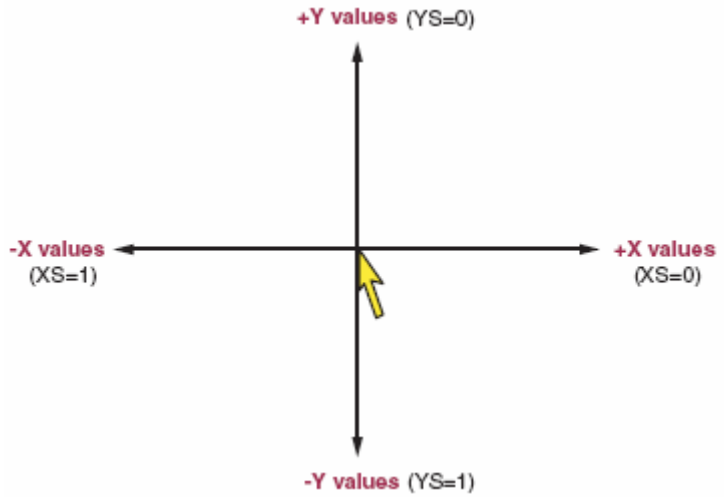
3.3.2.6.3 Ποντίκι

Το ποντίκι γεννά παλμούς ρολογιού και σήματα δεδομένων όταν κινείται. Σε αντίθετη περίπτωση τα σήματα παραμένουν στο λογικό ‘1’ υποδεικνύοντας την άεργη κατάσταση. Κάθε φορά που το ποντίκι κινείται στέλνει τρεις λέξεις των 11 bit. Κάθε μία από τις λέξεις περιλαμβάνει ένα bit έναρξης με τιμή λογικό ‘0’, ακολουθούμενο από 8 bit δεδομένων, με πρώτο το LSB, ένα bit ισοτιμίας και τέλος το bit τερματισμού με τιμή λογικό ‘1’. Επομένως με κάθε κίνηση του ποντικιού στέλνονται 33 bit, όπου τα bit 0, 11 και 22 είναι τα bit έναρξης και τα bit 10, 21 και 32 είναι bit τερματισμού. Τα δεδομένα των οκτώ bit περιλαμβάνουν πληροφορίες κίνησης και είναι έγκυρα στην ακμή μετάβασης του ρολογιού από λογικό ‘1’ σε λογικό ‘0’, με συχνότητα ρολογιού 20 με 30 kHz.



Σχήμα 3.39 – Εκπομπή δεδομένων του ποντικιού σε κάθε του κίνηση

Όπως φαίνεται στο σχήμα 3.40, ένα ποντίκι τύπου PS/2 χρησιμοποιεί ένα σχετικό σύστημα συντεταγμένων όπου όταν το ποντίκι κινείται δεξιά γεννά θετικές τιμές στο πεδίο του X, και όταν κινείται αριστερά γεννά αρνητικές τιμές. Ομοίως ισχύει και για τις τιμές του πεδίου Y όταν το ποντίκι κινείται επάνω ή κάτω αντίστοιχα. Τα bit XS και YS που βρίσκονται στην πρώτη λέξη των 11 bit του σχήματος 13, καθορίζουν το πρόσημο των τιμών του πεδίου X και Y αντίστοιχα. Το λογικό ‘1’ αντιστοιχεί σε αρνητικό πρόσημο.



Σχήμα 3.40 – Το σύστημα συντεταγμένων που χρησιμοποιεί το ποντίκι για την ανίχνευση κίνησης

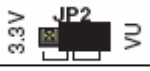
Το μέγεθος των τιμών X και Y αναπαριστούν τον ρυθμό της κίνησης του ποντικιού. Όσο πιο μεγάλη είναι η τιμή, τόσο πιο γρήγορα κινείται το ποντίκι. Τα bit XV και YV της πρώτης λέξης του σχήματος 3.39 δείχνουν εάν τα X και Y έχουν ξεπεράσει τις μέγιστες δυνατές τιμές τους. Η

τιμή λογικό ‘1’ αντιστοιχεί στην κατάσταση αυτή. Στην περίπτωση που το ποντίκι κινείται διαρκώς η μετάδοση των 33 bit πληροφορίας επαναλαμβάνεται κάθε 50 ms περίπου. Τέλος, τα πεδία L και R της πρώτης λέξης του σχήματος 3.39 δείχνουν εάν έχει πατηθεί είτε το αριστερό είτε το δεξί πλήκτρο του ποντικιού αντίστοιχα. Η τιμή λογικό ‘1’ σημαίνει ότι το πλήκτρο έχει πατηθεί.

3.3.2.6.4 Παροχή τάσης

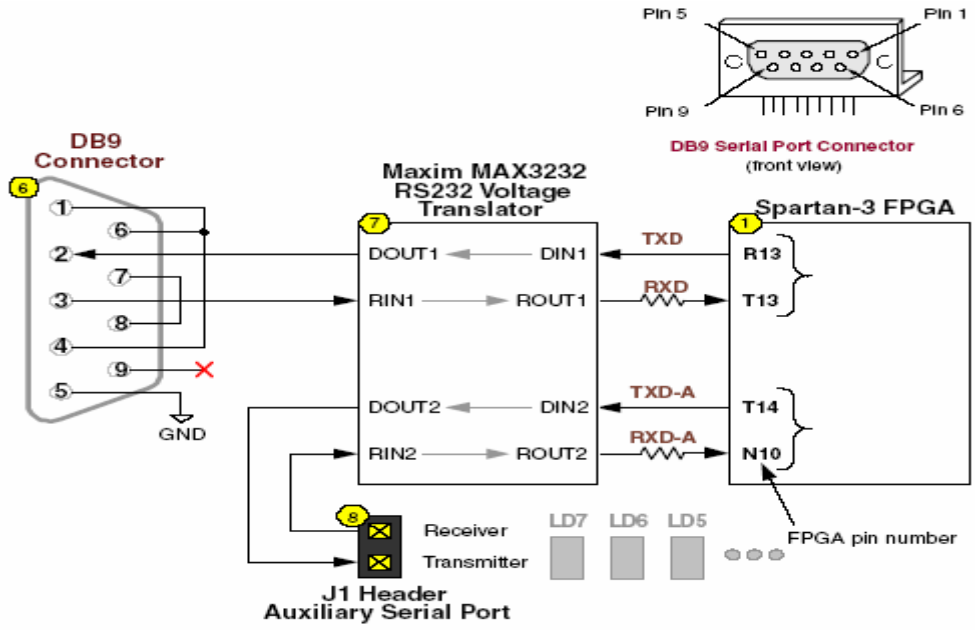
Τα περισσότερα πληκτρολόγια και ποντίκια δουλεύουν εξίσου καλά με παροχή 3.3V ή 5V. Η παροχή είναι δυνατόν να επιλεγεί για την θύρα PS/2 με την βοήθεια των ακροδεκτών ελέγχου JP2 που βρίσκονται ακριβώς επάνω από την θύρα PS/2. Η ονομαστική λειτουργία του FPGA είναι στα 3.3V και προτιμάται παρ’ όλ’ αυτά αν κάποιες συσκευές λειτουργούν μόνο με παροχή 5V, υπάρχει αυτή η δυνατότητα όπως φαίνεται στον πίνακα 3.28.

Πίνακας 3.28 – Επιλογή παροχής τάσης μέσω των ακροδεκτών ελέγχου JP2

PS/2 Port Supply Voltage	Jumper JP2 Setting
3.3V (DEFAULT)	
5V	

3.3.2.7 Σειριακή θύρα RS-232

Η κάρτα Spartan-3 Starter Kit διαθέτει μία σειριακή θύρα RS-232. Τα σήματα εκπομπής και λήψης οδηγούνται μέσω του θηλυκού connector DB9. Ο DB9 είναι μία θύρα τύπου DCE, Data Communications Equipment, και συνδέεται με μία αντίστοιχη θύρα τύπου DTE, Data Terminal Equipment, που βρίσκεται στους περισσότερους προσωπικούς υπολογιστές.



Σχήμα 3.41 – Η σειριακή θύρα RS-232

Στο σχήμα 3.41 φαίνονται οι συνδέσεις του DB9 με τους ακροδέκτες του FPGA, στις οποίες παρεμβάλλεται ο μετατροπέας τάσης Maxim MAX3232 RS-232, ο οποίος μετατρέπει τα επίπεδα τάσης των σημάτων που στέλνει στα κατάλληλα επίπεδα για την σειριακή επικοινωνία και αντιστρόφως πράττει για τα σήματα που έρχονται σειριακά μέσω του DB9 ώστε να πάρουν τις σωστές τιμές που επιτρέπουν την λήψη τους από το FPGA. Έλεγχος ροής υλικού δεν υποστηρίζεται από τον connector. Τα σήματα της σειριακής θύρας DCD (ακροδέκτης 1), DTR (ακροδέκτης 4), και DSR (ακροδέκτης 6) συνδέονται μεταξύ τους, όπως και τα σήματα RTS (ακροδέκτης 7) και CTS (ακροδέκτης 8). Οι συνδέσεις των ακροδεκτών του FPGA με τον μετατροπέα τάσης φαίνονται στον πίνακα 3.29.

Πίνακας 3.29 – Οι συνδέσεις των ακροδεκτών του FPGA με τον μετατροπέα τάσης

Signal	FPGA Pin
RXD	T13
TXD	R13
RXD-A	N10
TXD-A	T14

Η κάρτα διαθέτει και δύο βοηθητικούς ακροδέκτες σειριακής επικοινωνίας RXD-A (δέκτης) και TXD-A (εκπομπός) για την εκτέλεση δοκιμών ή συνδέσεων με άλλου τύπου connector.

3.3.2.8 Πηγές παλμών ρολογιού

Η κάρτα Spartan-3 Starter Kit διαθέτει έναν ταλαντωτή, της σειράς Epson SG-8002JF, που παράγει παλμούς ρολογιού αποκλειστικά στα 50 MHz και μία βοηθητική υποδοχή (socket) για την τοποθέτηση κάποιου άλλου ταλαντωτή. Ο ταλαντωτής των 50 MHz βρίσκεται στο πίσω μέρος της κάρτας και μπορεί να χρησιμοποιηθεί η συχνότητά του είτε όπως έχει είτε να προκύψουν παράγωγα αυτής με την βοήθεια των DCM, Digital Clock Managers, του FPGA. Η υποδοχή για τους εξωτερικούς ταλαντωτές, δέχεται ταλαντωτές οκτώ ακροδεκτών.

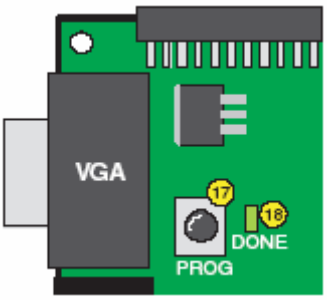
Πίνακας 3.30 – Οι συνδέσεις των ταλαντωτών με ακροδέκτες του FPGA

Oscillator Source	FPGA Pin
50 MHz (IC4)	T9
Socket (IC8)	D9

3.3.2.9 Μέθοδοι διαμόρφωσης (Configuration Modes), λειτουργίες του FPGA

Στις περισσότερες εφαρμογές όταν η κάρτα Spartan-3 starter Kit τροφοδοτείται με ρεύμα ή πιέζεται ο πιεστικός διακόπτης PROG, το FPGA ξεκινάει αυτόματα ένα πρόγραμμα που είναι αποθηκευμένο στην μνήμη Flash. Παρόλα αυτά η κάρτα υποστηρίζει και άλλες μεθόδους διαμόρφωσης με την βοήθεια των ακροδεκτών ελέγχου J8, ο αριθμός 16 στο σχήμα 3.25. Οι δυνατές μέθοδοι διαμόρφωσης και οι αντίστοιχες ρυθμίσεις για τους ακροδέκτες ελέγχου αναλύονται στον πίνακα 3.31. Επιπροσθέτως, είναι απαραίτητη και η ρύθμιση των ακροδεκτών ελέγχου JP1, θέση 3 στο σχήμα 3.25, όπως θα αναφερθεί και στην ενότητα 'Μνήμη δεδομένων διαμόρφωσης, μνήμη Flash'. Οι ονομαστικές ρυθμίσεις του J8 είναι με βραχυκυκλωμένους όλους τους ακροδέκτες ελέγχου.

Η επαναδιαμόρφωση του FPGA είναι δυνατή πιέζοντας τον διακόπτη PROG που βρίσκεται δίπλα στην θύρα VGA και συνδέεται με τον ακροδέκτη προγραμματισμού του FPGA PROG_B. Όταν η διαμόρφωση και ο προγραμματισμός έχει ολοκληρωθεί επιτυχώς, το LED DONE, ακριβώς δίπλα από τον διακόπτη PROG, ενεργοποιείται.



Σχήμα 3.42 – Ο πιεστικός διακόπτης PROG και το LED DONE

Πίνακας 3.31 – Περιγραφή των μεθόδων διαμόρφωσης του FPGA

Μέθοδος Διαμόρφωσης <M0:M1:M2>	Ακροδέκτες Ελέγχου J8	Ακροδέκτες Ελέγχου JP1	Περιγραφή
Master Serial <0:0:0>	GND J8 MODE, Mo M1 M2	JP1 or JP1	Ονομαστική λειτουργία. Το FPGA φορτώνει αυτόματα το πρόγραμμα της μνήμης Flash.
		JP1	Το FPGA επιχειρεί να φορτώσει από κάποια σειριακή πηγή διαμόρφωσης, συνδεδεμένη στις θύρες επέκτασης A2 ή B1.
Slave Serial <1:1:1>	GND J8 MODE, Mo M1 M2	JP1	Κάποια άλλη συσκευή, συνδεδεμένη στις θύρες επέκτασης A2 ή B1, παρέχει σειριακά δεδομένα και ρολόι στο FPGA.
Master Parallel <1:1:0>	GND J8 MODE, Mo M1 M2	JP1	Το FPGA επιχειρεί να φορτώσει από κάποια παράλληλη πηγή διαμόρφωσης, συνδεδεμένη στην θύρα επέκτασης B1.
Slave Parallel <0:1:1>	GND J8 MODE, Mo M1 M2	JP1	Κάποια άλλη συσκευή συνδεδεμένη στην θύρα επέκτασης B1, παρέχει παράλληλα δεδομένα και ρολόι στο FPGA.
JTAG <1:0:1>	GND J8 MODE, Mo M1 M2	JP1	Το FPGA αναμένει για διαμόρφωση μέσω της διεπαφής JTAG.

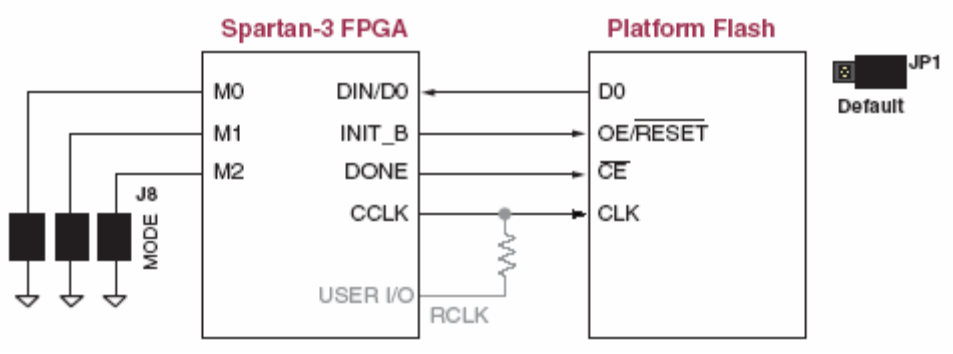
3.3.2.10 Μνήμη δεδομένων διαμόρφωσης, μνήμη Flash

Η κάρτα Spartan-3 Starter Kit διαθέτει μία μνήμη Flash Prom για την αποθήκευση δεδομένων διαμόρφωσης και προγραμματισμού του FPGA καθώς και μη μεταβλητών δεδομένων, συμπεριλαμβανομένου και κώδικα εφαρμογής. Προκειμένου να διαμορφωθεί το FPGA από την μνήμη Flash θα πρέπει όλοι οι ακροδέκτες ελέγχου του J8 να είναι βραχυκυκλωμένοι. Η μνήμη Flash έχει τρεις δυνατές καταστάσεις λειτουργίας ελεγχόμενες από τους ακροδέκτες ελέγχου JP1, που βρίσκονται δίπλα στο ολοκληρωμένο της μνήμης Flash. Ο πίνακας 3.32 συνοψίζει και περιγράφει τις τρεις καταστάσεις λειτουργίας.

Πίνακας 3.32 – Λειτουργίες της μνήμης Flash ανάλογα με της κατάσταση των ακροδεκτών ελέγχου

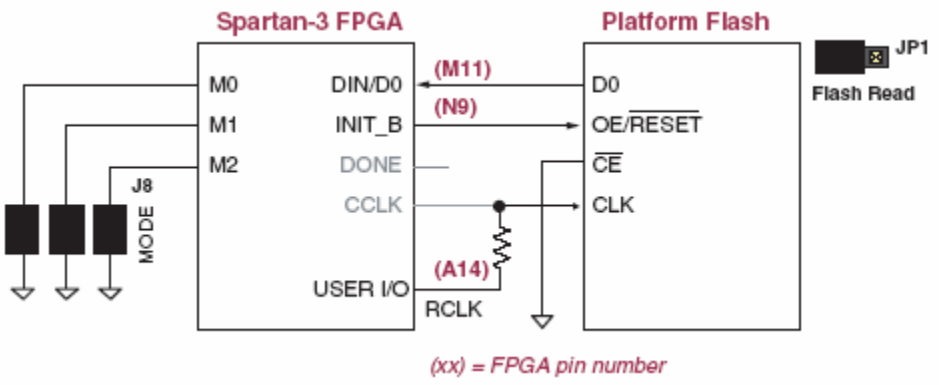
Λειτουργία	Ακροδέκτες Ελέγχου JP1	Περιγραφή
Default		Το FPGA φορτώνει από την Flash. Δεν είναι δυνατή η ανάκτηση πρόσθετων δεδομένων.
Flash Read		Το FPGA φορτώνει από την μνήμη Flash, η οποία είναι μονίμως ενεργοποιημένη.. Το FPGA μπορεί να διαβάσει πρόσθετα δεδομένα από την Flash
Disable		Οι ακροδέκτες ελέγχου όλοι ανοιχτοκυκλωμένοι. Η μνήμη Flash απενεργοποιημένη. Άλλες πηγές δεδομένων διαμόρφωσης χρησιμοποιούνται από το FPGA.

Η λειτουργία Default (ονομαστική) είναι αυτή που συνηθίζεται περισσότερο. Το FPGA διαμορφώνεται από την μνήμη Flash και μόλις η διαμόρφωση ολοκληρωθεί και ενεργοποιηθεί το LED DONE, η μνήμη Flash απενεργοποιείται.



Σχήμα 3.43 – Η λειτουργία Default

Το μέγεθος της μνήμης Flash PROM είναι 2Mbit. Το FPGA χρειάζεται κάτι λιγότερο από 1Mbit για τα δεδομένα διαμόρφωσης. Το υπόλοιπο που απομένει μπορεί να χρησιμοποιηθεί για αποθήκευση σταθερών δεδομένων, όπως σειριακοί αριθμοί (serial numbers), συντελεστές φίλτρων, το ID ενός Ethernet MAC ή κώδικας εφαρμογής για έναν επεξεργαστή όπως είναι ο MicroBlaze. Την δυνατότητα αυτή προσφέρει η λειτουργία Flash Read, όπου μετά την διαμόρφωση του FPGA η μνήμη Flash δεν απενεργοποιείται και είναι σε θέση να τροφοδοτεί το FPGA με σειριακά δεδομένα.

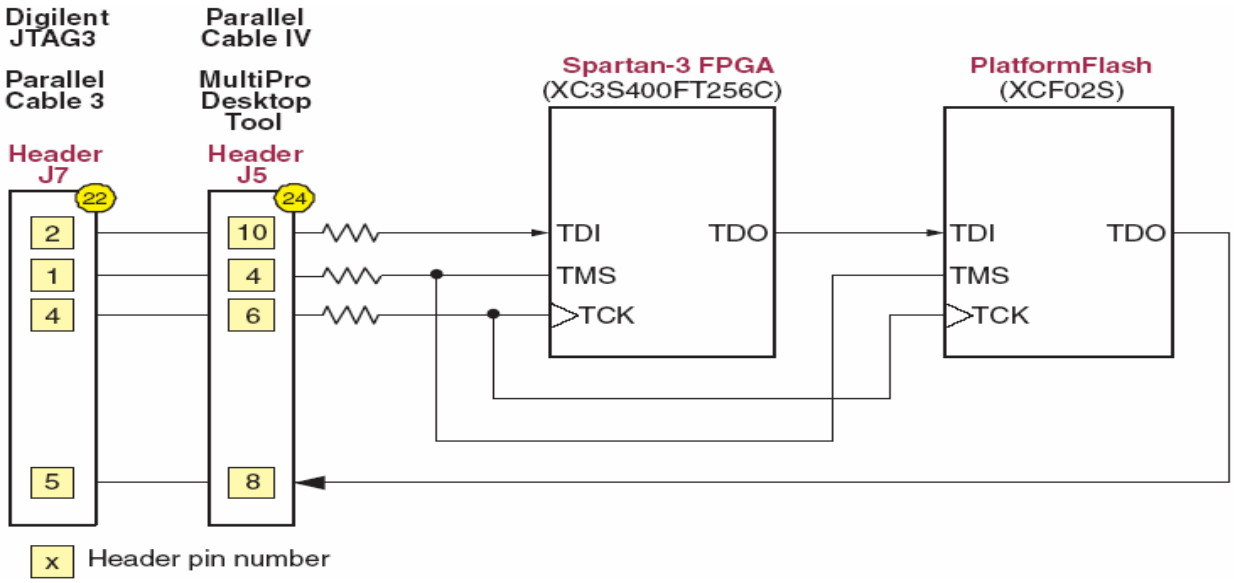


Σχήμα 3.44 – Η λειτουργία Flash Read

Στην λειτουργία Disable όπως είναι κατανοητό η μνήμη Flash είναι απενεργοποιημένη, πράγμα που επιτρέπει την διαμόρφωση του FPGA από άλλες συσκευές συνδεδεμένες στις θύρες επέκτασης.

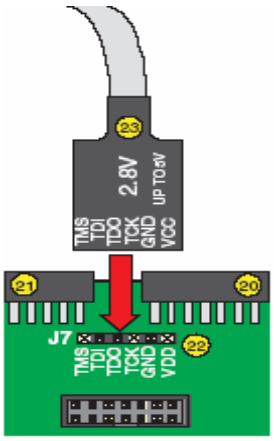
3.3.2.11 Θύρα προγραμματισμού / εκσφαλμάτωσης JTAG

Η κάρτα Spartan-3 Starter Kit διαθέτει μία αλυσίδα JTAG για προγραμματισμό και εκσφαλμάτωση. Τόσο το FPGA Spartan-3 όσο και η μνήμη Flash, αποτελούν μέρος αυτής της αλυσίδας, όπως φαίνεται στο σχήμα 3.45. Παράλληλα υπάρχουν και δύο σειρές από ακροδέκτες (Header J7 και Header J5) που υποστηρίζουν την οδήγηση των σημάτων JTAG από διάφορα υποστηριζόμενα καλώδια.



Σχήμα 3.45 – Η αλυσίδα JTAG της κάρτας Spartan-3 Starter Kit

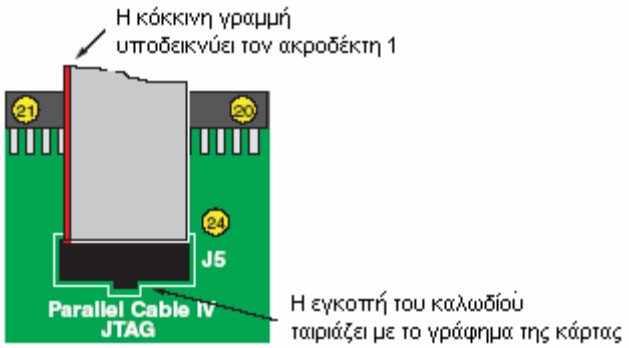
Η σειρά ακροδεκτών J7 αποτελείται από έξι ακροδέκτες, που βρίσκονται ακριβώς κάτω από τις δύο θύρες επέκτασης, και στους οποίους εφαρμόζεται το παράλληλο προς JTAG καλώδιο της Digilent, JTAG3. Το καλώδιο εφαρμόζει κάθετα στην κάρτα και πρέπει οι ετικέτες των σημάτων στην άκρη του να είναι ευθυγραμμισμένες με τις αντίστοιχές τους στην κάρτα. Αφού συνδεθεί με την παράλληλη θύρα ενός προσωπικού υπολογιστή το καλώδιο είναι συμβατό με το πρόγραμμα λογισμικού iMPACT της Xilinx.



Σχήμα 3.46 – Σύνδεση του καλωδίου της Digilent JTAG3

Επίσης υποστηρίζεται και το καλώδιο Parallel Cable 3 (PC3) της Xilinx, για την σειρά ακροδεκτών J7.

Η σειρά ακροδεκτών J5 υποστηρίζει τα καλώδια MultiPro Desktop Tool και Parallel Cable IV (PC IV) της Xilinx. Τα καλώδια είναι ταινίες των δεκατεσσάρων ακροδεκτών και όταν εφαρμόζονται στην κάρτα θα πρέπει η κόκκινη γραμμή να βρίσκεται στα αριστερά, υποδεικνύοντας τον ακροδέκτη 1. Παρότι τα καλώδια υποστηρίζουν πολλές μεθόδους διαμόρφωσης, η κάρτα Spartan-3 Starter Kit υποστηρίζει μόνο την διαμόρφωση JTAG.



Σχήμα 3.47 – Σύνδεση καλωδίου με την σειρά ακροδεκτών J5

3.3.2.12 Διανομή ισχύος

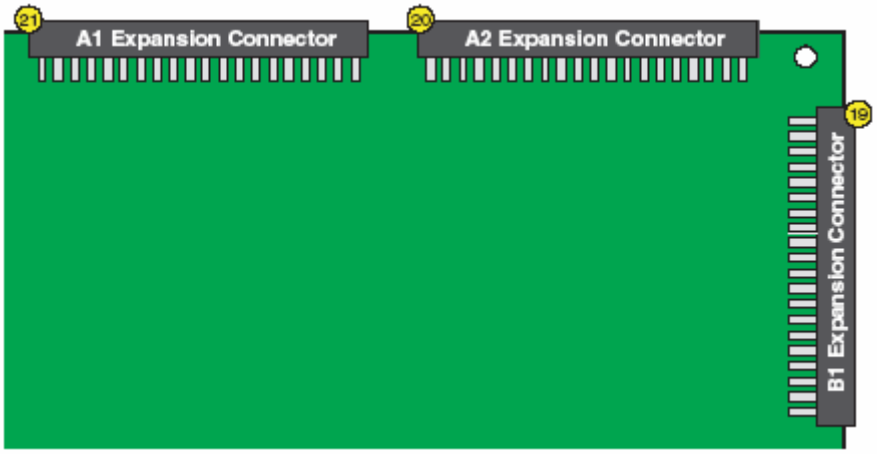
Η κάρτα Spartan-3 Starter Kit χρειάζεται τροφοδοσία 5V DC για να λειτουργήσει. Η τροφοδοσία παρέχεται στην υποδοχή του μετασχηματιστή, στην θέση 25 του σχήματος 3.25. Δεν υπάρχει διακόπτης τροφοδοσίας επάνω στην κάρτα, ενεργοποιείται μονάχα το LED POWER όταν η τροφοδοσία παρέχεται σωστά.

Τα 5V με την σειρά τους τροφοδοτούν έναν ρυθμιστή τάσης των 3.3V, στην θέση 27 του σχήματος 2.25, ο οποίος τροφοδοτεί τα περισσότερα μέρη της κάρτας καθώς και το V_{CCO} των σημάτων εισόδου / εξόδου (I/O) του FPGA. Αυτός ο ρυθμιστής τάσης, τροφοδοτεί δύο άλλους, έναν των 2.5V στη θέση 28 του σχήματος 3.26, και έναν των 1.5V στη θέση 29 του σχήματος 3.26.

Κάποιοι ακροδέκτες διαμόρφωσης όπως οι DONE, PROG_B, CCLK και οι ακροδέκτες JTAG του FPGA τροφοδοτούνται από την τάση V_{CCAUX}, την οποία παράγει ο ρυθμιστής των 2.5V. Την ίδια τάση χρησιμοποιούν και οι Digital Clock Managers. Τέλος το FPGA λειτουργεί εσωτερικά με την τάση V_{CCINT}, που παράγει ο ρυθμιστής των 1.5V.

3.3.2.13 Θύρες επέκτασης

Η κάρτα Spartan-3 Starter Kit διαθέτει τρεις θύρες επέκτασης, A1, A2, B1, των 40 ακροδεκτών η κάθε μία. Ο πίνακας 3.33 συνοψίζει τις δυνατότητες της κάθε θύρας. Η θύρα A1 υποστηρίζει το πολύ 32 σήματα I/O ορισμένα από τον χρήστη ενώ οι άλλες δύο υποστηρίζουν το πολύ 34. Βέβαια δεν είναι όλοι οι ακροδέκτες των θυρών εντελώς ανεξάρτητοι, καθώς κάποιοι μοιράζονται τα ίδια σήματα με κάποιες λειτουργίες του FPGA, όπως φαίνεται και στον πίνακα 3.33.

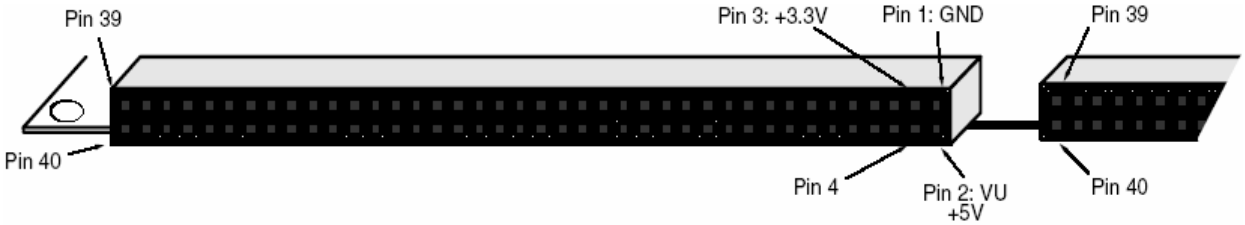


Σχήμα 3.48 – Οι τρεις θύρες επέκτασης της κάρτας Spartan-3 Starter Kit

Πίνακας 3.33 – Χαρακτηριστικά των σημάτων των θυρών επέκτασης

Connector	User I/O	SRAM Address	JTAG	Serial Configuration	Parallel Configuration
A1	32	√	√		
A2	34			√	
B1	34			√	√

Σε κάθε θύρα επέκτασης, ο ακροδέκτης 1 αντιστοιχεί στη γείωση, ο ακροδέκτης 2 στα 5V DC και ο ακροδέκτης 3 στα 3.3V DC.



Σχήμα 3.49 – Οι τρεις από τους 40 ακροδέκτες κάθε θύρας οδηγούν τα ίδια σήματα

Τα σήματα των ακροδεκτών της θύρας επέκτασης A1, καθώς και οι συνδέσεις τους με τους αντίστοιχους ακροδέκτες του FPGA αναγράφονται στον πίνακα 3.34. Η θύρα A1 μοιράζεται τα σήματα των διευθύνσεων της SRAM και τα σήματα ελέγχου αυτής OE και WE. Ομοίως ισχύει και για τα σήματα της αλυσίδας JTAG που οδηγούνται στους ακροδέκτες 36 έως 40.

Πίνακας 3.34 – Οι ακροδέκτες της θύρας επέκτασης A1

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)	5	6	(L5) SRAM A0	ADR1
DB1	(T8)	7	8	(N3) SRAM A1	ADR2
DB2	(R6)	9	10	(M4) SRAM A2	ADR3
DB3	(T5)	11	12	(M3) SRAM A3	ADR4
DB4	(R5)	13	14	(L4) SRAM A4	ADR5
DB5	(C2)	15	16	(G3) SRAM WE#	WE
DB6	(C1)	17	18	(K4) SRAM OE#	OE
DB7	(B1)	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

Τα σήματα των ακροδεκτών της θύρας επέκτασης A2 και οι συνδέσεις τους με τους αντίστοιχους ακροδέκτες του FPGA αναγράφονται στον πίνακα 3.35. Οι περισσότεροι ακροδέκτες συνδέονται μόνο με το FPGA και δεν μοιράζονται σήματα με άλλες λειτουργίες. Ο ακροδέκτης 35 συνδέεται με την υποδοχή του βοηθητικού ταλαντωτή, ενώ οι ακροδέκτες 36 έως 40 οδηγούν τα απαραίτητα σήματα για την σειριακή μέθοδο διαμόρφωσης του FPGA.

Πίνακας 3.35 – Οι ακροδέκτες της θύρας επέκτασης A2

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

Τα σήματα των ακροδεκτών της θύρας επέκτασης B1 και οι συνδέσεις τους με τους αντίστοιχους ακροδέκτες του FPGA αναγράφονται στον πίνακα 3.36. Οι ακροδέκτες 36 έως 40 οδηγούν τα ίδια σήματα με τους αντίστοιχούς της θύρας A2. Οι ακροδέκτες 5, 7, 9, 11, 13, 15, 17, 19 και 20 οδηγούν τα απαραίτητα σήματα για την παράλληλη μέθοδο διαμόρφωσης του FPGA.

Πίνακας 3.36 - Οι ακροδέκτες της θύρας επέκτασης B1

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(C10)	PB-ADR0
PB-DB0	(T3) FPGA RD_WR_B config	5	6	(E10)	PB-ADR1
PB-DB1	(N11) FPGA D1 config	7	8	(C11)	PB-ADR2
PB-DB2	(P10) FPGA D2 config	9	10	(D11)	PB-ADR3
PB-DB3	(R10) FPGA D3 config	11	12	(C12)	PB-ADR4
PB-DB4	(T7) FPGA D4 config	13	14	(D12)	PB-ADR5
PB-DB5	(R7) FPGA D5 config	15	16	(E11)	PB-WE
PB-DB6	(N6) FPGA D6 config	17	18	(B16)	PB-OE
PB-DB7	(M6) FPGA D7 config	19	20	(R3) FPGA CS_B config	PB-CS
PB-CLK	(C15)	21	22	(C16)	MB1-DB0
MB1-DB1	(D15)	23	24	(D16)	MB1-DB2
MB1-DB3	(E15)	25	26	(E16)	MB1-DB4
MB1-DB5	(F15)	27	28	(G15)	MB1-DB6
MB1-DB7	(G16)	29	30	(H15)	MB1-ASTB
MB1-DSTB	(H16)	31	32	(J16)	MB1-WRITE
MB1-WAIT	(K16)	33	34	(K15)	MB1-RESET
MB1-INT	(L15)	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

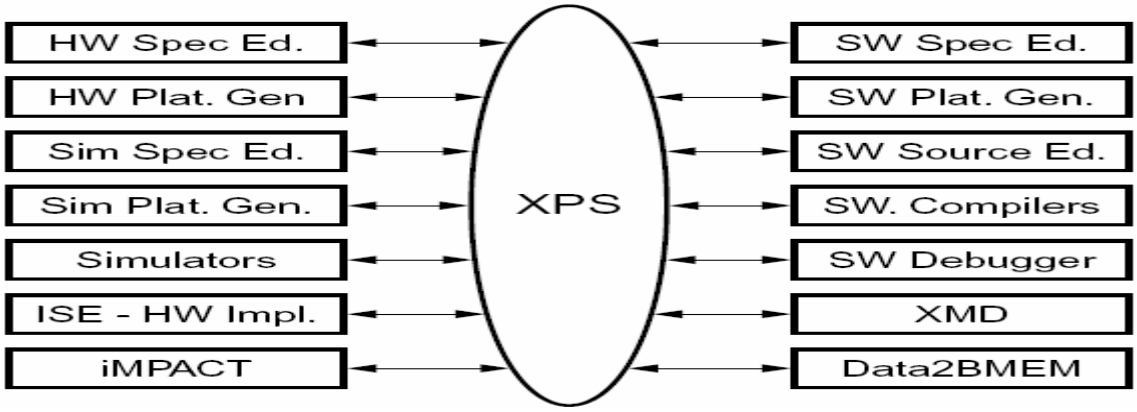
Πηγές

1. Digilent: “Digilab 2 Reference Manual”
2. Xilinx: “Spartan-II 2.5 V FPGA Family: Complete Data Sheet
3. Digilent: “Digilab DIO2 Reference Manual”
4. Xilinx: “Spartan-3 Starter Kit Board User Guide”
5. Xilinx: “Spartan-3 FPGA Family: Complete Data Sheet”

4. Η αρχιτεκτονική των εργαλείων σχεδίασης των ενσωματωμένων συστημάτων, του EDK

4.1 Γενικά

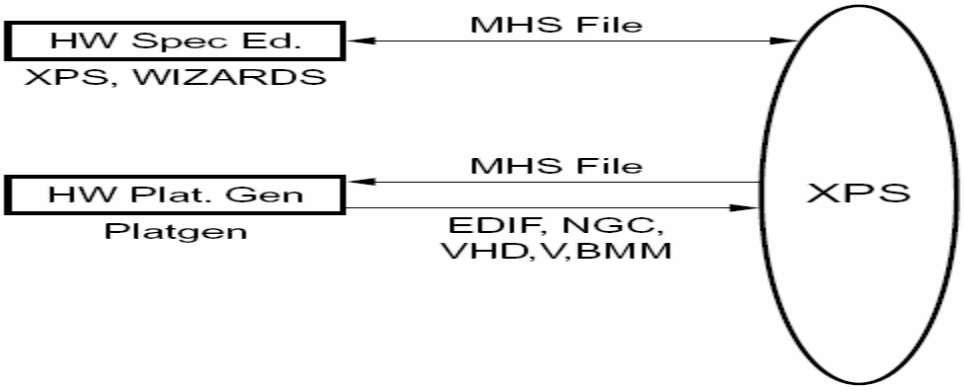
Το EDK, Embedded Development Kit, αποτελεί μία συλλογή σχεδιαστικών εργαλείων, καθώς και πολλών περιφερειακών, με τα οποία είναι δυνατόν να χτιστεί ένα ενσωματωμένο σύστημα επεξεργαστή, χρησιμοποιώντας τον MicroBlaze ή τον PowerPC. Τα σχεδιαστικά εργαλεία χωρίζονται σε δύο μεγάλες κατηγορίες, μία που αφορά τα εργαλεία σχεδιασμού του υλικού (hardware) και μία που αφορά το λογισμικό (software). Ένα σχεδιάγραμμα της αρχιτεκτονικής των εργαλείων (EST, Embedded System Tools) φαίνεται στο σχήμα 4.1. Όπως γίνεται αντιληπτό όλα τα εργαλεία λειτουργούν σε συνεργασία με το XPS, Xilinx Platform Studio.



Σχήμα 4.1 – Η αρχιτεκτονική των EST

Το κομμάτι του σχεδιασμού του υλικού περιλαμβάνει την αυτόματη δημιουργία μιας πλατφόρμας υλικού, hardware platform, και την μετέπειτα τροποποίηση και επέκταση αυτής, ώστε να περιλαμβάνει τις επιθυμητές hardware λειτουργίες του χρήστη. Ομοίως, στο κομμάτι που αφορά το λογισμικό, αφού τα εργαλεία δημιουργήσουν μία πλατφόρμα λογισμικού, software platform, ο χρήστης προσθέτει τις δικές του εφαρμογές. Εκτός από το κομμάτι του υλικού και του λογισμικού, ο σχεδιασμός ενός συστήματος μπορεί να περιλαμβάνει και το κομμάτι της προσομοίωσης ή της επιβεβαίωσης του συστήματος, στο οποίο δημιουργούνται αυτόματα μοντέλα προσομοίωσης τα οποία βασίζονται στο υλικό και το λογισμικό του συστήματος.

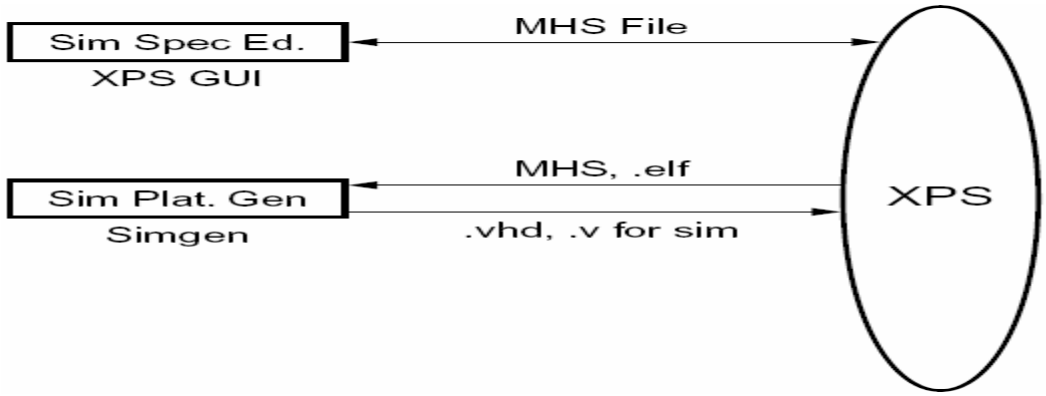
Η δημιουργία της πλατφόρμας υλικού, αναπαριστάται στο σχήμα 4.2.



Σχήμα 4.2 – Δημιουργία πλατφόρμας υλικού

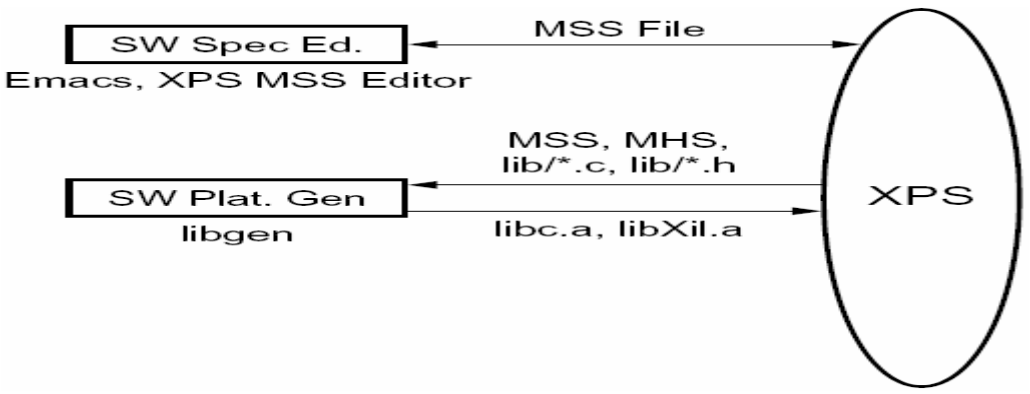
Η πλατφόρμα υλικού καθορίζεται πλήρως από το αρχείο MHS, Microprocessor Hardware Specification και αποτελείται από έναν ή περισσότερους επεξεργαστές και περιφερειακά, συνδεδεμένα στους διαδρόμους δεδομένων των επεξεργαστών. Τα περιφερειακά μπορούν είτε να δίνονται έτοιμα από το EDK, με την μορφή της πνευματικής ιδιοκτησίας, Intellectual Property (IP), είτε να περιγράφονται από τον χρήστη με βάση συγκεκριμένες οδηγίες. Το αρχείο MHS καθορίζει την αρχιτεκτονική και την συνδεσιμότητα του συστήματος, των χάρτη διευθύνσεων του κάθε περιφερειακού καθώς και τις διαμορφώσιμες ιδιότητές τους. Το εργαλείο Platform Generator (PlatGen) δημιουργεί την πλατφόρμα υλικού χρησιμοποιώντας σαν είσοδο το αρχείο MHS. Το PlatGen δημιουργεί τα αρχεία του netlist σε διάφορες μορφές (NGC, EDIF), καθώς και κώδικα VHDL που επιτρέπει στον χρήστη να προσθέσει και άλλα components στην αυτόματα δημιουργημένη πλατφόρμα. Στη συνέχεια με τα δημιουργημένα αρχεία και με κατάλληλα εργαλεία σύνθεσης παράγεται το bitstream, το οποίο τελικά διαμορφώνει το FPGA.

Η πλατφόρμα επιβεβαίωσης, verification platform, στηρίζεται στην πλατφόρμα υλικού. Το αρχείο MHS επεξεργάζεται από το εργαλείο Simgen, και δημιουργεί αρχεία κατάλληλα για προσομοίωση που αντιστοιχούν στους επεξεργαστές και τα περιφερειακά του συστήματος. Και στην περίπτωση αυτή είναι δυνατόν να προστεθούν στοιχεία από τον χρήστη στην αυτόματα δημιουργημένη πλατφόρμα. Εάν και η εφαρμογή που θα τρέχει στο hardware είναι διαθέσιμη σε εκτελέσιμη μορφή, τότε είναι δυνατό να αρχικοποιηθούν οι μνήμες στην πλατφόρμα επιβεβαίωσης.



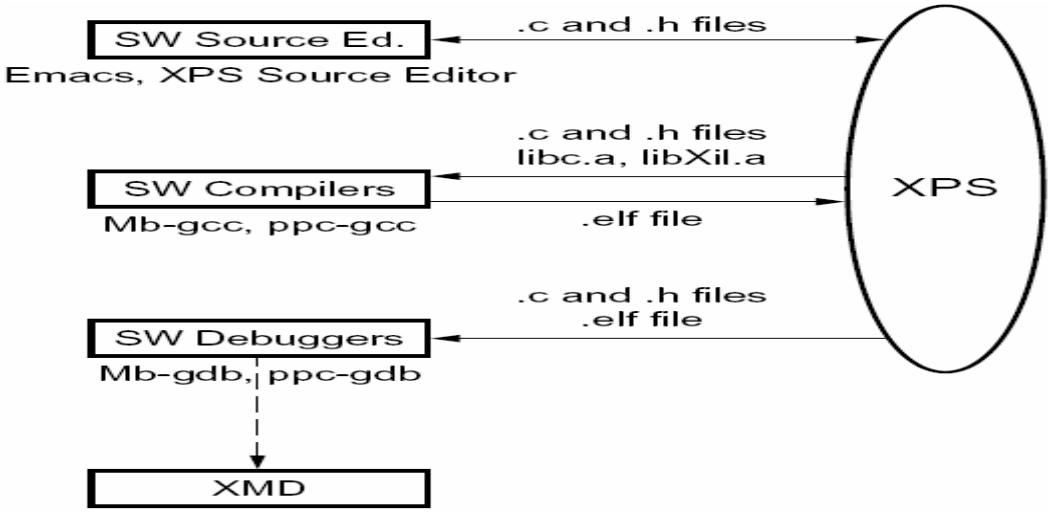
Σχήμα 4.3 – Η πλατφόρμα επιβεβαίωσης

Αντίστοιχο ρόλο με το αρχείο MHS, παίζει το αρχείο MSS, Microprocessor Software Specification, για την πλατφόρμα λογισμικού. Στο αρχείο αυτό καθορίζονται οι οδηγοί (drivers) και οι βιβλιοθήκες για τα περιφερειακά και τους επεξεργαστές, οι συσκευές εισόδου / εξόδου, οι ρουτίνες χειρισμού διακοπών, και άλλα στοιχεία που σχετίζονται με το λογισμικό του συστήματος. Το αρχείο MSS είναι μία από τις εισόδους του εργαλείου Library Generator (LibGen).



Σχήμα 4.4 – Η πλατφόρμα λογισμικού

Η εφαρμογή λογισμικού είναι ο κώδικας που τρέχει στις πλατφόρμες υλικού και λογισμικού. Ο πηγαίος κώδικας των εφαρμογών είναι γραμμένος είτε σε γλώσσα υψηλού επιπέδου (C ή C++), είτε σε assembly. Τα αρχεία του πηγαίου κώδικα μεταγλωττίζονται (compile) και συνδέονται (link) σε ένα εκτελέσιμο αρχείο τύπου ELF (Executable and Link Format). Το EDK διαθέτει εργαλεία μεταγλώττισης, τόσο για τα τον επεξεργαστή MicroBlaze όσο και για τον PowerPC, αλλά μπορούν να χρησιμοποιηθούν και άλλοι compilers. Τα εργαλεία XMD, Xilinx Microprocessor Debugger, και GNU Debugger (GDB), χρησιμοποιούνται μαζί για την εκσφαλμάτωση της εφαρμογής λογισμικού. Το XMD παρέχει μία ομάδα εντολών προσομοίωσης, ενώ προαιρετικά μπορεί να συνδεθεί με μια πλατφόρμα υλικού που βρίσκεται σε λειτουργία και να επιτρέψει στον GDB να τρέξει την εφαρμογή του χρήστη.



Σχήμα 4.5 – Δημιουργία και επιβεβαίωση εφαρμογής λογισμικού

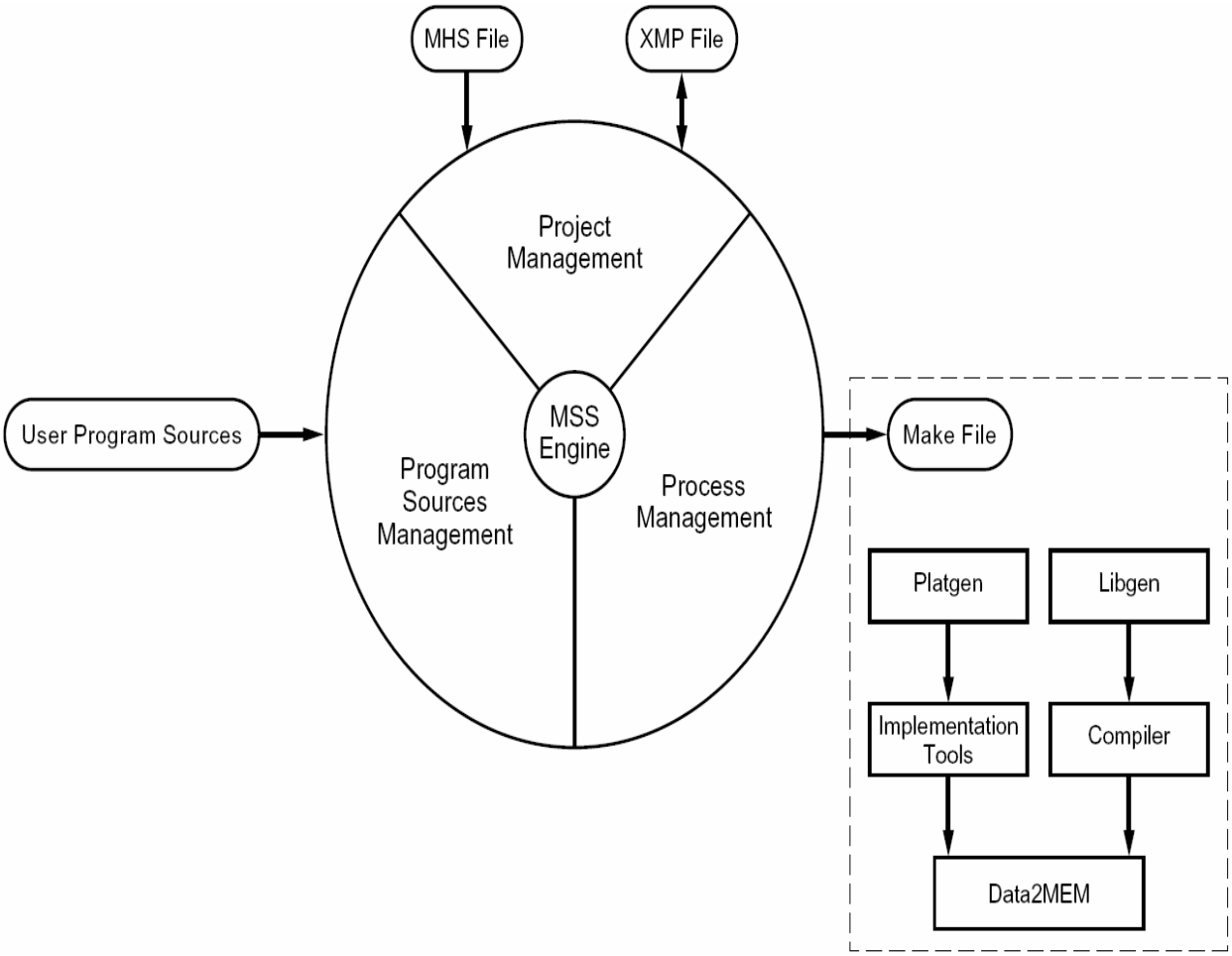
Στην περιγραφή που θα ακολουθήσει θα αναφερθούν τα κυριότερα εργαλεία του EDK που χρησιμοποιήθηκαν για την τέλεση των εφαρμογών και τα οποία ανήκουν στην αρχιτεκτονική της σχεδίασης των ενσωματωμένων συστημάτων.

4.2 Xilinx Platform Studio (XPS)

4.2.1 Εισαγωγή

Το XPS είναι ένα ολοκληρωμένο περιβάλλον για την προδιαγραφή των ροών του λογισμικού και του υλικού ενός ενσωματωμένου συστήματος με επεξεργαστή. Παρέχει έναν επεξεργαστή κειμένου και μία διεπαφή διαχείρισης του εκάστοτε project, καθώς και επεξεργασία των συνδέσεων του συστήματος μέσω περιβάλλοντος γραφικών. Το XPS υποστηρίζει την δημιουργία των απαραίτητων αρχείων MHS και MSS και βοηθάει στην επεξεργασία τους μέσω παράθυρων διαλόγου και γραφικού περιβάλλοντος. Υποστηρίζει την προσαρμογή των βιβλιοθηκών λογισμικού, των οδηγών, των χειριστών διακοπών και της μεταγλώττισης των προγραμμάτων του χρήστη. Επίσης μέσω του XPS μπορεί να γίνει η διαχείριση των πηγαίων αρχείων κώδικα, σε C, της εφαρμογής του χρήστη και η επιλογή του μοντέλου προσομοίωσης για όλο το σύστημα.

Κάθε νέο project μπορεί να ξεκινήσει είτε τροφοδοτώντας το XPS με ένα ήδη υπάρχων αρχείο MHS, είτε ξεκινώντας με ένα κενό αρχείο MHS και προσθέτοντας στοιχεία σε αυτό. Το XPS επιτηρεί τη διαχείριση και την αλληλεξάρτηση μεταξύ του υλικού, του λογισμικού και της προσομοίωσης, καλώντας τα σωστά εργαλεία την κατάλληλη στιγμή όπως φαίνεται στο σχήμα 4.6.



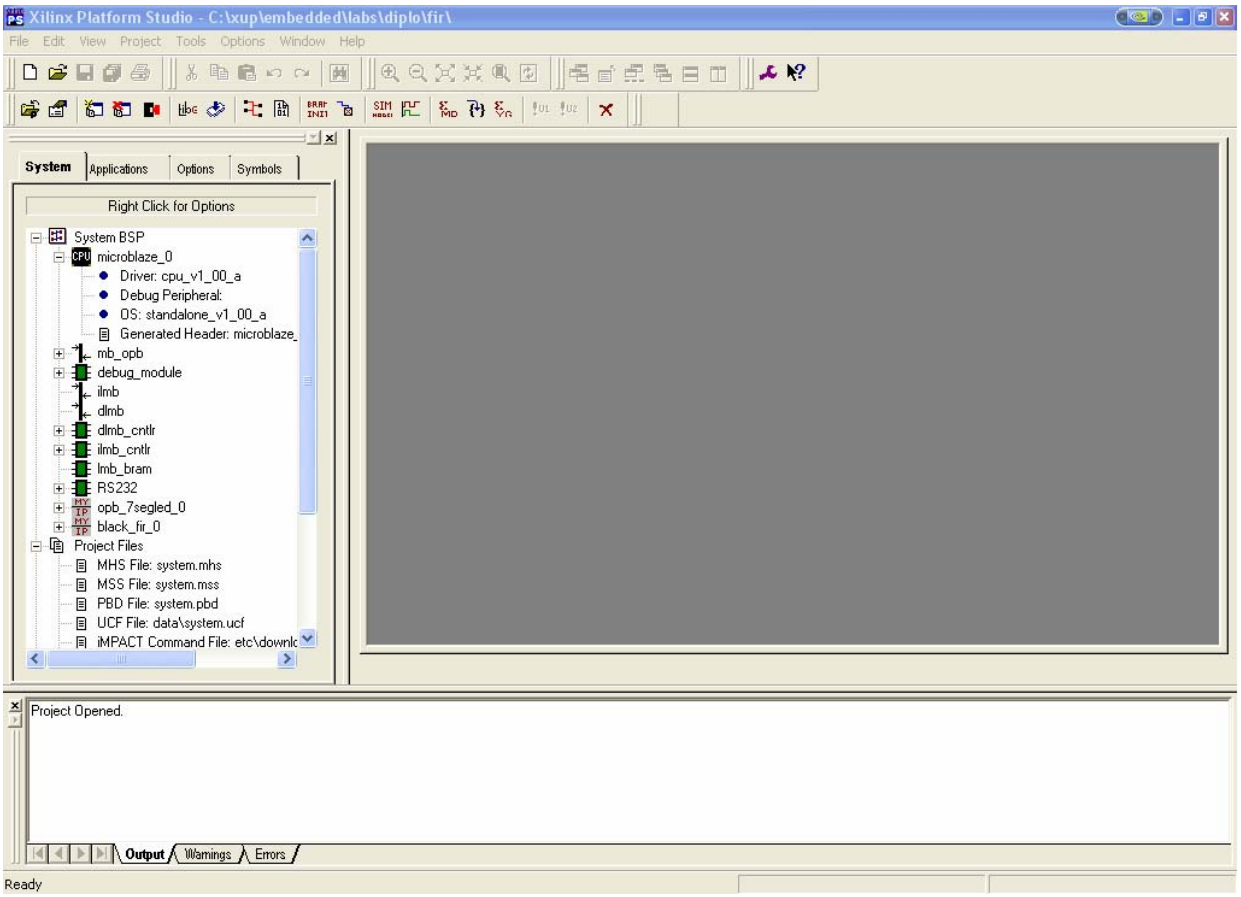
Σχήμα 4.6 – Οι διαδικασίες του XPS

4.2.2 Διαχείριση του Project

Ένα νέο project μπορεί να ξεκινήσει με δύο τρόπους, όσες και οι επιλογές του υπό-μενού New Project του μενού File. Επιλέγοντας Base System Builder, εμφανίζεται ένας βοηθός που παρέχει καθοδήγηση βήμα προς βήμα για την εκκίνηση του project, για περιορισμένο όμως αριθμό καρτών μεταξύ των οποίων είναι και η κάρτα Spartan-3 Starter Kit, ενώ με την επιλογή Platform Studio το νέο project πρέπει να συσταθεί από τον χρήστη μέσω του XPS ή να φορτωθεί από ένα ήδη υπάρχων αρχείο MHS. Ένα αρχείο XMP, το οποίο αναφέρεται και στο σχήμα 4.6, κρατάει πληροφορίες του project για λογαριασμό του XPS, όπως τις θέσεις των αρχείων MHS και MSS και την θέση των πηγαίων αρχείων κώδικα μιας εφαρμογής. Εάν η υλοποίηση πρόκειται να γίνει σε κάρτα διαφορετική από αυτές που υποστηρίζει ο βοηθός υπάρχει και η δυνατότητα να αλλάξει είτε η συσκευή, είτε το πακέτο επιλέγοντας Project Options στο μενού Options.

4.2.3 Η διεπαφή του XPS

Στο σχήμα 4.7 φαίνεται ότι το XPS χωρίζεται σε τρία παράθυρα.



Σχήμα 4.7 – Το XPS

Ο επεξεργαστής, είτε κειμένου, είτε γραφικών, είναι το κυρίως παράθυρο και βρίσκεται στο δεξί μέρος του XPS. Οσοσδήποτε αριθμός αρχείων είναι δυνατόν να είναι ταυτόχρονα ανοιχτά, μεταξύ των οποίων αρχεία πηγαίου κώδικα, αρχεία επικεφαλίδων και αρχεία VHDL. Το αρχείο που επιτρέπει την γραφική επεξεργασία του συστήματος είναι το PBD (Platform Block Diagram). Το PBD αρχείο ανοίγει επιλέγοντας View Block Diagram, στο μενού Project του XPS.

Στο αριστερό μέρος του παραθύρου του XPS, υπάρχουν τέσσερις καρτέλες. Από αυτές η καρτέλα System, αναπαριστά το δημιουργημένο σύστημα με τη μορφή δέντρου. Η αναπαράσταση αυτή περιλαμβάνει τρία υπόδεντρα.

- Το υπόδεντρο System BSP, αποτελείται από τα μέρη που αποτελούν το σύστημα. Κάθε στοιχείο έχει ένα δικό του υπόδεντρο που περιέχει πληροφορίες για το εκάστοτε στοιχείο.
- Το υπόδεντρο Project Files, αποτελείται από τα κυριότερα αρχεία, μεταξύ των οποίων τα MHS, MSS, PBD, UCF, που αντιστοιχούν στο project. Κάνοντας διπλό-κλικ επάνω σε αυτά εμφανίζονται στο κυρίως παράθυρο του XPS.
- Το υπόδεντρο Project Options δείχνει τις τρέχουσες τιμές διαφόρων παραμέτρων του project. Κάνοντας διπλό-κλικ επάνω σε μία από τις παραμέτρους, εμφανίζεται το παράθυρο διαλόγου Project Options.

Μία άλλη από τις τέσσερις καρτέλες του XPS είναι η καρτέλα Applications, στην οποία βρίσκονται όλες οι εφαρμογές λογισμικού που περιλαμβάνονται στο project. Ο χρήστης μπορεί να δημιουργεί τις εφαρμογές λογισμικού που συνδέονται με κάποιον επεξεργαστή του συστήματος. Μια εφαρμογή λογισμικού διακρίνεται από το όνομά της, καθώς και από το σύνολο των αρχείων πηγαίου κώδικα και τα αρχεία επικεφαλίδων, τα οποία μπορούν να δημιουργηθούν από τον χρήστη. Σε κάθε εφαρμογή αντιστοιχεί ένα εκτελέσιμο αρχείο το οποίο

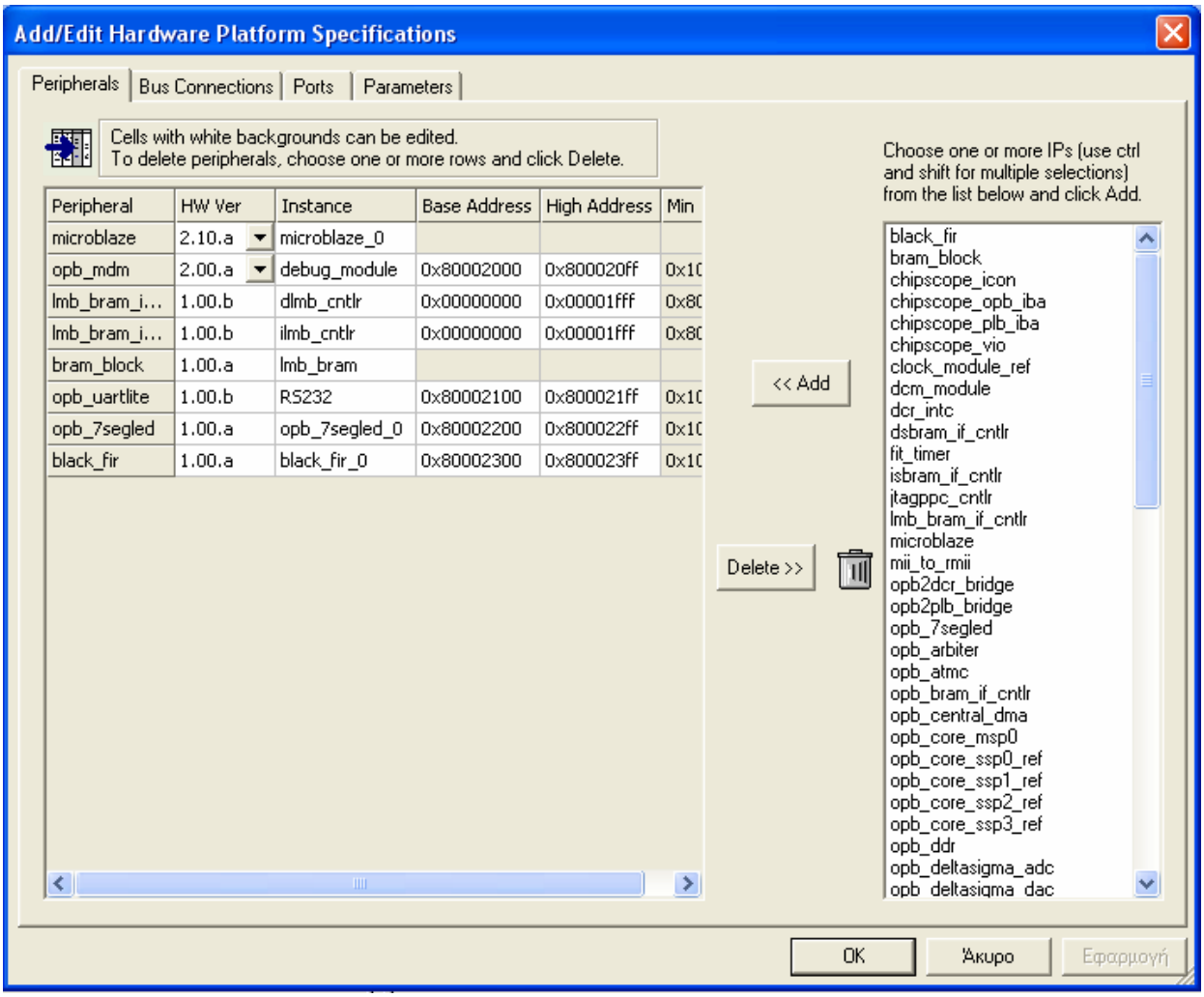
μπορεί να κατέβει στο FPGA. Υπάρχει η επιλογή, εάν σε κάποιο συγκεκριμένο σύστημα δεν είναι απαραίτητες όλες οι εφαρμογές, κάποιες εφαρμογές να σημειωθούν ως ανενεργές, κάνοντας δεξί-κλικ επάνω τους και επιλέγοντας Make Project Inactive. Με τον τρόπο αυτό διατηρούνται οι εφαρμογές, αλλά αγνοούνται από το σύστημα. Πέρα από την παραπάνω επιλογή που αφορά την ενεργοποίηση ή όχι του project, υπάρχει μια σειρά από επιλογές όπως η ρύθμιση των παραμέτρων του compiler, το άνοιγμα αρχείων σχετικών με την εφαρμογή και συσχέτιση της κάθε εφαρμογής με τον αντίστοιχο επεξεργαστή. Άλλη μία επιλογή είναι η αρχικοποίηση της μνήμης BRAM με τον εκτελέσιμο κώδικα της εφαρμογής (Mark to Initialize BRAMs). Ενεργοποιώντας αυτή την επιλογή το XPS κατά το κατέβασμα στο FPGA ενημερώνει το bitstream με το αρχείο ELF που αντιστοιχεί στην εφαρμογή.

Το τρίτο από τα παράθυρα του XPS είναι αυτό που βρίσκεται στο κάτω μέρος, και το οποίο χρησιμεύει για την εμφάνιση των ενεργειών του XPS και μηνυμάτων που αφορούν λάθη ή ειδοποιήσεις του XPS ή των εργαλείων που αυτό καλεί.

4.2.4 Διαχείριση των πλατφόρμων υλικού, λογισμικού και προσομοίωσης

Προκειμένου ο χρήστης να τροποποιεί της πλατφόρμες υλικού, λογισμικού, και προσομοίωσης, το XPS διαθέτει μια σειρά από λειτουργίες που αλλάζουν τα χαρακτηριστικά και τις παραμέτρους του συστήματος.

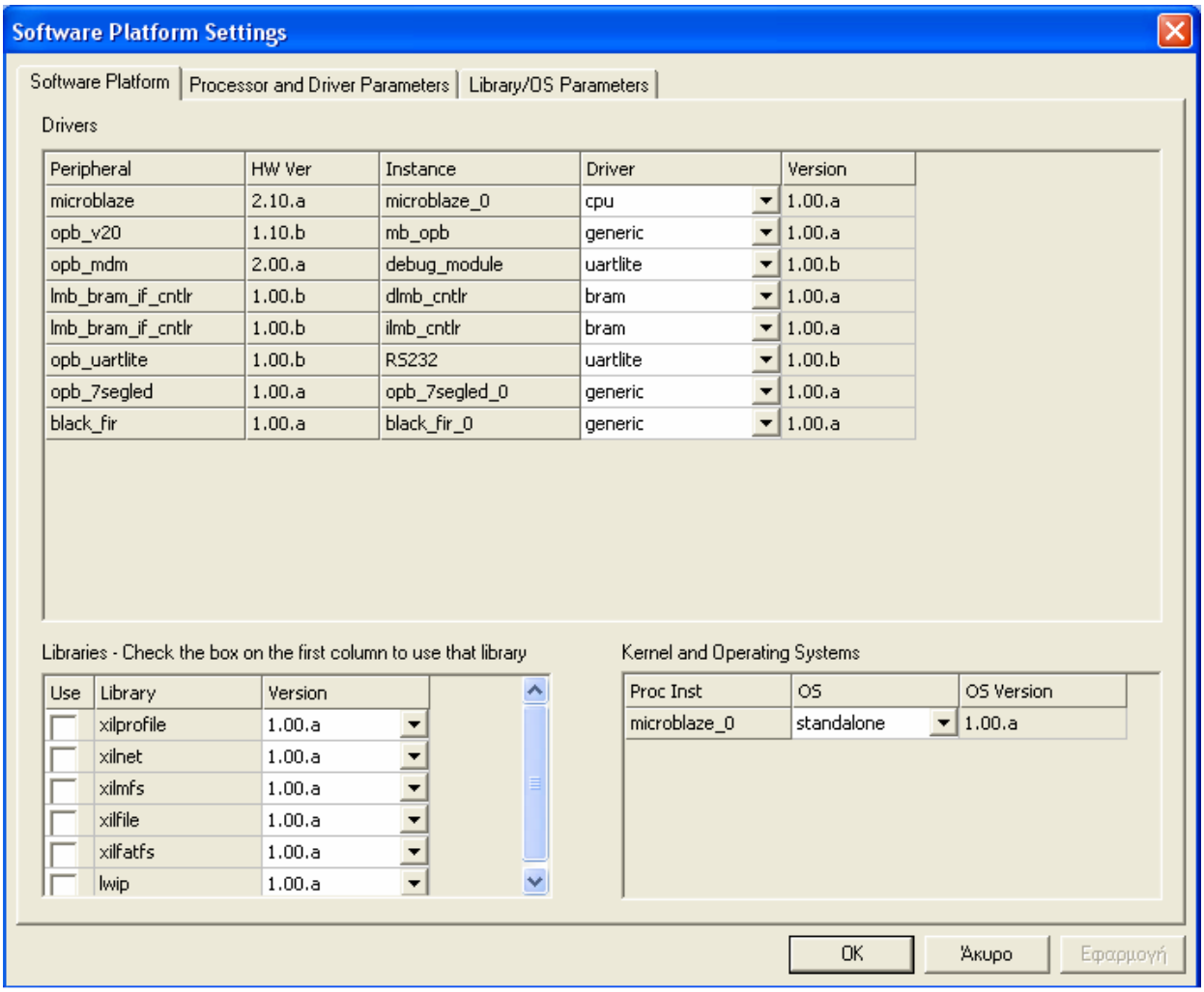
Στην καρτέλα System του παραθύρου με την δενδρική μορφή του XPS, κάνοντας δεξί-κλικ στο υπόδεντρο System BSP, εμφανίζεται η επιλογή Add/Edit Cores…(dialog). Επιλέγοντας εμφανίζεται ένα παράθυρο διαλόγου, στο οποίο είναι δυνατόν να προστεθούν και να αφαιρεθούν cores (IPs) από το σύστημα, να ορισθούν οι συνδέσεις των επιλεχθέντων cores στους αντίστοιχους διαδρόμους δεδομένων, να ορισθούν οι θύρες των cores και τα χαρακτηριστικά τους, και τέλος να ορισθούν οι παράμετροι λειτουργίας για τα ίδια τα cores. Κάθε μία από τις παραπάνω ενέργειες αντιστοιχεί στις τέσσερις καρτέλες που αποτελούν του παράθυρο διαλόγου. Οι αλλαγές που γίνονται σε αυτό επηρεάζουν το αρχείο MPD, Microprocessor Peripheral Description, του κάθε περιφερειακού, και το αρχείο MHS. Εκτός από την παραπάνω λειτουργία, κάνοντας δεξί-κλικ στο System BSP, εμφανίζεται και η επιλογή Simulation Models, όπου καθορίζεται ο τύπος του επιθυμητού μοντέλου προσομοίωσης, μεταξύ του Behavioral, του Structural, και του Timing.



Σχήμα 4.8 – Το παράθυρο διαλόγου Add/Edit Cores

Στο υπόδεντρο System BSP όπως έχει αναφερθεί βρίσκονται τα περιφερειακά που αποτελούν το σύστημα. Κάνοντας δεξί-κλικ πάνω σε κάθε ένα από αυτά υπάρχει η δυνατότητα ανοίγματος δύο αρχείων χαρακτηριστικών για τα περιφερειακά. Τα αρχεία αυτά είναι το MPD και το MDD, Microprocessor Driver Definition. Το MPD αρχείο περιέχει πληροφορίες όπως τις θύρες του περιφερειακού, τις συνδέσεις του με τους διαδρόμους δεδομένων και τις παραμέτρους λειτουργίας του, ενώ το MDD αναφέρεται στον οδηγό (driver), στον οποίο αναθέτεται η λειτουργία του εκάστοτε περιφερειακού.

Οι λειτουργίες που αφορούν αποκλειστικά την πλατφόρμα του λογισμικού συγκεντρώνονται στο παράθυρο διαλόγου Software Platform Settings. Το παράθυρο αυτό εμφανίζεται κάνοντας διπλό-κλικ σε ένα οποιοδήποτε στοιχείο του υπόδεντρου System BSP. Τρεις καρτέλες αποτελούν το παράθυρο, Software Platform, Processor and Driver Parameters και Library/OS Parameters.



Σχήμα 4.9 – Το παράθυρο διαλόγου Software Platform Settings

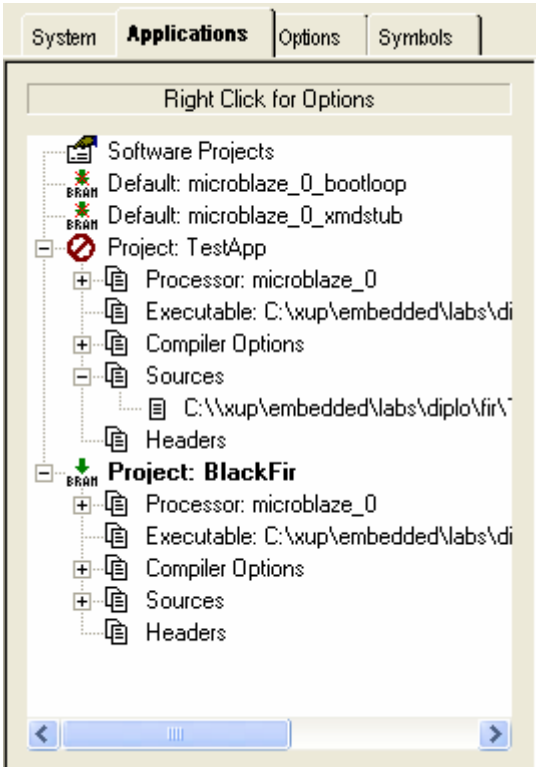
- Καρτέλα Software Platform. Η πρώτη κατά σειρά καρτέλα διαιρείται σε τρία τμήματα, Drivers, Libraries και Kernel and Operating Systems, όπου ορίζονται οι οδηγοί για τα περιφερειακά και τους επεξεργαστές, οι δυνατές βιβλιοθήκες που προσφέρει το EDK και μπορούν να συμπεριληφθούν στο σύστημα και το λειτουργικό σύστημα που χρησιμοποιείται αντίστοιχα. Η Xilinx διαθέτει ένα kernel για λειτουργικό σύστημα, αλλά αυτό δεν αποτελεί την προεπιλογή.

- Καρτέλα Processor and Driver Parameters. Η καρτέλα αυτή χωρίζεται σε δύο πίνακες. Στον έναν, Processor Parameters, ορίζονται ιδιότητες που αφορούν την σχεδίαση και την λειτουργία του επεξεργαστή (compilers, διεπαφή εκσφαλμάτωσης, συχνότητα λειτουργίας) και στον άλλον, Driver Parameters, ορίζονται παράμετροι που αφορούν τους οδηγούς των συσκευών. Εάν κάποιο περιφερειακό συνδέεται με θύρα διακοπής του επεξεργαστή, στον πίνακα Driver Parameters είναι δυνατόν να οριστεί το όνομα της ρουτίνας διακοπής που θα κληθεί σε περίπτωση διακοπής.

- Καρτέλα Library/OS Parameters. Στην τελευταία καρτέλα του παραθύρου διαλόγου Software Platform Settings, υπάρχει μια λίστα παραμέτρων του λειτουργικού συστήματος και των εκάστοτε βιβλιοθηκών που έχουν επιλεγεί για το σύστημα.

4.2.5 Διαχείριση των εφαρμογών λογισμικού

Όπως αναφέρθηκε και παραπάνω, το αντίστοιχο του αρχείου MHS, που περιγράφει το υλικό, είναι το αρχείο MSS, που περιγράφει το λογισμικό. Το αρχείο αυτό περιέχει εγγραφές για το λειτουργικό σύστημα, για τους οδηγούς των cores που αποτελούν το σύστημα και για τις βιβλιοθήκες. Το άλλο κομμάτι που αφορά το λογισμικό του συστήματος έχει να κάνει με τις εφαρμογές. Οι διαχείριση των εφαρμογών γίνεται από την καρτέλα Applications, που βρίσκεται στο δεξί παράθυρο του XPS. Στην καρτέλα αυτή η, οποία έχει επίσης δενδρική μορφή, μία νέα εφαρμογή είναι δυνατόν να προστεθεί στο σύστημα κάνοντας δεξί-κλικ στο Software Projects και επιλέγοντας Add New Project. Μετά την προσθήκη δημιουργείται ένα υπόδεντρο με την ονομασία της νέας εφαρμογής, και στο οποίο ανήκουν όλες οι παράμετροι και οι λειτουργίες που αφορούν την εφαρμογή.



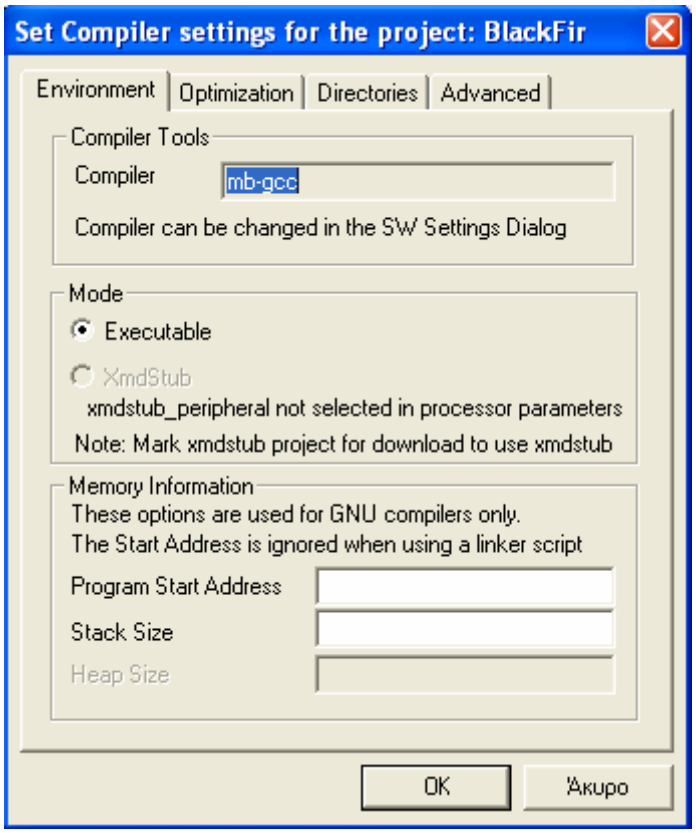
Σχήμα 4.10 – Τα υπόδεντρα των εφαρμογών λογισμικού

Αρχεία πηγαίου κώδικα ή επικεφαλίδων, μπορούν να προστεθούν σε κάθε ενεργή εφαρμογή κάνοντας δεξί-κλικ στα στοιχεία του δέντρου Sources ή Headers αντίστοιχα και επιλέγοντας Add File. Με την ενέργεια αυτή ο φάκελος στον οποίο ανήκουν τα προστεθέντα αρχεία συμπεριλαμβάνεται στο μονοπάτι αναζήτησης επικεφαλίδων του compiler. Μετά την προσθήκη του υπάρχει η δυνατότητα για κάθε αρχείο να εξαιρεθεί από την εφαρμογή. Κάνοντας δεξί-κλικ επάνω του και επιλέγοντας Delete File το αρχείο μπορεί να μην σβήνεται, από την μνήμη του υπολογιστή, αλλά πλέον δεν λαμβάνει μέρος στην μεταγλώττιση και στην δημιουργία του εκτελέσιμου αρχείου της εφαρμογής. Άλλες δύο λειτουργίες που αφορούν την επεξεργασία της εφαρμογής και έχουν ήδη αναφερθεί είναι η επεξεργασία των αρχείων, που με διπλό-κλικ επάνω τους ανοίγουν στο κυρίως παράθυρο του XPS και η ενημέρωση του bitstream του FPGA με τον εκτελέσιμο κώδικα της εφαρμογής, ώστε αυτός να βρίσκεται στην BRAM του συστήματος. Για να γίνει αυτό πρέπει η επιλογή Mark to Initialize BRAMs να είναι ενεργή ώστε το XPS μέσω του εργαλείου data2mem να αρχικοποιήσει το bitstream. Κατά αντίστοιχο τρόπο αρχικοποιούνται και οι μνήμες των μοντέλων προσομοίωσης, όταν έχει γίνει η παραπάνω επιλογή. Ο εκτελέσιμος κώδικας που ενημερώνει ή όχι την μνήμη του συστήματος είναι

δυνατόν να έχει δημιουργηθεί και σε ένα άλλο περιβάλλον εκτός του XPS, το οποίο προτιμά ο χρήστης. Με τον τρόπο αυτό το XPS δεν χειρίζεται καθόλου την εφαρμογή ούτε είναι δυνατόν να προστεθούν νέα αρχεία C σε αυτήν, απλώς κοινοποιείται στο XPS η θέση του έτοιμου αρχείου ELF.

Το XPS σε κάθε επεξεργαστή αντιστοιχεί δύο προεπιλεγμένες εφαρμογές που δεν μπορούν να επεξεργασθούν από τον χρήστη. Η μόνη επιλογή έγκειται στο αν ο χρήστης θα επιλέξει ή όχι να ενημερώσει την μνήμη με τις εφαρμογές αυτές. Οι δύο αυτές εφαρμογές βρίσκονται στο επάνω μέρος της καρτέλας Applications. Η πρώτη, bootloop, είναι μία εφαρμογή, μιας εντολής, η οποία διακλαδώνεται πίσω στον εαυτό της, δηλαδή στην ουσία αποτελεί έναν ατέρμονο βρόγχο. Η χρησιμότητα της εφαρμογής αυτής είναι το ότι αποτρέπει τον επεξεργαστή από το να εκτελεί αυθαίρετες εντολές, ενώ έχει ξεκινήσει την λειτουργία του και ακόμα δεν έχουν φορτωθεί οι πραγματικές εφαρμογές από κάποια εξωτερική μνήμη. Η δεύτερη εφαρμογή, xmdstub, χρειάζεται όταν υπάρχει περιφερειακό που να εξυπηρετεί την εκσφαλμάτωση και εφαρμογή τύπου XMDSTUB.

Σε κάθε εφαρμογή είναι δυνατό να αντιστοιχούν διαφορετικές επιλογές όσον αφορά τον compiler. Η επεξεργασία των επιλογών αυτών γίνεται μέσω του παράθυρου διαλόγου Set Compiler settings… που αναδύεται κάνοντας διπλό-κλικ επάνω στο όνομα της εκάστοτε εφαρμογής. Το παράθυρο διαλόγου αποτελείται από τέσσερις καρτέλες.



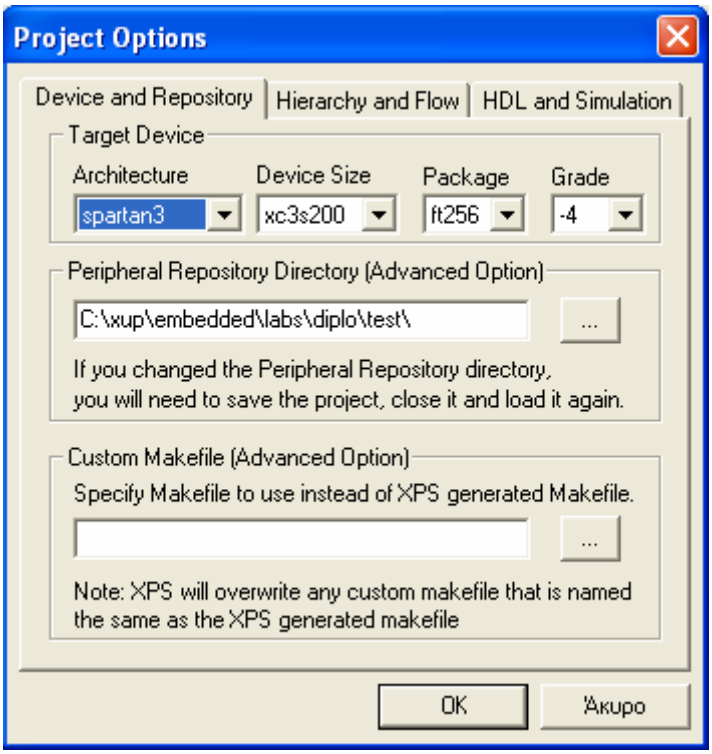
Σχήμα 4.11 – Το παράθυρο διαλόγου για την επεξεργασία των επιλογών του compiler

- Καρτέλα Environment. Στην καρτέλα αυτή φαίνεται ποιος compiler χρησιμοποιείται κάτι που όπως αναφέρθηκε μπορεί να αλλάξει από το παράθυρο διαλόγου Software Platform Settings και καθορίζει το αν η εφαρμογή είναι τύπου EXECUTABLE ή XMDSTUB, κάτι που ισχύει μόνο για εφαρμογές που αφορούν τον επεξεργαστή MicroBlaze. Επίσης στην ίδια καρτέλα δίνεται η δυνατότητα να οριστεί η διεύθυνση εκκίνησης του προγράμματος, το μέγεθος της στοίβας και το μέγεθος του σωρού (σωρό διαθέτει μόνο ο επεξεργαστής PowerPC).

- Καρτέλα Optimization. Εδώ ορίζεται ο βαθμός βελτιστοποίησης σε ένα από τα τρία επίπεδα που υποστηρίζεται, καθώς και άλλες επιλογές βελτιστοποίησης και εκσφαλμάτωσης.
- Καρτέλα Directories. Οι φάκελοι στους οποίους βρίσκονται αρχεία χρήσιμα για τον compiler, γνωστοποιούνται σε αυτήν εδώ την καρτέλα. Οι φάκελοι που ορίζονται αφορούν τον compiler, τις βιβλιοθήκες (Libraries), και τα αρχεία επικεφαλίδων (Include). Στο πεδίο Libs to Link ορίζονται οι βιβλιοθήκες των χρηστών που είναι απαραίτητες από τον linker. Το μονοπάτι που οδηγεί στο αρχείο Linker Script, που παίζει μεγάλο ρόλο στην δημιουργία του εκτελέσιμου αρχείου της εφαρμογής, πρέπει να δοθεί στο αντίστοιχο πεδίο, ενώ και η τοποθεσία στην οποία θα αποθηκευτεί το εκτελέσιμο αρχείο ELF ορίζεται στην καρτέλα αυτή.
- Καρτέλα Advanced. Στην καρτέλα αυτή δίνεται η δυνατότητα να περαστούν διάφορες οδηγίες στον Preprocessor, στον Assembler, και στον Linker, μέσω του compiler. Τέλος υπάρχει και το πεδίο Program Sources Compiler Options, όπου μπορούν να δοθούν οδηγίες στον compiler, που δεν ανήκουν σε καμία κατηγορία από τις υπόλοιπες που υπάρχουν στις καρτέλες.

4.2.6 Ιδιότητες του Project

Η επιλογή από το μενού Options του XPS, Options → Project Options, ανοίγει ένα παράθυρο διαλόγου, όπως αυτό του σχήματος 4.12, που επιτρέπει στον χρήστη να ορίσει διάφορες επιλογές του project. Αυτό το παράθυρο διαλόγου αποτελείται από τρεις καρτέλες.



Σχήμα 4.12 – Το παράθυρο διαλόγου Project Options

- Καρτέλα Device and Repository. Στην καρτέλα αυτή δίνεται η δυνατότητα να αλλάξει η κάρτα στην οποία στοχεύει το project. Οι άλλες δύο επιλογές που δίνει η καρτέλα αυτή είναι ο ορισμός ενός φακέλου όπου ενδεχομένως υπάρχουν οδηγοί, βιβλιοθήκες, και περιφερειακά που μπορούν να προστεθούν στο σύστημα και η χρησιμοποίηση ενός αρχείου make ορισμένου από τον χρήστη, αντί αυτού που χρησιμοποιεί το EDK. Σε ένα αρχείο make

περιγράφεται ο τρόπος με τον οποίο δημιουργείται το σύστημα και καθορίζεται η σωστή σειρά κλήσης των κατάλληλων εργαλείων για τον σκοπό αυτό.

- Καρτέλα Hierarchy and Flow. Εδώ ορίζεται η ιεραρχία του συστήματος, αν το τρέχον σύστημα είναι το top-level ή αν αποτελεί κομμάτι αυτού και το αν η υλοποίηση θα γίνει από το XPS ή από το ISE. Υλοποίηση στο ISE είναι επιβεβλημένη εάν το τρέχον σύστημα δεν είναι το top-level.
- Καρτέλα HDL and Simulation. Στην τελευταία καρτέλα ορίζονται παράμετροι που έχουν να κάνουν με την προσομοίωση του συστήματος, όπως η γλώσσα και το εργαλείο προσομοίωσης.

4.2.7 Κλήση των εργαλείων σύνθεσης και υλοποίησης

Τα εργαλεία του EDK, που καλούνται μέσω του XPS, χρησιμεύουν στην πραγματοποίηση δύο ροών, την ροή λογισμικού και την ροή υλικού. Η ροή λογισμικού οδηγεί στον σχηματισμό του λογισμικού μέρους του συστήματος και περιλαμβάνει δύο σημαντικά βήματα.

1. Δημιουργία των βιβλιοθηκών. Επιλέγοντας από το μενού του XPS Tools → Generate Libraries and BSPs, καλείται το εργαλείο LibGen δεχόμενο ως είσοδο το αρχείο MSS, ώστε να δημιουργηθούν οι κατάλληλες βιβλιοθήκες που να υποστηρίζουν τον έλεγχο όλων των περιφερειακών στις εφαρμογές λογισμικού.
2. Μεταγλώττιση και η ένωση των αρχείων πηγαίου κώδικα μιας εφαρμογής σε ένα εκτελέσιμο αρχείο. Πατώντας το πλήκτρο  στο μενού του XPS, που αντιστοιχεί στην ενέργεια Compile Program Sources, ο compiler δημιουργεί τον εκτελέσιμο κώδικα της εφαρμογής. Το βήμα αυτό προϋποθέτει την εκτέλεση του βήματος 1, αν κάτι τέτοιο δεν έχει γίνει τότε αυτόματα με την κλήση του βήματος 2 ξεκινάει πρώτα το βήμα 1.

Αντίστοιχα με την ροή λογισμικού, η ροή υλικού είναι υπεύθυνη για την δημιουργία του υλικού μέρους του συστήματος. Και σε αυτήν την περίπτωση υπάρχουν δύο σημαντικά βήματα.

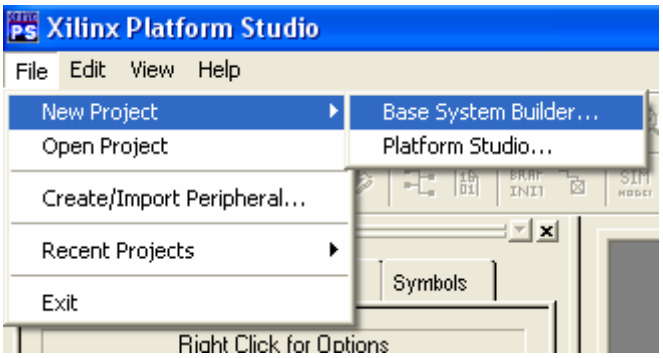
1. Δημιουργία των αρχείων του netlist του συστήματος. Επιλέγοντας Tools → Generate Netlist, καλείται το εργαλείο PlatGen δεχόμενο ως είσοδο το αρχείο MHS και παράγει τα αρχεία netlist σε τυποποίηση NGC.
2. Το βήμα της υλοποίησης. Επιλέγοντας Tools → Generate Bitstream, δημιουργείται το αρχείο bitstream του συστήματος. Αυτό όμως συμβαίνει μόνο στην περίπτωση που έχει επιλεχθεί υλοποίηση μέσα στο XPS. Στην περίπτωση που έχει επιλεχθεί υλοποίηση στο ISE, η παραπάνω επιλογή είναι αδύνατη καθώς εμφανίζεται αχνή. Τότε επιλέγοντας Tools → Export to ProjNav, τα κατάλληλα αρχεία του συστήματος προστίθενται σε ένα project στο ISE και αφού ολοκληρωθεί εκεί η υλοποίηση του bitstream, με την επιλογή Tools → Import from ProjNav, τα απαραίτητα αρχεία εισάγονται στο project του XPS.

Δύο άλλα εργαλεία του EDK είναι το bitinit και το iMPACT της Xilinx. Το πρώτο, είναι το εργαλείο που ενώνει τις δύο ροές, του υλικού και του λογισμικού. Επιλέγοντας Tools → Update Bitstream, ο εκτελέσιμος κώδικας των εφαρμογών ενσωματώνεται στο bitstream του υλικού. Εάν πριν από αυτήν την ενέργεια δεν έχουν κληθεί τα απαραίτητα εργαλεία για τον σχηματισμό των αρχείων του υλικού και του λογισμικού, τότε αυτά καλούνται αυτόματα. Το αποτέλεσμα που προκύπτει από την παραπάνω διαδικασία, είναι το αρχείο download.bit το οποίο προορίζεται για να διαμορφώσει το FPGA. Την διαδικασία αυτή αναλαμβάνει το iMPACT, που κατεβάζει το download.bit στο FPGA, επιλέγοντας Tools → Download.

4.3 Βοηθός δημιουργίας συστήματος, Base System Builder

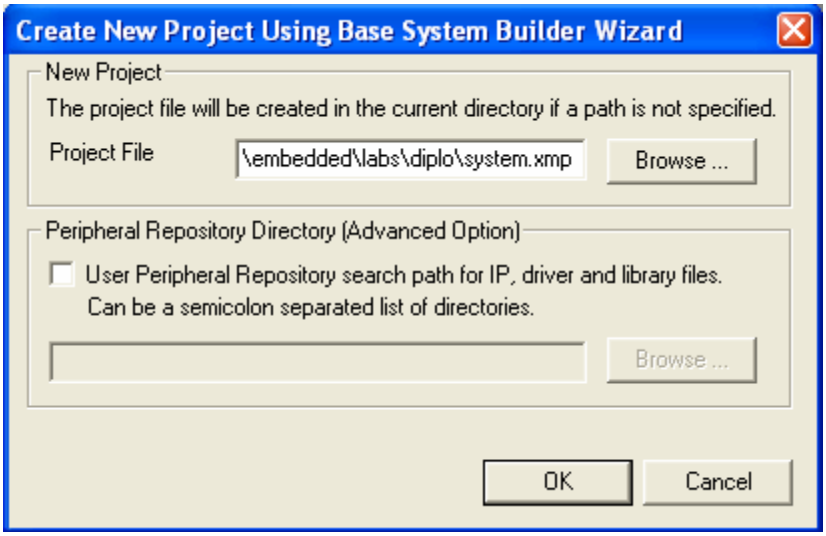
Ο βοηθός Base System Builder (BSB), οδηγεί στην εύκολη και γρήγορη δημιουργία του project του XPS ενός συστήματος, το οποίο απευθύνεται σε μία συγκεκριμένη αναπτυξιακή κάρτα. Βασιζόμενο στην επιλογή της κάρτα, ο βοηθός BSB προσφέρει μια σειρά από επιλογές σχετικές με αυτό, που οδηγούν στην δημιουργία ενός βασικού και στοιχειώδους συστήματος. Με το πέρας του βοηθού, έχει δημιουργηθεί ένα αρχείο MHS του συστήματος αυτού, το οποίο φορτώνεται σε ένα project του XPS. Από εκείνο το σημείο, είναι στην ευχέρεια του χρήστη να βελτιώσει περαιτέρω το σύστημα πριν την υλοποίηση.

Ο βοηθός ξεκινάει με την επιλογή File → New Project → Base System Builder.



Σχήμα 4.13 – Ξεκίνημα του βοηθού Base System Builder

Στο παράθυρο διαλόγου Create New Project που εμφανίζεται ζητείται να εισαχθεί η διαδρομή και ο φάκελος στον οποίο θα αποθηκευθεί το project. Επίσης μπορούν να ορισθούν φάκελοι από τον χρήστη που να περιέχουν περιφερειακά (IP cores), οδηγούς ή βιβλιοθήκες στο πεδίο Peripheral Repository Directory.



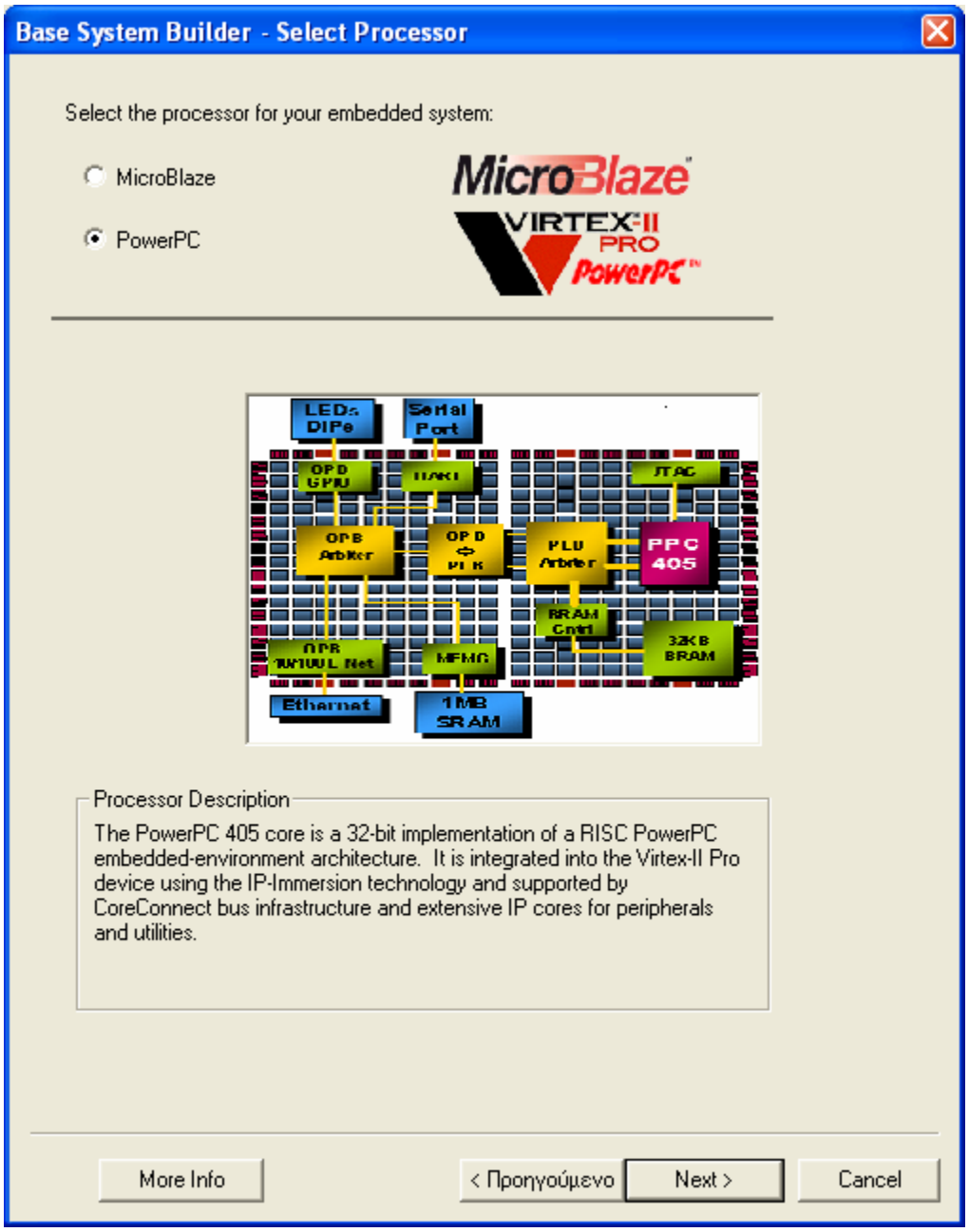
Σχήμα 4.14 – Διαχείριση των φακέλων του project

Μετά τον ορισμό του φακέλου του project, η πρώτη επιλογή του βοηθού είναι αυτή της αναπτυξιακής κάρτας. Ζητείται ο κατασκευαστής, το όνομα και η έκδοση της κάρτας, ενώ δίνεται από τον βοηθό και μια σύντομη περιγραφή της εκάστοτε επιλογής. Σε αυτό το βήμα του βοηθού δίνεται και η δυνατότητα φόρτωσης ενός αρχείου .bsb, το οποίο δημιουργείται όταν ολοκληρώνεται ο βοηθός και που ενδεχομένως ο χρήστης διαθέτει από παλαιότερη χρήση του

βοηθού. Με την ενέργεια αυτή όλες οι επιλογές που είχαν γίνει τότε φορτώνονται στον βοηθό αλλά ο χρήστης εξακολουθεί να έχει την δυνατότητα να τις αλλάξει όπως αυτός επιθυμεί.

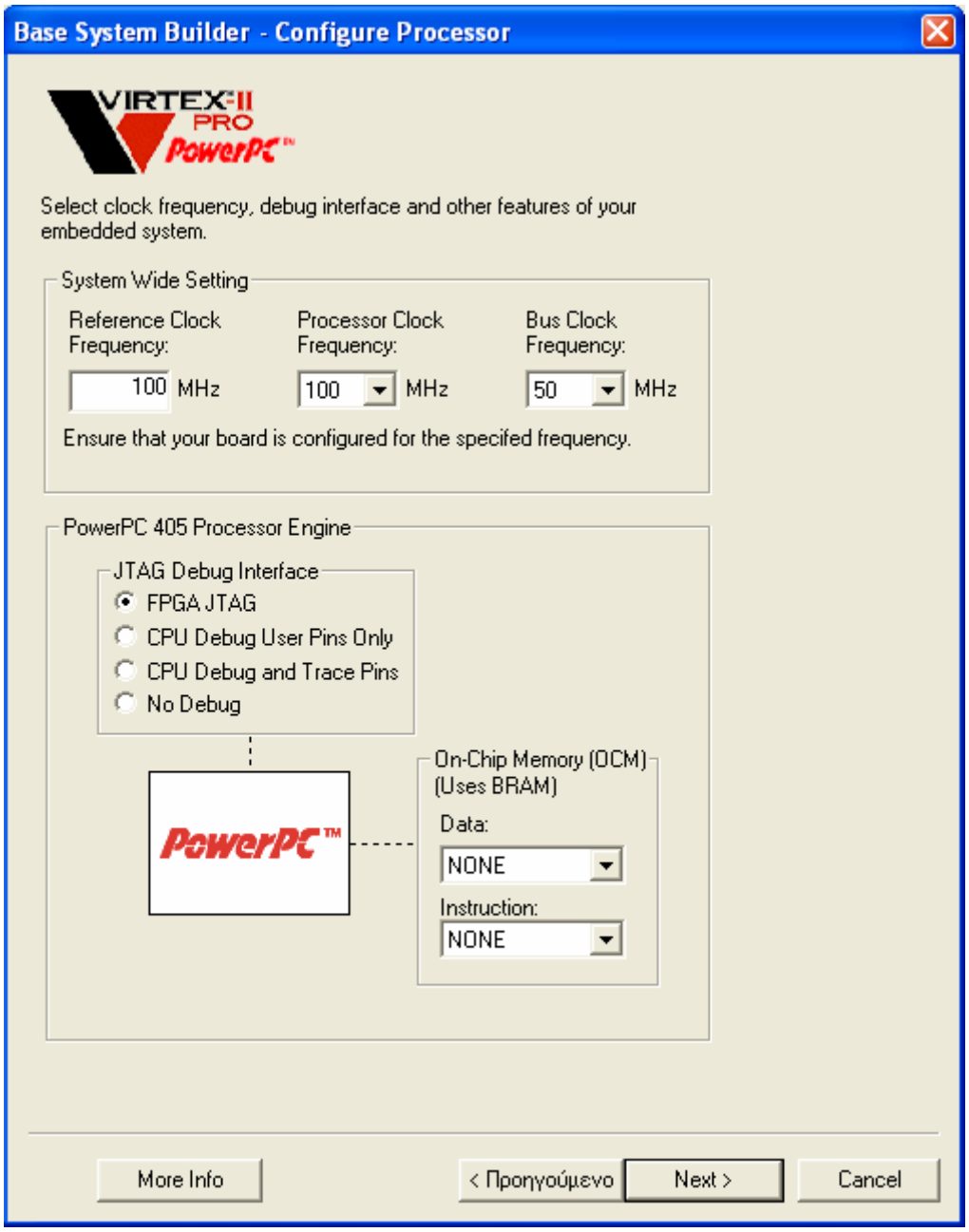
Σχήμα 4.15 – Επιλογή της αναπτυξιακής κάρτας

Επόμενο βήμα είναι η επιλογή του επεξεργαστή μεταξύ του MicroBlaze και του PowerPC. Η επιλογή του PowerPC δεν είναι πάντα δυνατή αφού δεν υποστηρίζεται από όλες τις κάρτες, όπως δεν υποστηρίζεται και από την κάρτα Spartan-3 Starter Kit που χρησιμοποιήθηκε στις εφαρμογές. Στο παράθυρο αυτό εμφανίζεται επίσης μία μικρή περιγραφή του επιλεγμένου επεξεργαστή και μία απεικόνιση ενός τυπικού συστήματος που χρησιμοποιεί αυτόν τον επεξεργαστή.



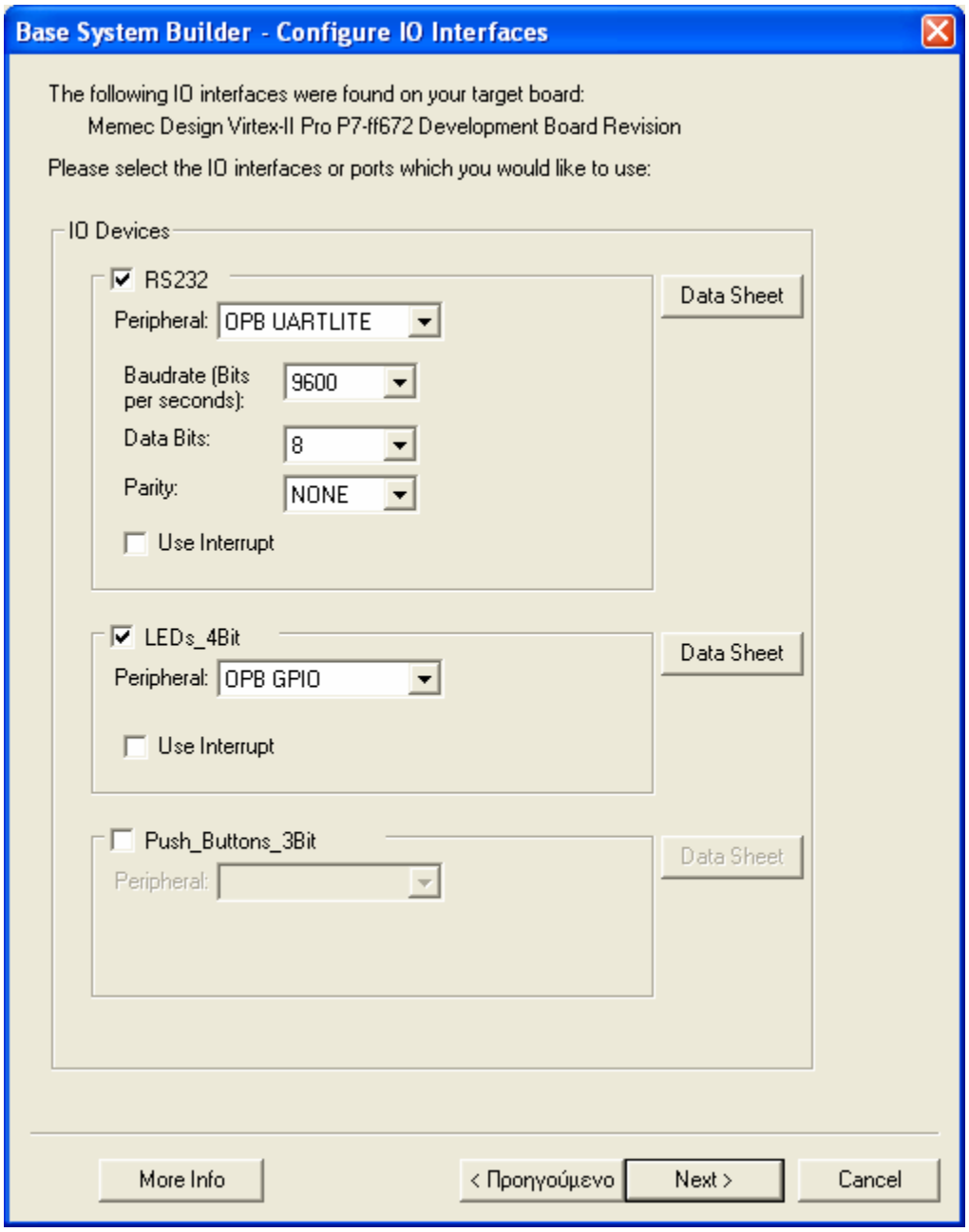
Σχήμα 4.16 – Επιλογή επεξεργαστή

Το επόμενο παράθυρο του βοηθού βασίζεται στην επιλογή του επεξεργαστή, καθώς αφορά επιλογές που αφορούν αποκλειστικά το σύστημα με τον συγκεκριμένο επεξεργαστή. Οι ρυθμίσεις που αφορούν το σύστημα έχουν να κάνουν με την συχνότητα λειτουργίας του επεξεργαστή, ενώ για τον PowerPC καθορίζεται και η συχνότητα του διαδρόμου δεδομένων. Οι ρυθμίσεις που αφορούν τον επεξεργαστή είναι για το αν θα υπάρχει ή όχι και ποιος θα είναι ο τύπος της διεπαφής εκσφαλμάτωσης και το μέγεθος της εσωτερικής μνήμης BRAM. Για τον MicroBlaze υπάρχει και επιλογή ενεργοποίησης / απενεργοποίησης της βοηθητικής μνήμης (cache).



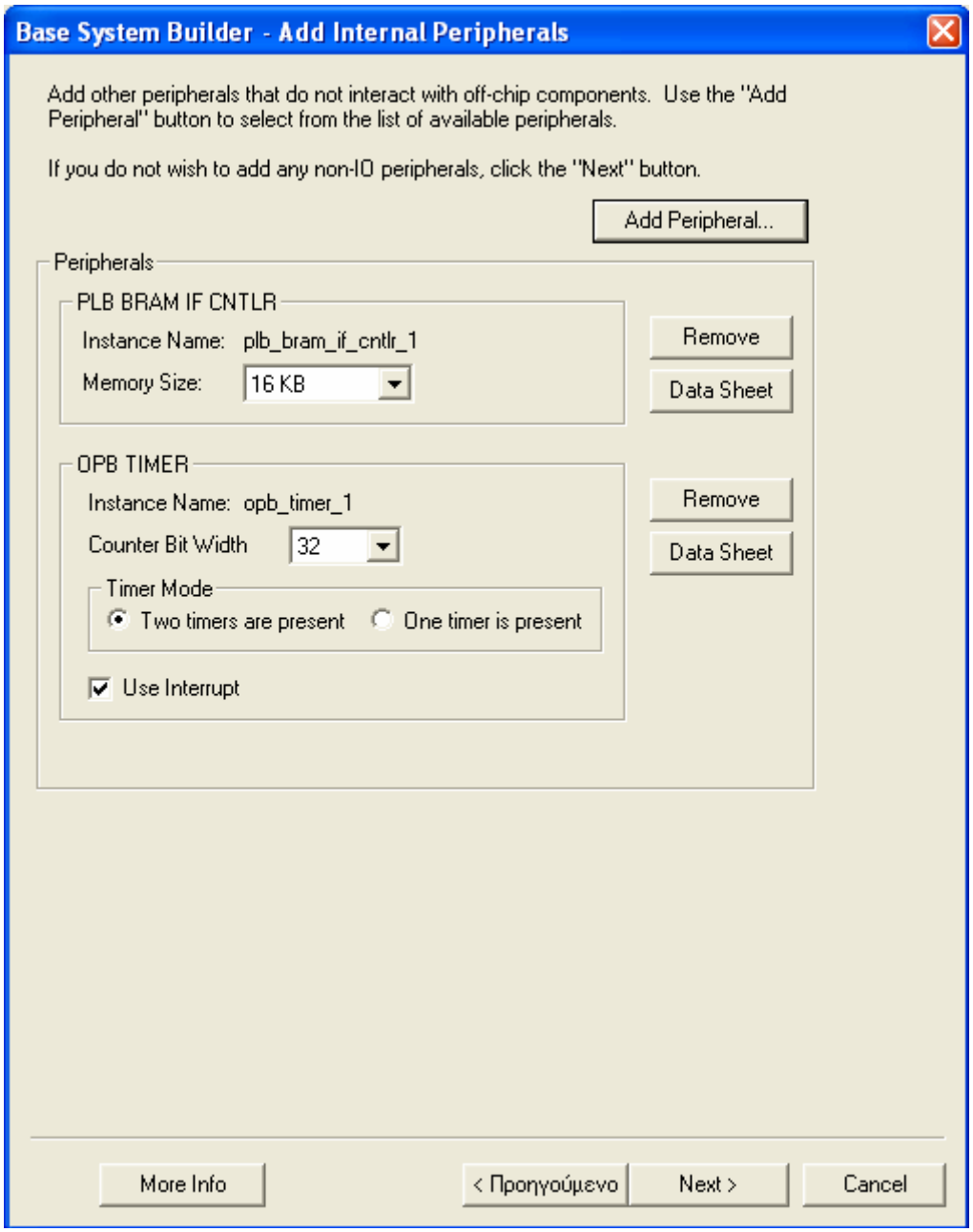
Σχήμα 4.17 – Ρυθμίσεις συστήματος και επεξεργαστή

Ένα μεγάλο κομμάτι του συστήματος συνήθως αποτελούν η εξωτερική μνήμη και οι συσκευές εισόδου / εξόδου. Για το λόγο αυτό ο βοηθός μετά την ολοκλήρωση της επιλογής του επεξεργαστή και του ορισμού των ρυθμίσεών του, ζητά από τον χρήστη να καθορίσει τις συσκευές που θα χρησιμοποιηθούν, τις οποίες βέβαια υποστηρίζει η εκάστοτε κάρτα, ενώ υπάρχει και η δυνατότητα επιλογής το επιθυμητού περιφερειακού που θα ελέγχει την συσκευή. Ο βοηθός BSB ζητά από τον χρήστη τις τιμές κάποιων βασικών παραμέτρων των συσκευών, αλλά χρησιμοποιεί προεπιλεγμένες τιμές για τις πιο προχωρημένες παραμέτρους. Ακόμα και έτσι ο χρήστης έχει την δυνατότητα, μετά την ολοκλήρωση του βοηθού, να επεξεργαστεί όλες τις παραμέτρους μέσω του αρχείου MHS. Για κάθε συσκευή που προστίθεται ο βοηθός δημιουργεί τις κατάλληλες θύρες στο σύστημα και τις συνδέει με τους σωστούς ακροδέκτες του FPGA σε ένα αρχείο UCF, User Constraints File.



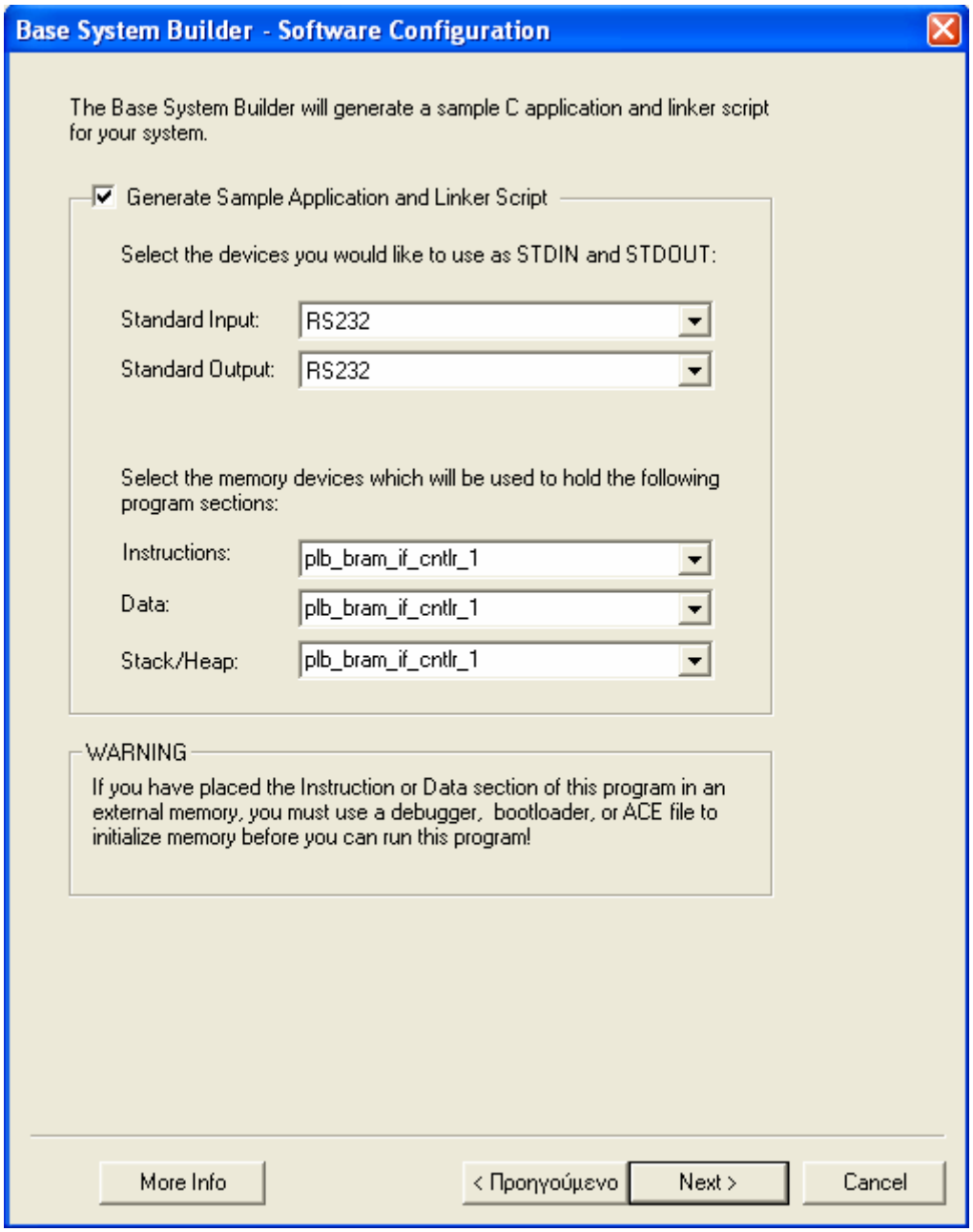
Σχήμα 4.18 – Προσθήκη συσκευών στο σύστημα

Εκτός από τις συσκευές που διαθέτει η αναπτυξιακή κάρτα και που αλληλεπιδρούν με συσκευές εκτός αυτής, υπάρχει και η δυνατότητα προσθήκης εσωτερικών περιφερειακών, όπως είναι οι ελεγκτές εσωτερικής μνήμης και οι χρονιστές. Η προσθήκη αυτών των περιφερειακών γίνεται στο παράθυρο διαλόγου που φαίνεται στο σχήμα 4.19. Επιλέγοντας Add Peripheral, αναδύεται μία λίστα με τα υποστηριζόμενα περιφερειακά από την οποία ο χρήστης μπορεί να επιλέξει αυτό της αρεσκείας του. Όπως είναι φυσικό, μιας και πρόκειται για εσωτερικά περιφερειακά, ο βοηθός δεν πραγματοποιεί καμία εγγραφή στο αρχείο UCF, για τα περιφερειακά αυτά.



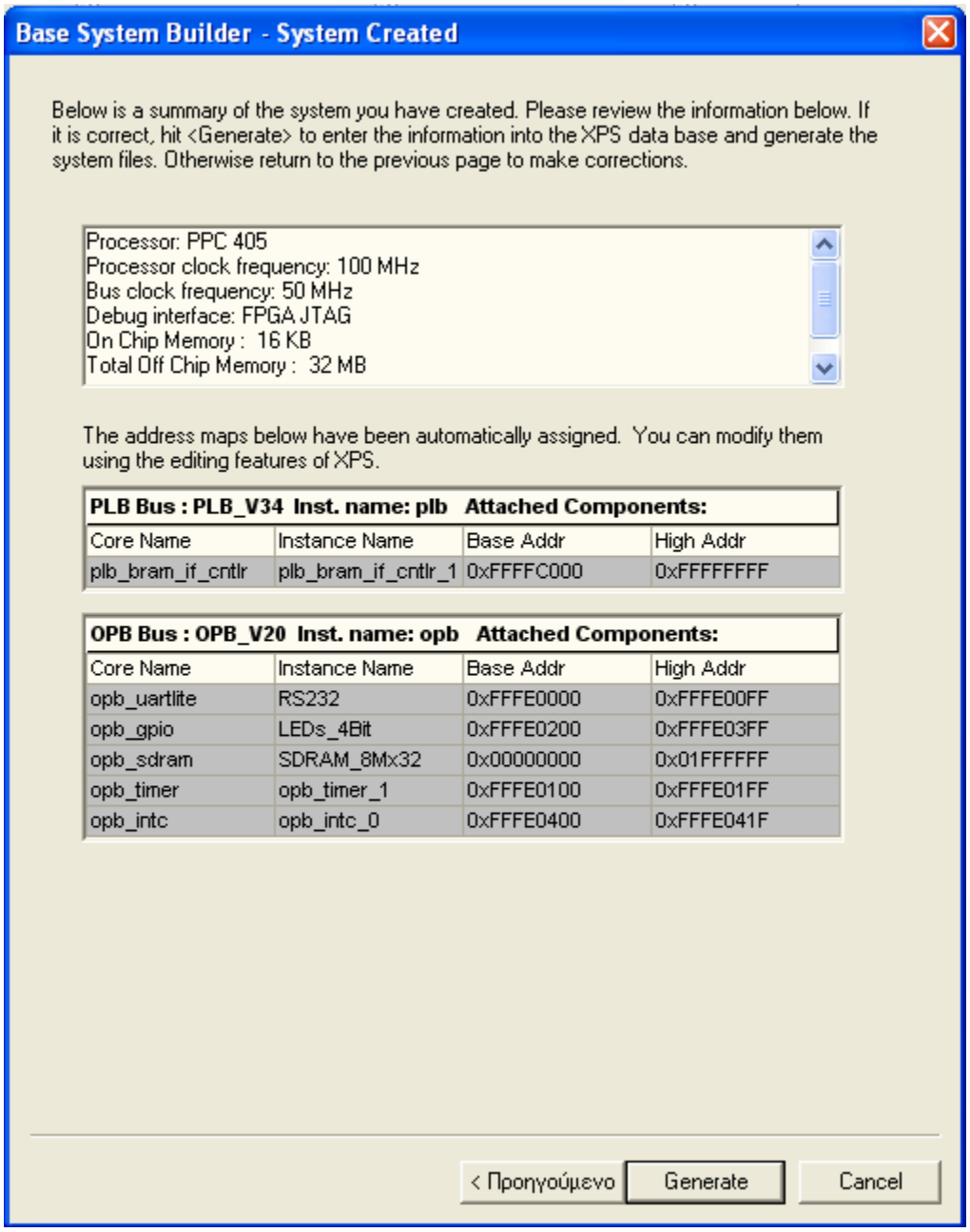
Σχήμα 4.19 – Προσθήκη εσωτερικών περιφερειακών

Μετά την ολοκλήρωση των βημάτων που αφορούν το υλικό, σειρά παίρνουν οι ρυθμίσεις που αφορούν το λογισμικό. Ο βοηθός BSB δημιουργεί, εκτός εάν ο χρήστης δεν το επιθυμεί, μία εφαρμογή λογισμικού TestApp καθώς και ένα linker script που σχετίζεται με το δημιουργούμενο σύστημα.. Σκοπός της εφαρμογής αυτής είναι η επιβεβαίωση της λειτουργικότητας του συστήματος, αλλά και ο παραδειγματισμός όσον αφορά την διαδικασία συγγραφής μιας πραγματικής εφαρμογής. Εάν στο παράθυρο διαλόγου του σχήματος 4.20 έχει επιλεγεί περιφερειακό εξόδου, τότε στην εφαρμογή συμπεριλαμβάνεται και μία εντολή εκτύπωσης στην επιλεγμένη συσκευή. Στο ίδιο παράθυρο καθορίζεται σε ποια συσκευή μνήμης θα τοποθετηθούν τα διάφορα μέρη του προγράμματος, όπως τα δεδομένα, οι εντολές και η στοίβα. Αν επιλεγεί η εσωτερική μνήμη τότε υπάρχει η δυνατότητα απευθείας φόρτωσης της εφαρμογής στο bitstream του συστήματος, ενώ αν επιλεχθεί η εξωτερική μνήμη τότε θα πρέπει να υπάρχει κάποιο εργαλείο που να κατεβάσει τον εκτελέσιμο κώδικα της εφαρμογής στην συσκευή της εξωτερικής μνήμης.



Σχήμα 4.20 – Δημιουργία δοκιμαστικής εφαρμογής

Στο επόμενο βήμα του βοηθού BSB παρουσιάζεται μία περίληψη του συστήματος που έχει σχηματιστεί, λίγο προτού δημιουργηθούν τα αρχεία του project. Σε αυτήν φαίνονται τα IP cores που θα χρησιμοποιηθούν για την υλοποίηση των συσκευών και των περιφερειακών, ενώ υπάρχει και ο χάρτης διευθύνσεων όλου του συστήματος. Σε αυτό το παράθυρο διαλόγου πατώντας Back, ο χρήστης μπορεί να επιστρέψει σε κάποιο προηγούμενο παράθυρο ώστε να διαφοροποιήσει κάποιες επιλογές που έχει κάνει ή μπορεί να πατήσει Generate ώστε να δημιουργηθούν τα αρχεία του συστήματος που έχει περιγράψει.

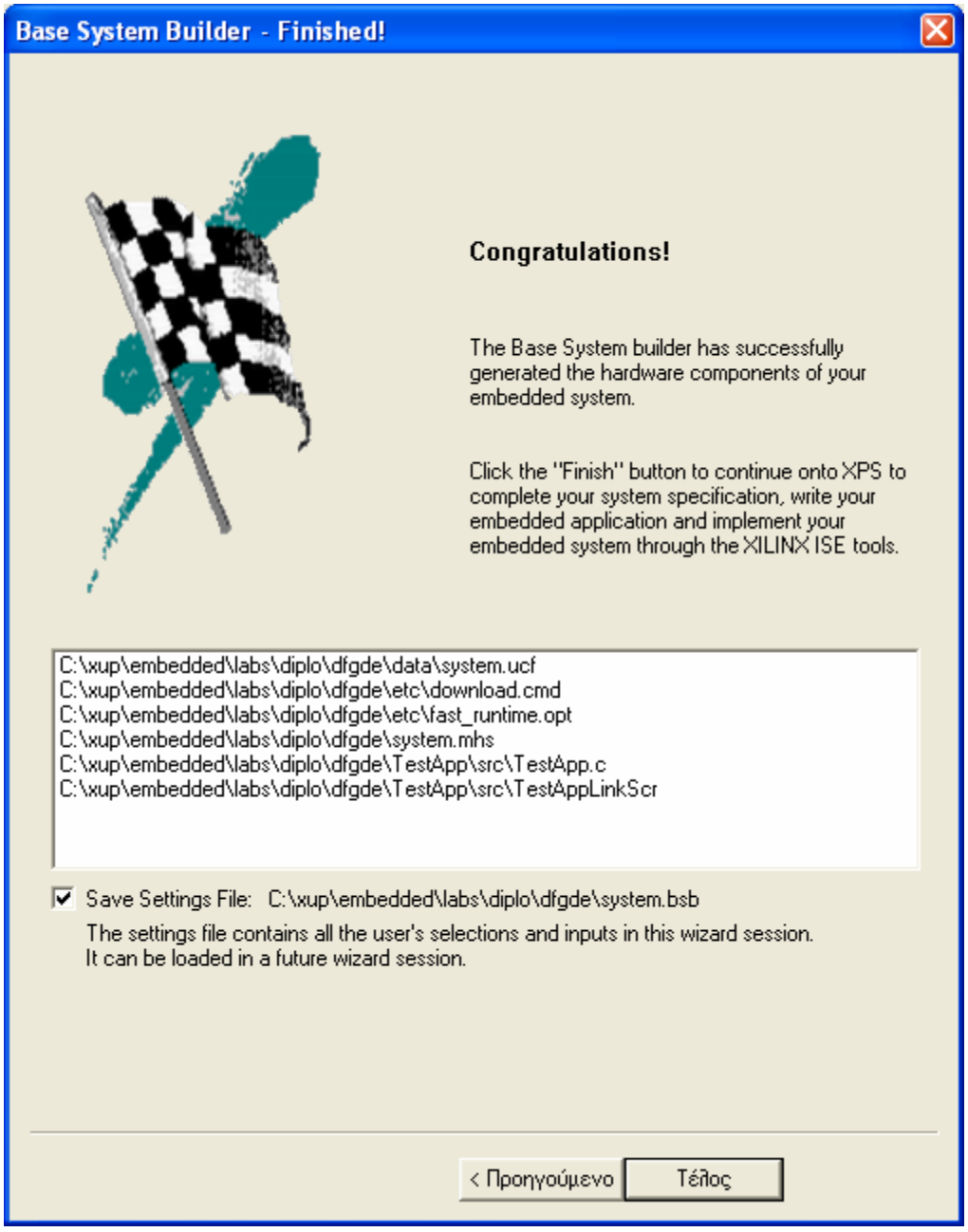


Σχήμα 4.21 – Περίληψη συστήματος

Ο βοηθός BSB δημιουργεί τέσσερα αρχεία που είναι απαραίτητα για κάθε project και τρία αρχεία που εξαρτάται από τον χρήστη ως προς το αν θα δημιουργηθούν ή όχι. Η λίστα με τα αρχεία αυτά εμφανίζεται στο τελευταίο βήμα του βοηθού και είναι τα παρακάτω:

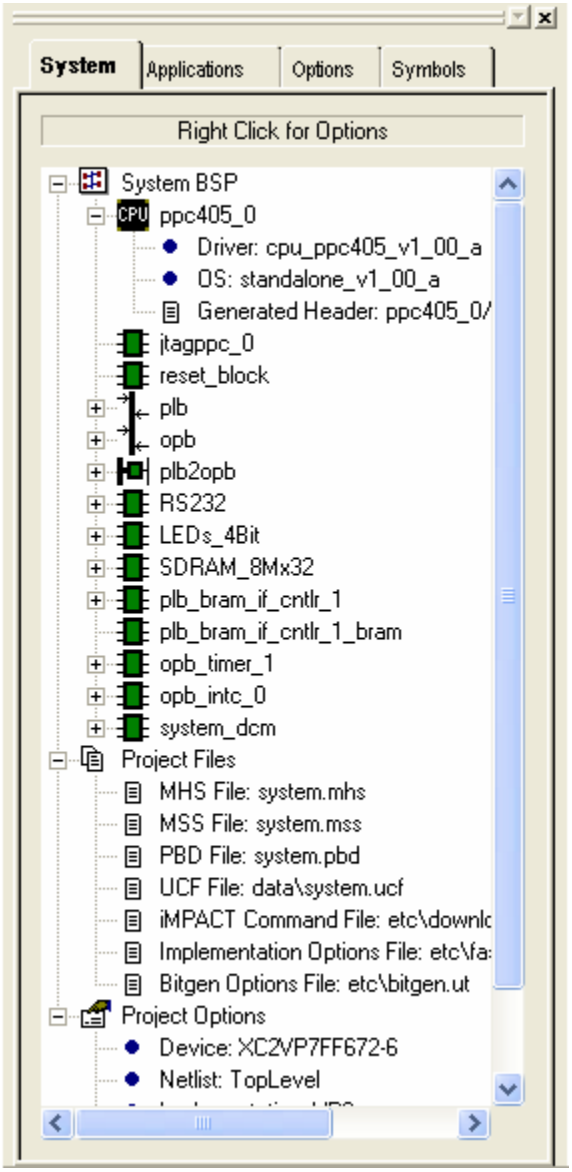
- Το αρχείο MHS, system.mhs, που περιέχει την περιγραφή του υλικού
- Το αρχείο UCF, system.ucf, που περιέχει τις αντιστοιχίσεις των ακροδεκτών του FPGA με τις θύρες του συστήματος, παραμέτρους χρονισμού και πρότυπα εισόδου / εξόδου
- Το αρχείο fast_runtime.opt, που περιέχει χρήσιμες πληροφορίες για τα εργαλεία υλοποίησης του EDK
- Το αρχείο download.cmd, που χρησιμοποιείται από το εργαλείο iMPACT για το κατέβασμα του συστήματος στην αναπτυξιακή κάρτα
- Το αρχείο TestApp.c, που είναι το αρχείο πηγαίου κώδικα της εφαρμογής TestApp
- Το αρχείο TestAppLnkScr, που είναι το linker script που χρησιμοποιεί η εφαρμογή TestApp και καθορίζει την θέση στη μνήμη κάθε τμήματος κώδικα της εφαρμογής

- Το αρχείο system.bsb, που αποθηκεύει όλες τις επιλογές που έγιναν από τον χρήστη κατά την χρήση του βοηθού



Σχήμα 4.22 – Ολοκλήρωση του βοηθού και παρουσίαση των αρχείων που δημιουργήθηκαν

Με την έξοδο από τον βοηθό BSB, το XPS ανοίγει το project που έχει δημιουργηθεί, ενώ δημιουργεί και δύο ακόμα αρχεία, ένα αρχείο XMP που αποθηκεύει στοιχεία που αφορούν το project και το αρχείο MSS που περιέχει τις απαραίτητες παραμέτρους λογισμικού που αφορούν το σύστημα.



Σχήμα 4.23 – Τα στοιχεία του project που δημιουργήθηκε με την βοήθεια του βοηθού BSB

Η φύση του βοηθού BSB είναι τέτοια ώστε να επιτρέπει την δημιουργία απλών και βασικών συστημάτων, χωρίς προχωρημένες αρχιτεκτονικές και ειδικές διαμορφώσεις, όπως χρήση άνω του ενός επεξεργαστή ή επεξεργασία του χάρτη διευθύνσεων. Παρά το γεγονός αυτό μετά την ολοκλήρωση του βοηθού υπάρχει η δυνατότητα επέκτασης και βελτίωσης του συστήματος μέσω του XPS. Επομένως το σύστημα που παράγει ο βοηθός BSB μπορεί να θεωρηθεί σαν την αφετηρία για την κατασκευή ενός πιο πολύπλοκου συστήματος.

4.4 Βοηθός δημιουργίας / εισαγωγής περιφερειακού

4.4.1 Εισαγωγή

Το EDK διαθέτει μία μεγάλη συλλογή από περιφερειακά (IP cores), τα οποία ο χρήστης μπορεί να χρησιμοποιήσει. Παρά τον αριθμό των προσφερόμενων περιφερειακών υπάρχει και η δυνατότητα δημιουργίας νέων περιφερειακών από τον χρήστη. Την ενέργεια αυτή αναλαμβάνει ο βοηθός δημιουργίας / εισαγωγής περιφερειακού, Create Import Peripheral Wizard. Όταν χρησιμοποιείται για την δημιουργία νέων περιφερειακών, ο βοηθός οδηγεί τον χρήστη στο να

δημιουργήσει τα απαραίτητα αρχεία ώστε να υλοποιήσει το περιφερειακό, χωρίς να χρειάζεται πλήρης και λεπτομερής γνώση, των πρωτοκόλλων των διαδρόμων δεδομένων, των συμβάσεων της ονοματολογίας και της διαμόρφωσης των διαφόρων διεπαφών που χρησιμοποιεί το EDK. Όταν ο βοηθός χρησιμοποιείται για την εισαγωγή διαθέσιμων περιφερειακών, τότε βοηθάει στο να δημιουργηθούν τα αρχεία των διεπαφών και να δοθεί η κατάλληλη δομή στους φακέλους που περιέχουν τα αρχεία του περιφερειακού, ώστε να είναι ορατά από τα διάφορα εργαλεία του EDK.

Για τις ανάγκες των εφαρμογών που σχεδιάστηκαν και υλοποιήθηκαν, ο βοηθός χρησιμοποιήθηκε με σκοπό την δημιουργία νέων περιφερειακών, όπως το περιφερειακό που ελέγχει την κάρτα DIO2 ή το ψηφιακό φίλτρο. Η διαδικασία της δημιουργίας ενός περιφερειακού μέσω του βοηθού αποτελείται από διαδοχικά παράθυρα διαλόγου τα οποία καθοδηγούν τον χρήστη. Στην καλύτερη περίπτωση το μόνο που απαιτείται από τον χρήστη είναι ο συγγραφή του σώματος ενός αρχείου HDL, στο οποίο περιγράφεται η λειτουργία του περιφερειακού. Ακόμα όμως και για την συγγραφή του κομματιού αυτού υπάρχουν κατευθύνσεις που δεν καθιστούν αναγκαία την πλήρη γνώση των σημάτων και της δομής που χρησιμοποιείται. Στην παρούσα έκδοση του EDK (6.2) υπάρχουν δύο περιορισμοί όσον αφορά την δημιουργία περιφερειακών από τον χρήστη:

- Για την περιγραφή του περιφερειακού υποστηρίζεται μόνο η γλώσσα VHDL, καθώς όλες οι βιβλιοθήκες είναι υλοποιημένες στη γλώσσα αυτή
- Τα περιφερειακά που υλοποιούνται συμπεριφέρονται μόνο σαν εξαρτημένα (slave mode) από τον διάδρομο δεδομένων OPB και μόνο αν συνδέονται στον LMB μπορούν να συμπεριφερθούν και σαν κύριες συσκευές (master-slave mode)

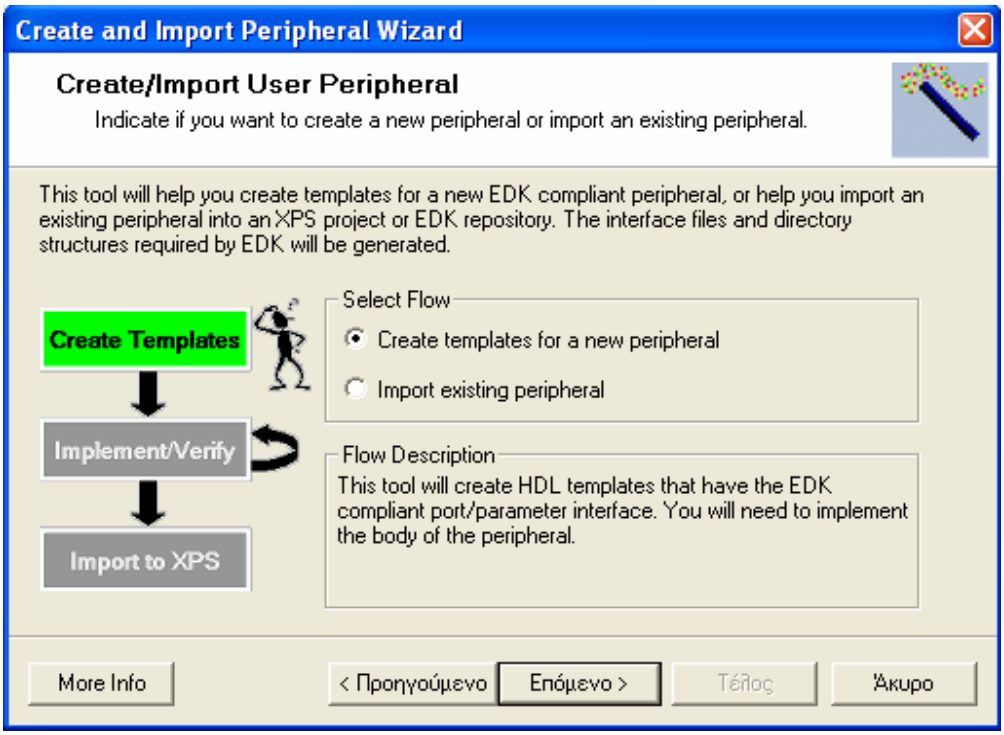
Κάθε περιφερειακό που είναι συμμορφώσιμο με τις συμβάσεις του EDK περιέχει τα τρία παρακάτω μέρη (components):

- Μία διεπαφή με τον διάδρομο στον οποίο συνδέεται. Η σύνδεση πραγματοποιείται μέσω των θυρών που διαθέτει το περιφερειακό, για να επικοινωνεί με το υπόλοιπο σύστημα
- Την διεπαφή IPIF, IP Interface. Μέσω της IPIF συνδέεται το περιφερειακό με το προηγούμενο component. Εκτελεί όλες τις βασικές λειτουργίες που χρειάζεται για την διαχείριση ενός περιφερειακού, όπως αποκωδικοποίηση διευθύνσεων, διαχείριση καταχωρητών, διαχείριση διακοπών, υποστήριξη DMA. Από όλες τις λειτουργίες που υποστηρίζει υλοποιούνται μονάχα όσες απαιτούνται από το περιφερειακό του χρήστη.
- Το τελευταίο component αποτελεί την ειδική λογική που χρησιμοποιεί το περιφερειακό για τις εφαρμογές του, και η οποία δεν καλύπτεται από την IPIF. Το κομμάτι αυτό αποκαλείται user-logic.

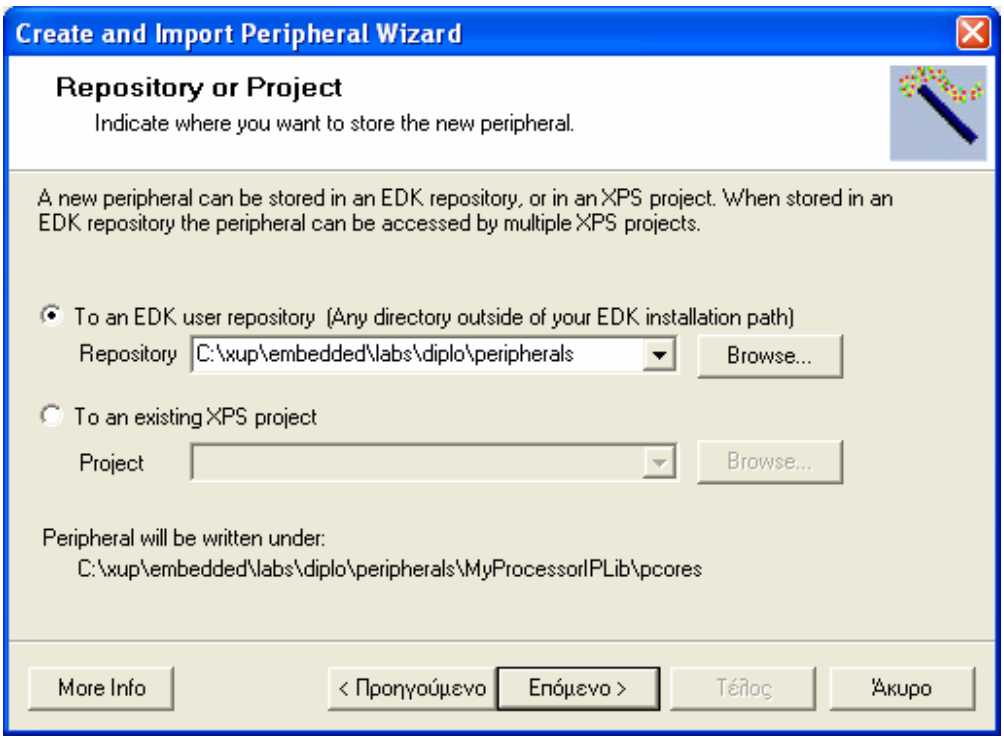
Η ένωση του user-logic με την IPIF γίνεται με την βοήθεια μιας σειράς από θύρες, που ονομάζονται IPIC, IP Interconnect, και απλουστεύουν την υλοποίηση του user-logic.

4.4.2 Εκτέλεση του Βοηθού

Ο βοηθός ξεκινά επιλέγοντας Tools → Import Peripheral Wizard. Επιλέγοντας να δημιουργηθεί το περίγραμμα για ένα νέο περιφερειακό, Create templates for a new peripheral, ο χρήστης καλείται να αποφασίσει εάν θα αποθηκεύσει τα αρχεία, και την ειδική δομή φακέλου που ακολουθούν, μέσα στους φακέλους του τρέχοντος project ή σε κάποια τοποθεσία ειδική για όλα τα περιφερειακά του χρήστη.

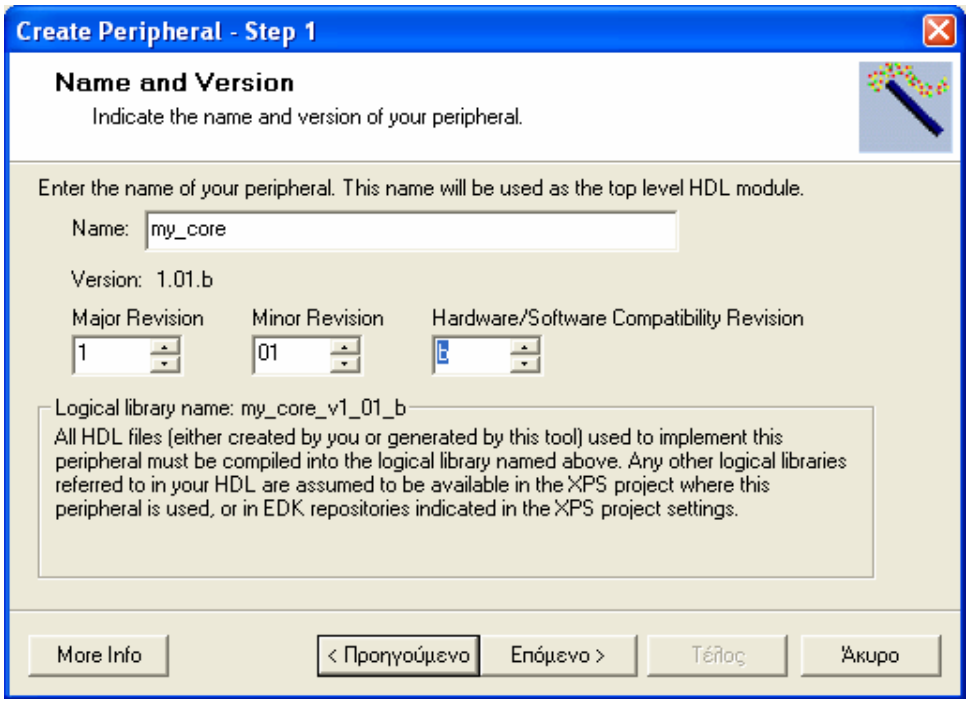


Σχήμα 4.24 – Επιλογή για δημιουργία νέου περιφερειακού



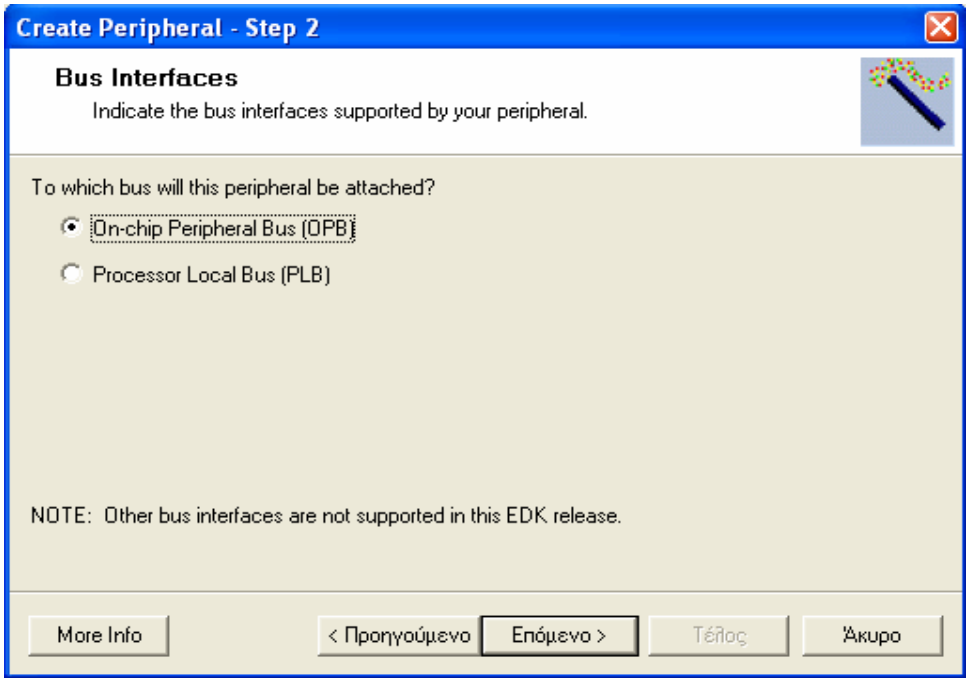
Σχήμα 4.25 – Ορισμός χώρου αποθήκευσης του περιφερειακού

Μετά την απόφαση για τον τόπο στον οποίο θα αποθηκευτούν τα αρχεία του νέου περιφερειακού, ο χρήστης πρέπει να δώσει την ονομασία του περιφερειακού που είναι ταυτόχρονα και αυτή του top-level HDL αρχείου στην ιεραρχία. Επίσης μπορούν να δοθούν αναγνωριστικά, όπως αριθμός κύριας και δευτερεύουσας αναθεώρησης και αναγνωριστικό συμβατότητας υλικού / λογισμικού.



Σχήμα 4.26 – Ορισμός της ονομασίας του περιφερειακού

Επόμενο βήμα του βοηθού είναι η επιλογή του διαδρόμου δεδομένων στον οποίο θα συνδεθεί το περιφερειακό, προκειμένου να υλοποιηθεί η κατάλληλη διεπαφή. Οι πιθανές επιλογές είναι ο LMB, για γρήγορα περιφερειακά, και ο OPB, για πιο απλά αλλά και πιο αργά περιφερειακά.



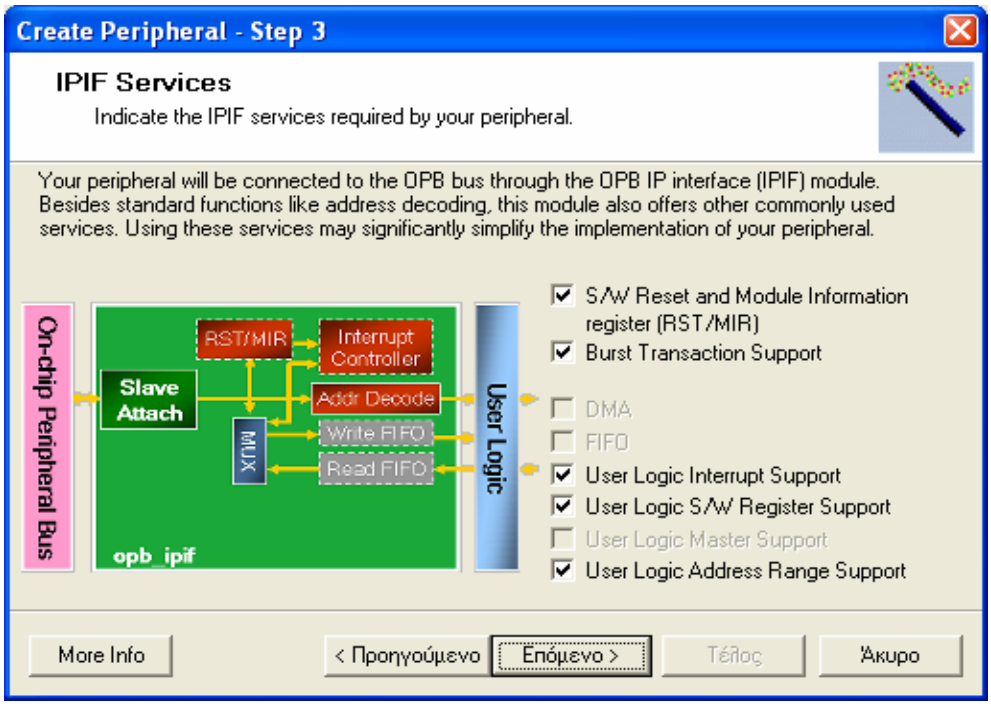
Σχήμα 4.27 – Επιλογή διαδρόμου δεδομένων

Όπως έχει ήδη αναφερθεί ένα από τα σημαντικότερα components κάθε νέου περιφερειακού, είναι η διεπαφή IPIF. Υλοποιείται σε δύο τύπους, έναν για τον LMB και έναν για τον OPB, ενώ η μία πλευρά της συνδέεται με την διεπαφή ενός από αυτούς τους δύο διαδρόμους δεδομένων και η άλλη με την διεπαφή IPIC, την οποία υλοποιεί το περιφερειακό του χρήστη. Η IPIC δεν

διαφοροποιείται ανάλογα με τον διάδρομο δεδομένων που χρησιμοποιείται και είναι ειδικά σχεδιασμένη για να συνεργάζεται με ευκολία με το περιφερειακό. Η διεπαφή IPIF προσφέρει μία σειρά από βασικές, και άλλες πιο σύνθετες, λειτουργίες οι οποίες είναι στην διάθεση του χρήστη για να τις επιλέξει. Μια περιγραφή των λειτουργιών που προσφέρονται από την διεπαφή IPIF δίνεται στον πίνακα 4.1.

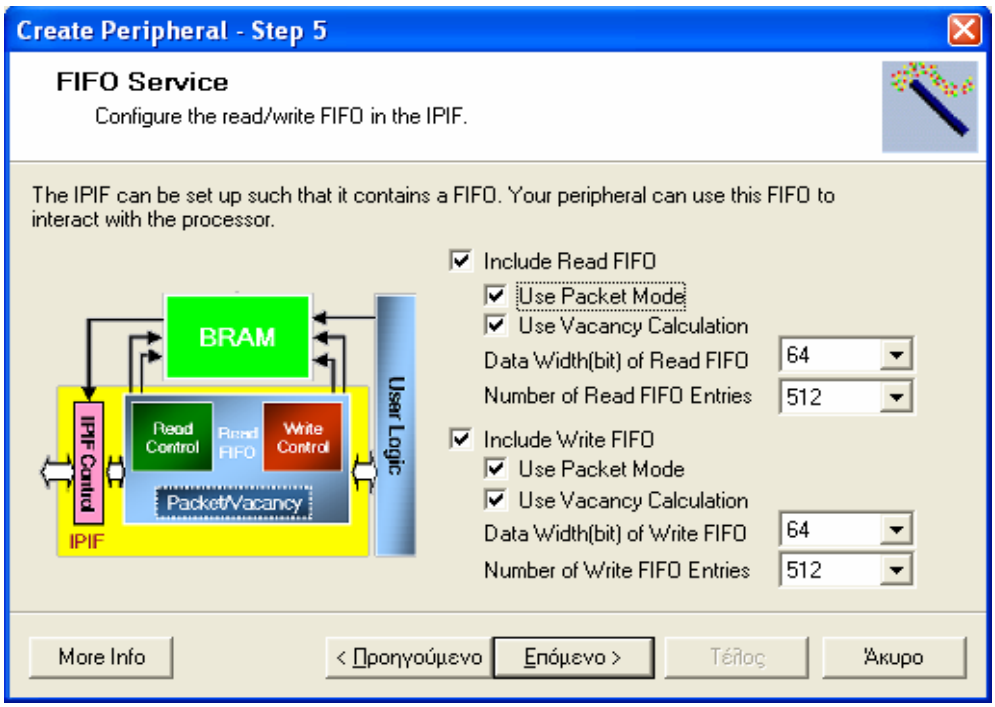
Πίνακας 4.1 – Οι λειτουργίες της διεπαφής IPIF

Λειτουργίες IPIF	Περιγραφή
Μηδενισμός (reset) από εφαρμογή, και καταχωρητής πληροφορίας, Module Information Register (MIR)	Το περιφερειακό διαθέτει μία ειδική διεύθυνση, μόνο για γράψιμο, όπου όταν γράφεται μία συγκεκριμένη λέξη, η διεπαφή IPIF δημιουργεί ένα σήμα μηδενισμού για το περιφερειακό. Με τον τρόπο αυτό το περιφερειακό μπορεί να μηδενιστεί από μία εφαρμογή λογισμικού. Το περιφερειακό έχει επίσης έναν καταχωρητή, μόνο για ανάγνωση που προσδιορίζει την έκδοση του περιφερειακού
Μεταφορά δεδομένων κατά ριπές (burst transaction) και μεταφορές Cacheline	Η λειτουργία αυτή επιτρέπει, με μία μόνο αίτηση, την μεταφορά μεγάλου όγκου δεδομένων. Η μεταφορά Cacheline υποστηρίζεται μόνο για περιφερειακά που συνδέονται στον LMB
DMA	Η διεπαφή IPIF του περιφερειακού υποστηρίζει την απευθείας πρόσβαση της μνήμης, Direct Memory Access, χωρίς την παρέμβαση του επεξεργαστή Ισχύει μόνο για περιφερειακά συνδεδεμένα στον LMB
FIFO	Η διεπαφή IPIF του περιφερειακού υποστηρίζει την λειτουργία ουράς FIFO Ισχύει μόνο για περιφερειακά συνδεδεμένα στον LMB
Μηχανισμός διακοπών στο περιφερειακό	Το περιφερειακό διαθέτη μηχανισμό συλλογής και διαχείρισης διακοπών. Το user-logic και η διεπαφή IPIF, παράγουν μία γραμμή εξόδου διακοπών, προερχόμενη από το περιφερειακό
Καταχωρητές ελεγχόμενοι από το λογισμικό	Το περιφερειακό διαθέτει έναν αριθμό καταχωρητών, οι οποίοι είναι προσβάσιμοι μέσω των εφαρμογών λογισμικού, με δικές τους διευθύνσεις.
Λειτουργία του user-logic σαν κυρίαρχο (master) περιφερειακό	Περιλαμβάνει τα σήματα της διεπαφής IPIC που αφορούν τις λειτουργίες κυρίαρχου (master) περιφερειακού. Συνοδεύεται και από απλό παράδειγμα, σε HDL, για το πως ελέγχεται ένα κυρίαρχο user-logic Ισχύει μόνο για περιφερειακά συνδεδεμένα στον LMB
Πολλαπλές περιοχές διευθύνσεων στο user-logic	Γεννά σήματα επίτρεψης για κάθε περιοχή διευθύνσεων, σε αντίθεση με το σήμα επίτρεψης που γεννάτε για κάθε καταχωρητή πυο ελέγχεται από λογισμικό. Η λειτουργία είναι χρήσιμη για περιφερειακά που χρησιμοποιούν πολλές περιοχές διευθύνσεων



Σχήμα 4.28 – Επιλογή λειτουργιών της διεπαφής IPIF

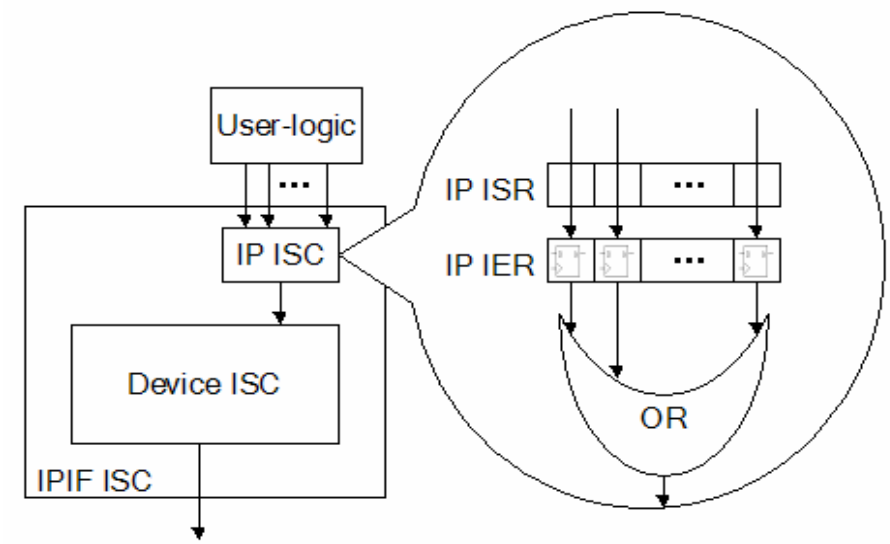
Ανάλογα με τις λειτουργίες που επιλέγονται, αλλά και υποστηρίζονται, τα επόμενα παράθυρα διαλόγου του βοηθού αποσκοπούν στην διαμόρφωση των λειτουργιών αυτών. Η λειτουργία του DMA δεν επιδέχεται αλλαγές και οι μόνες λειτουργίες είναι το γράψιμο και η ανάγνωση δεδομένων, ενώ τονίζεται ότι πρέπει να οριστούν τέσσερις καταχωρητές στην εφαρμογή του χρήστη για την εξυπηρέτηση του DMA. Η διαμόρφωση της λειτουργίας της ουράς FIFO περιλαμβάνει τον καθορισμό του βάθους της ουράς και του μεγέθους των δεδομένων που θα αποθηκεύει. Επίσης μπορεί να υλοποιηθεί ουρά μόνον ανάγνωσης ή εγγραφής ή και τα δύο, ενώ υπάρχουν και άλλα επιλέξιμα χαρακτηριστικά, όπως packet mode και ένδειξη πληρότητας της ουράς.



Σχήμα 4.29 – Διαμόρφωση της ουράς FIFO

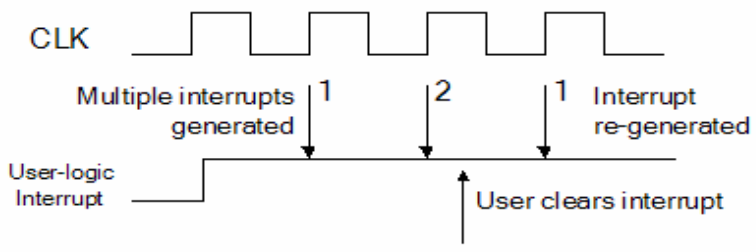
Για τον μηχανισμό των διακοπών, που προέρχονται τόσο από το user-logic όσο και από την διεπαφή IPIF, χρησιμοποιούνται, ένας καταχωρητής, Interrupt Enable Register (IER), προσπελάσιμος μέσω σταθερής διεύθυνσης, για την ενεργοποίηση / απενεργοποίηση των διακοπών, καθώς και οι καταχωρητές, Interrupt Status Register (ISR), που υποδεικνύουν την πηγή και την κατάσταση των διακοπών. Οι διακοπές που προέρχονται από το user-logic επεξεργάζονται πρώτα από τον ελεγκτή IP Interrupt Source Controller (IP ISC). Το σήμα που γεννά ο IP ISC περνάει σε έναν άλλο ελεγκτή, Device Interrupt Source Controller (Device ISC), όπου καταλήγουν και τα σήματα διακοπών που γεννούν οι υπόλοιπες λειτουργίες της IPIF, με τον χρήστη να έχει την επιλογή είτε να απενεργοποιήσει τις διακοπές από τις υπόλοιπες λειτουργίες της IPIF, είτε να επιβάλει σύστημα προτεραιοτήτων στην εξυπηρέτηση όλων των διακοπών. Άλλες επιλογές που αφορούν τον μηχανισμό των διακοπών είναι ο αριθμός των διακοπών που προκαλεί το user-logic, και ο τρόπος σύλληψης της διακοπής. Οι πιθανοί τρόποι σύλληψης είναι έξι:

- INTR_PASS_THRU. Οι διακοπές του user-logic δεν επιδέχονται καμία επεξεργασία, και περνούν απευθείας στον IP ISC και κατόπιν στον Device ISC.



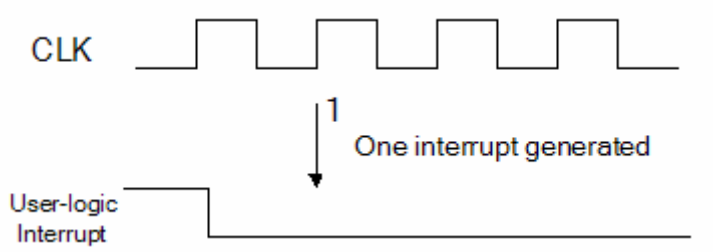
Σχήμα 4.30 – Μέθοδος σύλληψης INTR_PASS_THRU

- INTR_PASS_THRU_INV. Αποτελεί παραλλαγή της πρώτης περίπτωσης, καθώς το μόνο που αλλάζει είναι το ότι τα σήματα των διακοπών που μπαίνουν στον ISR είναι λογικά ανεστραμμένα. Ο τρόπος αυτός χρησιμοποιείται για διακοπές που ενεργοποιούνται σε χαμηλή λογική στάθμη.
- INTR_REG_EVENT. Ο καταχωρητής ISC δειγματοληπτεί το σήμα μιας εισόδου διακοπής στην ανερχόμενη τιμή του κάθε παλμού ρολογιού. Αν δειγματοληπτήσει λογικό ‘1’ τότε το αντίστοιχο bit του παραμένει ‘1’ έως ότου μηδενιστεί η διακοπή από την εφαρμογή του χρήστη.

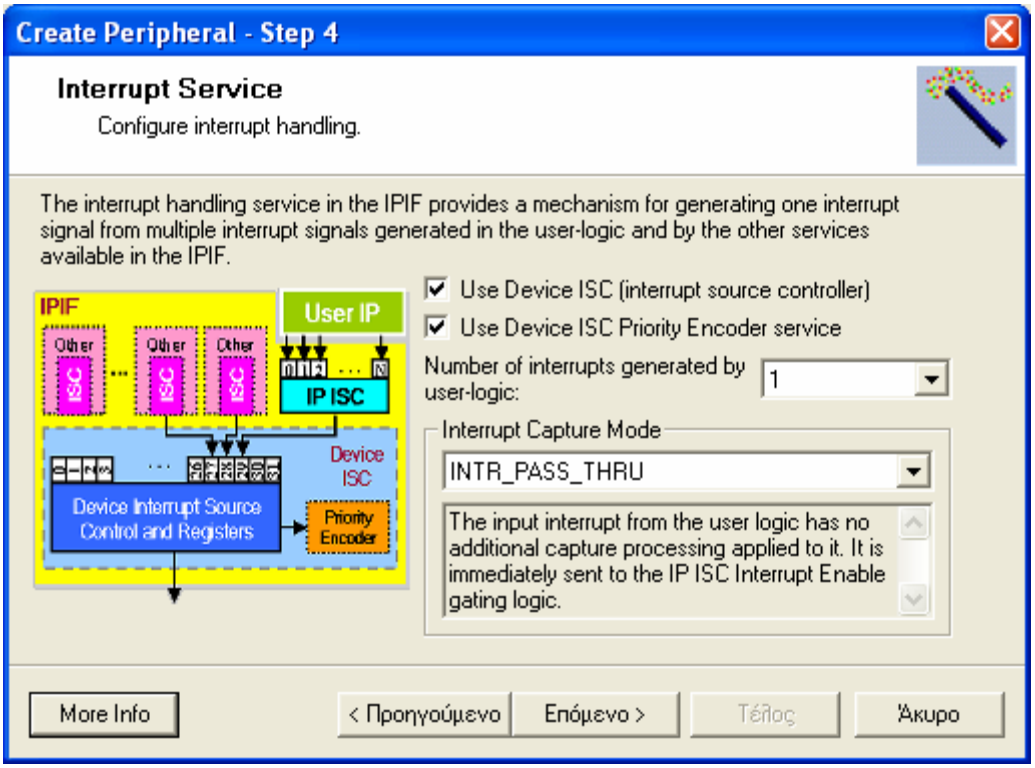


Σχήμα 4.31 – Η λογική της μεθόδου INTR_REG_EVENT

- INTR_REG_EVENT_INV. Αποτελεί παραλλαγή της τρίτης περίπτωσης με την διαφορά ότι τα σήματα των διακοπών είναι λογικά ανεστραμμένα πριν την δειγματοληψία.
- INTR_POS_EDGE_DETECT. Η μέθοδος αυτή έχει ομοιότητες με την τρίτη. Και πάλι ο ISR δειγματοληπτεί τα σήματα των διακοπών, αλλά η ληφθείσα τιμή συγκρίνεται με την χρονικά προηγούμενη. Δηλαδή εάν μια χρονική στιγμή ένα σήμα διακοπής είναι στο λογικό '0' και την επόμενη στο λογικό '1', τότε αυτό λαμβάνεται ως διακοπή. Και πάλι το σήμα διακοπής μηδενίζεται από την εφαρμογή του χρήστη.
- INTR_NEG_EDGE_DETECT. Και εδώ ακολουθείται η ίδια διαδικασία με την παραπάνω περίπτωση, με τη μόνη διαφορά ότι θεωρείται ότι έχει συμβεί διακοπή, όταν μετά από ένα λογικό '1' σε σήμα διακοπής, προκύψει από την δειγματοληψία λογικό '0'.



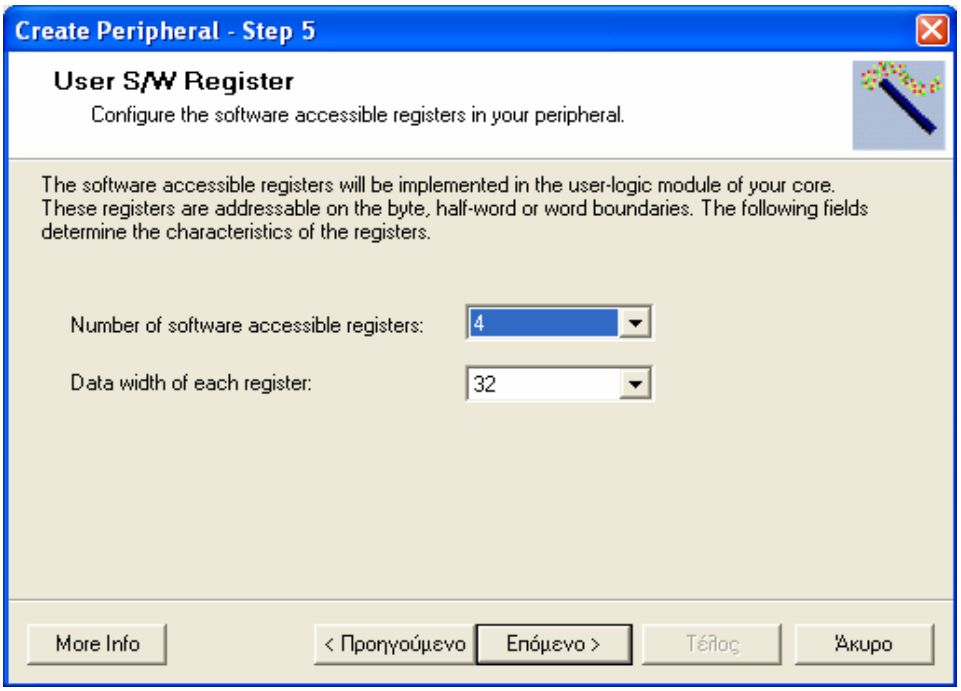
Σχήμα 4.32 – Μέθοδος σύλληψης INTR_NEG_EDGE_DETECT



Σχήμα 4.33 – Διαμόρφωση του μηχανισμού διακοπών

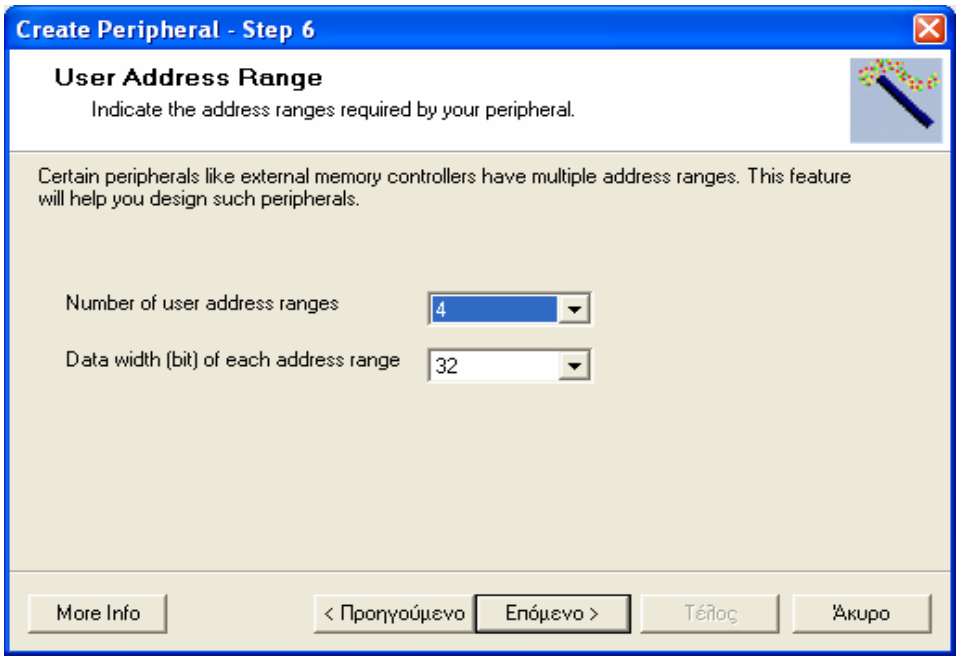
Ένα από τα πιο εύκολα και πιο χρήσιμα εργαλεία της διεπαφής IPIF, είναι οι καταχωρητές που ελέγχονται από εφαρμογές λογισμικού. Με τον τρόπο αυτό μεταφέρονται δεδομένα από και προς το περιφερειακό με δυνατότητα επεξεργασίας από μία εφαρμογή, με έλεγχο των καταχωρητών σαν να ήταν θέσεις μνήμης. Αν επιλεγεί αυτή η λειτουργία, ο βοηθός στο κομμάτι HDL του user-logic προσθέτει και ένα παράδειγμα χειρισμού των καταχωρητών. Ο αριθμός και το μέγεθος, σε bits, των καταχωρητών καθορίζεται από τον χρήστη, ενώ η επιλογή των 32 bit, για περιφερειακά που συνδέονται στον OPB, και των 64 bit, για περιφερειακά που

συνδέονται στον LMB, που αντιστοιχεί στο μέγεθος αυτών των διαδρόμων δεδομένων, είναι η ενδεδειγμένη, καθώς καθιστά ευκολότερη την υλοποίηση της διεπαφής IPIF.



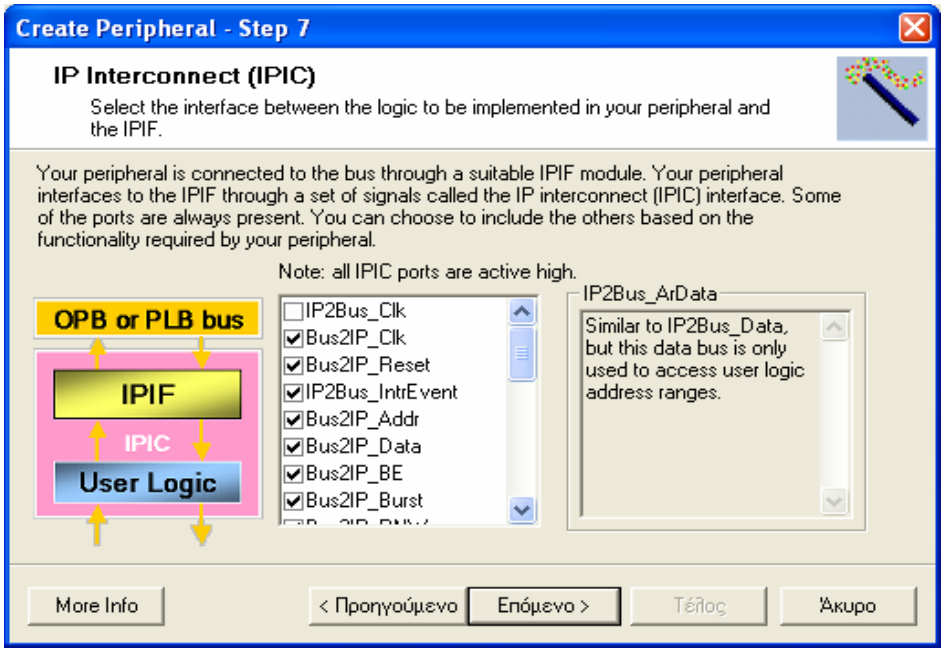
Σχήμα 4.34 – Επιλογή του αριθμού και του μεγέθους των καταχωρητών

Για τις περιπτώσεις όπου ένας αριθμός καταχωρητών δεν εξυπηρετεί το περιφερειακό, όπως για παράδειγμα έναν ελεγκτή μνήμης, τότε δίνεται η δυνατότητα ελέγχου περιοχών διευθύνσεων με την βοήθεια θυρών της διεπαφής IPIC. Αν αυτή η λειτουργία έχει επιλεχθεί (User Logic Address Range Support), τότε ο χρήστης καλείται να καθορίσει των αριθμό των περιοχών διευθύνσεων που χρειάζεται και το μέγεθος, σε bits, των δεδομένων που θα επεξεργάζεται σε αυτές. Ένα σήμα επίτρεψης αντιστοιχίζεται σε κάθε περιοχή διευθύνσεων, σε αντίθεση με την περίπτωση των καταχωρητών όπου κάθε καταχωρητής είχε το δικό του σήμα επίτρεψης.



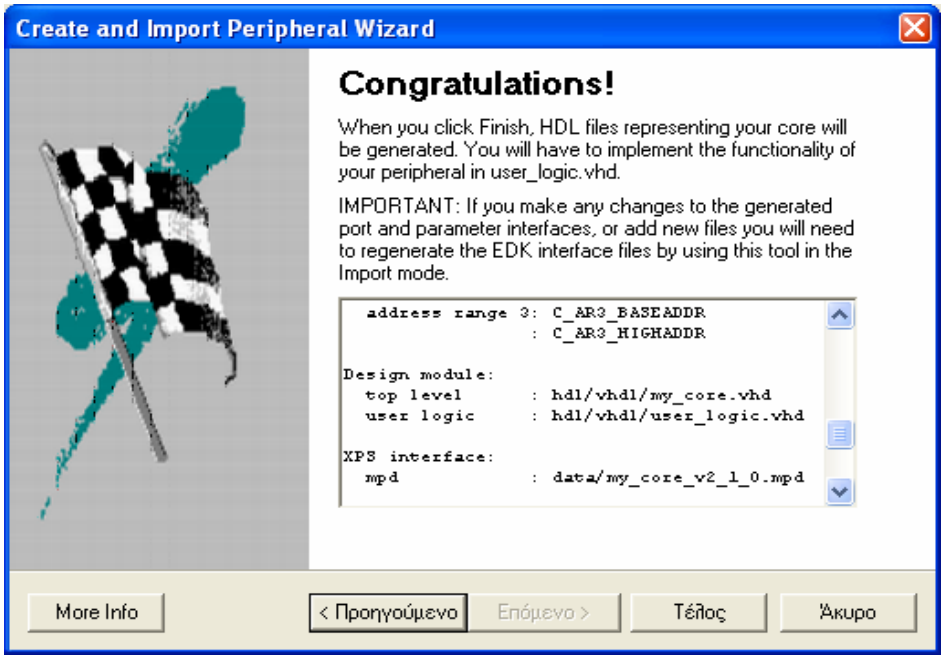
Σχήμα 4.35 – Επιλογή αριθμού και μεγέθους των περιοχών διευθύνσεων

Με την ολοκλήρωση των επιλογών για την διεπαφή IPIF, ο βοηθός ζητά από τον χρήστη να καθορίσει ποιες θύρες της διεπαφής χρειάζεται. Οι αναγκαίες θύρες, καθορίζονται από τις λειτουργίες που θα υποστηρίζει η διεπαφή IPIF και είναι προεπιλεγμένες στο παράθυρο διαλόγου, χωρίς την δυνατότητα μη επιλογής τους, αλλά πέρα από το γεγονός αυτό ο χρήστης έχει την ελευθερία να επιλέξει να έχει πρόσβαση και σε επιπλέον θύρες.



Σχήμα 4.36 – Επιλογή των θυρών της διεπαφής IPIC

Ο βοηθός δεχόμενος σαν είσοδο όλες τις επιλογές που κάνει ο χρήστης, εισάγει την αντίστοιχη πληροφορία στα αρχεία που δημιουργεί και το μόνο που απομένει είναι η συγγραφή του κομματιού του user-logic που αντιστοιχεί στην λειτουργία του περιφερειακού, σε γλώσσα VHDL, και η εισαγωγή του περιφερειακού στο σύστημα. Μια περίληψη του περιφερειακού και των λειτουργιών του γίνεται στο τελευταίο παράθυρο διαλόγου του βοηθού, καθώς και μία αναφορά στα αρχεία που δημιουργούνται.



Σχήμα 4.37 – Περίληψη των ενεργειών του βοηθού

Τα αρχεία που δημιουργεί ο βοηθός είναι επτά. Αν ο φάκελος που περιέχει το project του XPS ονομάζεται project directory και το όνομα και η έκδοση του περιφερειακού είναι my_core_v1_01_b, τότε η θέση και η σημασία των αρχείων είναι η εξής:

- project directory \ pcores \ my_core_v1_01_b \ hdl \ vhdl \ my_core.vhd. Το αρχείο αυτό είναι το top-level του περιφερειακού και μέσα σε αυτό υλοποιείται η διεπαφή IPIF και οι λειτουργίες της, όπως έχουν επιλεγεί από τον χρήστη. Το αρχείο αυτό δεν πρέπει να επεξεργάζεται από τον χρήστη παρά μόνο στην περίπτωση που αυτός θέλει να προσθέσει εξωτερικές θύρες στο περιφερειακό, όπως θα φανεί στο κομμάτι των εφαρμογών αυτού του συγγράμματος.
- project directory \ pcores \ my_core_v1_01_b \ hdl \ vhdl \ user_logic.vhd. Είναι το αρχείο στο οποίο οι πραγματικές λειτουργίες του περιφερειακού, πρέπει να περιγραφούν από τον χρήστη. Στο αρχείο αυτό περιέχεται ένα προσχέδιο αλλά ο χρήστης πρέπει να προσθέσει το σημαντικότερο κομμάτι.
- project directory \ pcores \ my_core_v1_01_b \ data \ my_core_v2_1_0.mpd. Στο αρχείο MPD, Microprocessor Peripheral Description, αναγράφονται οι ιδιότητες, οι παράμετροι και οι θύρες της διεπαφής του περιφερειακού, με τέτοιο τρόπο ώστε να την αντιλαμβάνονται τα εργαλεία του EDK.
- project directory \ pcores \ my_core_v1_01_b \ data \ my_core_v2_1_0.pao. Το αρχείο PAO, Peripheral Analysis Order, καθορίζει την σειρά με την οποία πρέπει να μεταγλωττιστούν τα αρχεία πηγαίου κώδικα HDL, καθώς και τις βιβλιοθήκες που θα πάρουν μέρος στην μεταγλώττιση.
- project directory \ pcores \ my_core_v1_01_b \ devl \ ipwiz.opt. Στο αρχείο αυτό περιέχονται οι επιλογές που έγιναν στον βοηθό μέσω των παραθύρων διαλόγου, σε μορφή όμως που θα χρησιμοποιούταν σε περιβάλλον δέσμης ενεργειών (batch mode).
- project directory \ pcores \ my_core_v1_01_b \ devl \ README.txt. Αυτό είναι ένα αρχείο που έχει χρησιμότητα μόνο για τον χρήστη και περιλαμβάνει την περίληψη του περιφερειακού, την περιγραφή των αρχείων που δημιουργούνται, την ανάλυση των σημάτων της διεπαφής IPIC και την σημασία κάποιων σταθερών παραμέτρων (generics), που χρησιμοποιούνται.
- project directory \ pcores \ my_core_v1_01_b \ devl \ ipwiz.log. Είναι το αρχείο λειτουργίας του βοηθού, που καταγράφει όλες τις κινήσεις του.

Όπως αναφέρθηκε και πιο πάνω το περιφερειακό θα πρέπει να ενσωματωθεί στο υπόλοιπο σύστημα. Για να γίνει αυτό υπάρχουν δύο τρόποι, να χρησιμοποιηθεί και πάλι ο βοηθός αλλά για εισαγωγή νέου περιφερειακού ή να γίνει η προσθήκη από τον χρήστη μέσω του παράθυρου διαλόγου Add/Edit Cores του XPS. Ο δεύτερος τρόπος είναι ο πιο απλός και είναι αυτός που χρησιμοποιήθηκε για τις εφαρμογές, η διαδικασία που πρέπει να ακολουθηθεί περιγράφεται αναλυτικά εκεί.

4.5 Τα Υπόλοιπα Εργαλεία

Όπως είναι φυσικό το EDK διαθέτει πλήθος άλλων εργαλείων, που χρησιμοποιούνται από το XPS, εκτός αυτών που έχουν περιγραφεί, όπως εργαλεία για την εκσφαλμάτωση (XMD) και την προσομοίωση (SimGen), εργαλεία γραφικής επεξεργασίας (PBD Editor). Επίσης υπάρχει και η δυνατότητα λειτουργίας των εργαλείων του EDK χωρίς την χρήση παραθύρων, σε αντίθεση δηλαδή με ό,τι είδαμε μέχρι τώρα. Όλα αυτά είτε δεν χρησιμοποιήθηκαν για τις εφαρμογές, είτε χρησιμοποιήθηκαν σε σημείο που να μην επηρεάζουν σημαντικά την διαμόρφωση του συστήματος και για τον λόγο αυτό δεν αναπτύσσονται αναλυτικά σε αυτό το

σύγγραμμα. Για την δεύτερη περίπτωση υπάρχουν αναλυτικές περιγραφές όλων των ενεργειών, που ακολουθήθηκαν, στο κομμάτι των εφαρμογών.

Τέλος ιδιαίτερη μνεία πρέπει να γίνει σε δύο εργαλεία τα οποία χρησιμοποιήθηκαν στις εφαρμογές αλλά δεν ανήκουν στο EDK. Πρόκειται για το ChipScope και για το ISE, και τα δύο της Xilinx. Το πρώτο χρησιμοποιήθηκε για την παρακολούθηση των σημάτων ενός ενσωματωμένου συστήματος, ενώ το δεύτερο για την υλοποίηση ενός συστήματος δίχως την χρήση του XPS. Και σε αυτήν την περίπτωση τα βήματα για την χρήση των εργαλείων αυτών βρίσκεται στο κομμάτι των εφαρμογών, με τις απαραίτητες επεξηγήσεις, καθώς η χρήση τους είναι πολύ περιορισμένη.

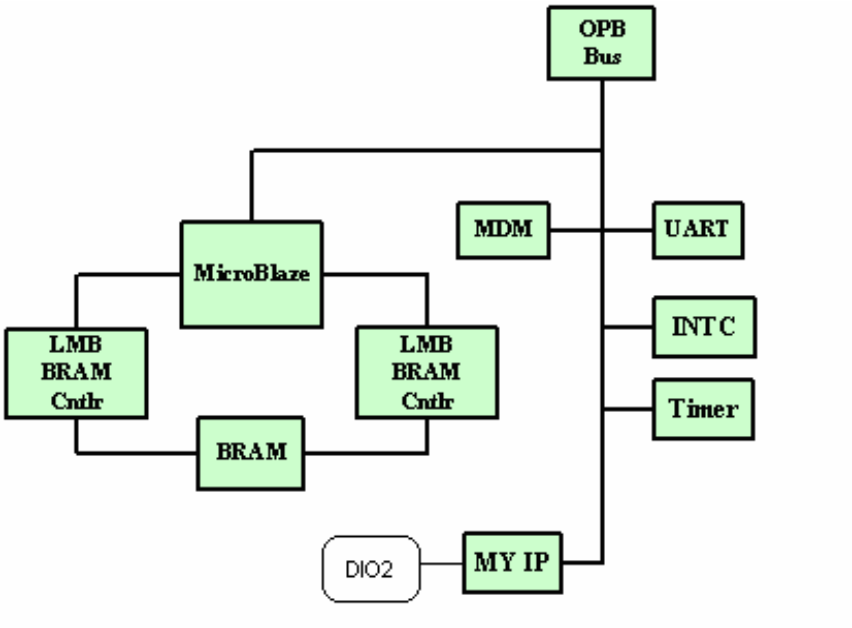
Πηγές

1. Xilinx: “Embedded System Tools Guide”
2. Xilinx: “OPB IPIF Product Specification”

5. Εφαρμογές

5.1 Εισαγωγή

Οι εφαρμογές που πραγματοποιήθηκαν σε αυτήν την διπλωματική εργασία χωρίζονται σε δύο μέρη. Στο πρώτο μέρος περιγράφονται, μέσα από έξι απλά βήματα, οι απαιτούμενες ενέργειες για την δημιουργία ενός πλήρους ενσωματωμένου συστήματος. Στα πρώτα πέντε βήματα χρησιμοποιήθηκαν οι κάρτες Digilab 2 και DIO2, ενώ στο τελευταίο βήμα, λόγω ανάγκης μεγαλύτερης χωρητικότητας, χρησιμοποιήθηκε η κάρτα Spartan-3 Starter Kit. Το τελικό σύστημα που θα δημιουργηθεί, φαίνεται στο σχήμα 5.1



Σχήμα 5.1 – Το πλήρες σύστημα

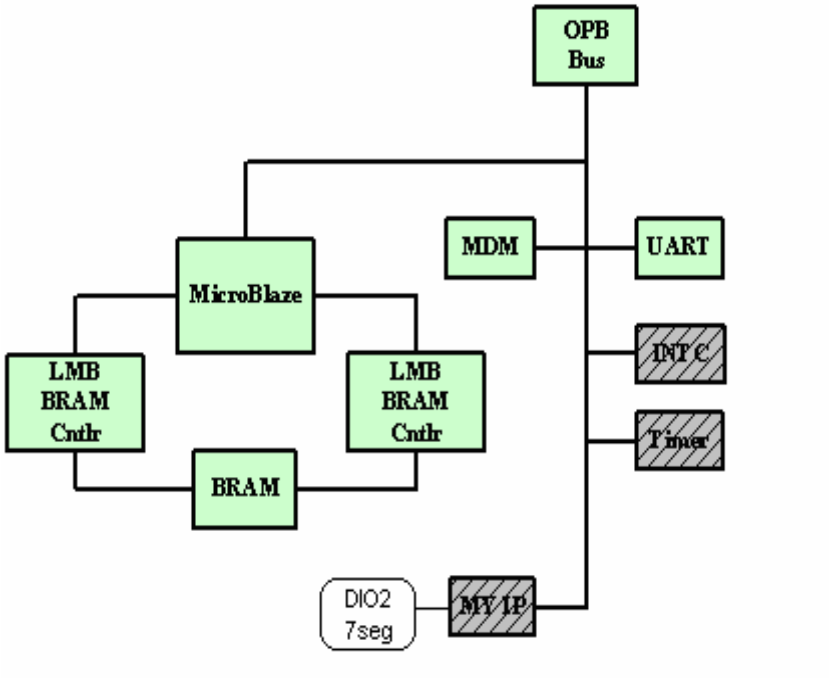
Στο δεύτερο μέρος των εφαρμογών περιγράφεται η κατασκευή ενός φίλτρου και η συνεχής λειτουργία του με την βοήθεια του MicroBlaze και της σειριακής θύρας.

5.2 Εφαρμογή 1^{η} : Σχεδιασμός ενός στοιχειώδους συστήματος

5.2.1 Βήμα 1^{ο} : Σχεδιασμός απλής πλατφόρμας υλικού με τον MicroBlaze

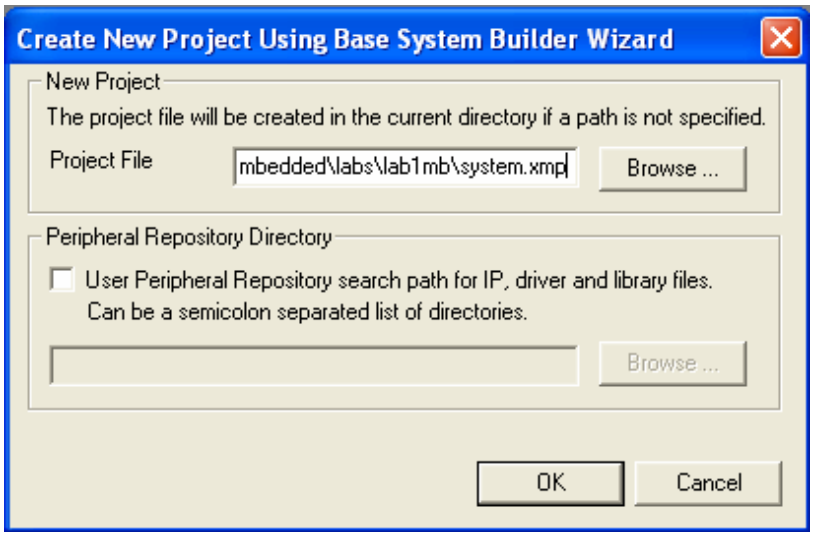
Στο πρώτο βήμα της εφαρμογής θα χρησιμοποιηθεί ο βοηθός BSB του XPS για να δημιουργηθεί ένα σύστημα που να περιέχει τα παρακάτω IP:

- MicroBlaze
- LMB BRAM controllers for BRAM
- OPB bus
- OPB MDM (μονάδα εκσφαλμάτωσης)
- OPB UART



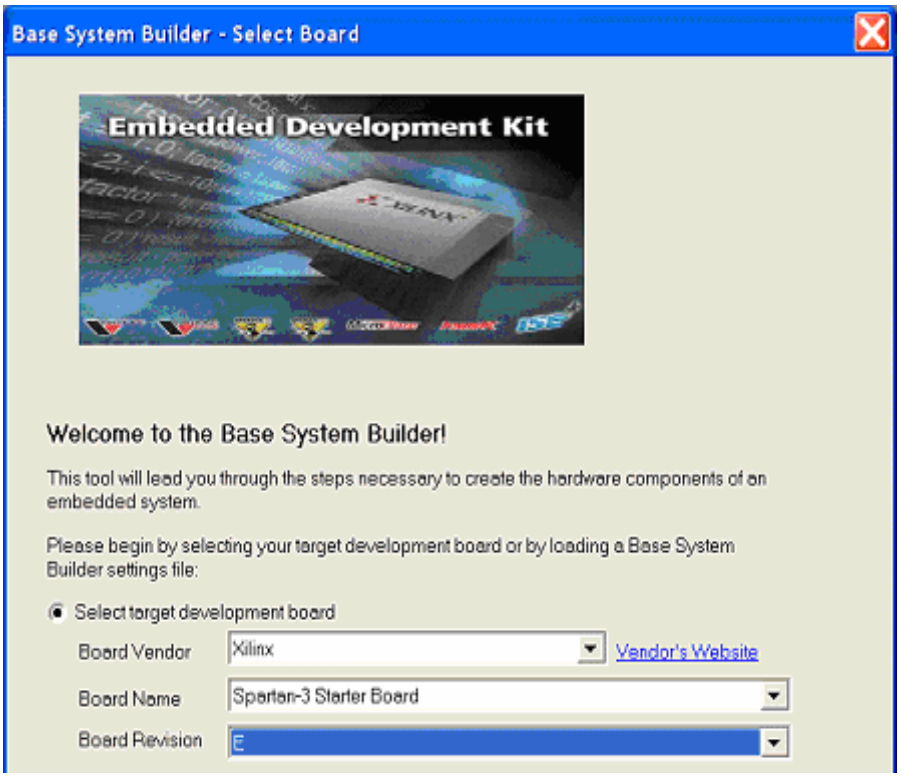
Σχήμα 5.2 – Τα IPs του πρώτου βήματος

Στο μενού του XPS, επιλέγοντας File → New Project → Base System Builder, εμφανίζεται το παράθυρο διαλόγου Create New Project Using Base System Builder Wizard.



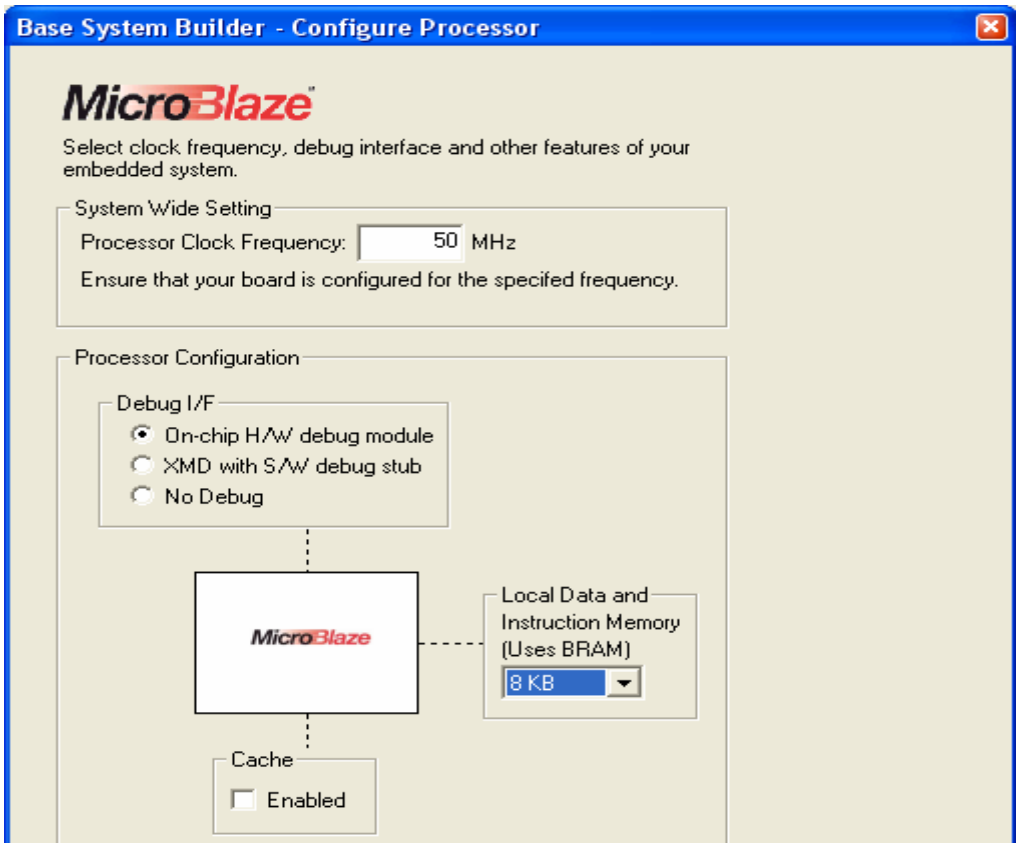
Σχήμα 5.3 – New Base System Builder Wizard Project

Αφού εισαχθεί ο επιθυμητός φάκελος στον οποίον θα αποθηκευτεί το project, πατώντας OK, διαλέγουμε στο επόμενο παράθυρο διαλόγου την αναπτυξιακή κάρτα Spartan-3 Starter Board (Kit) της Xilinx, παρότι τελικά δεν θα χρησιμοποιηθεί αυτή η κάρτα, αλλά η Digilab 2, όπως θα φανεί παρακάτω.



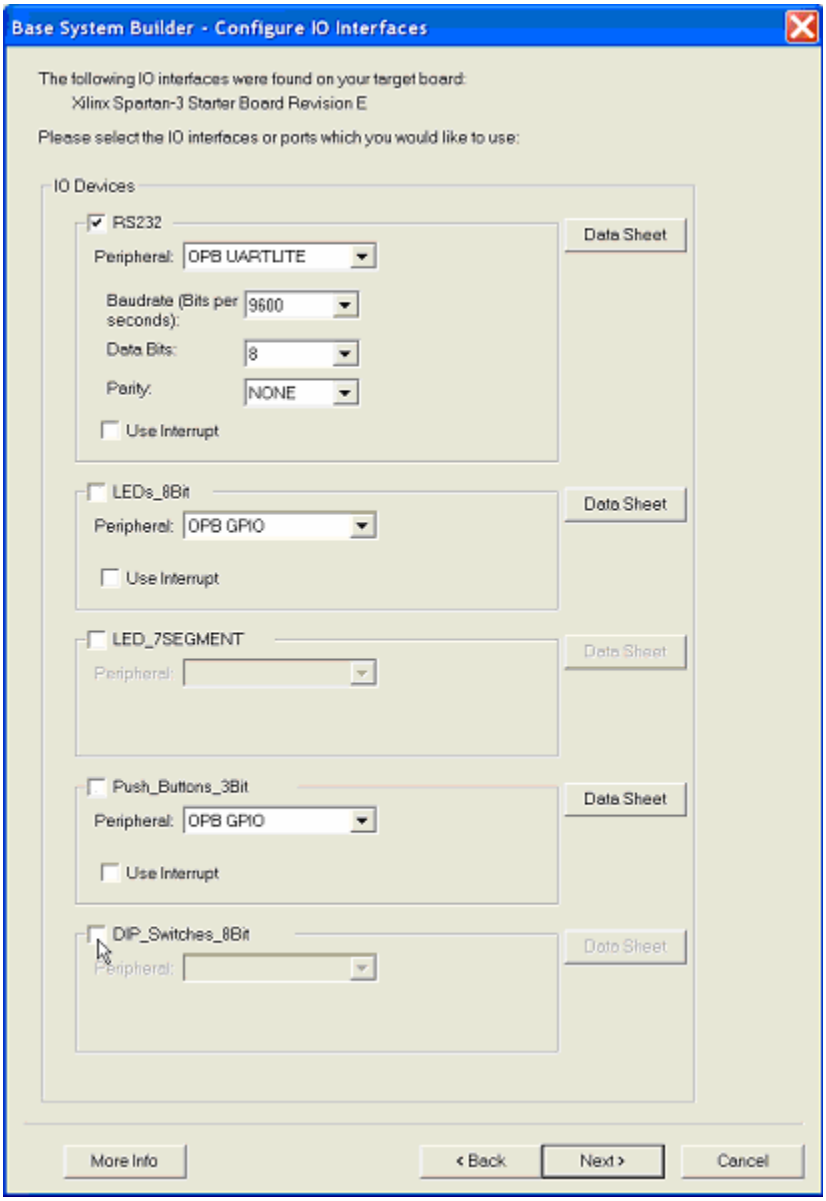
Σχήμα 5.4 – Επιλογή Board

Στο επόμενο παράθυρο γίνεται η επιλογή του επεξεργαστή MicroBlaze, ενώ έπειτα όταν ζητούνται οι παράμετροι του επεξεργαστή και του συστήματος, επιλέγουμε 50 MHz ως την συχνότητα λειτουργίας και 8 KB το μέγεθος της μνήμης BRAM, όπως φαίνεται στο σχήμα 5.5.



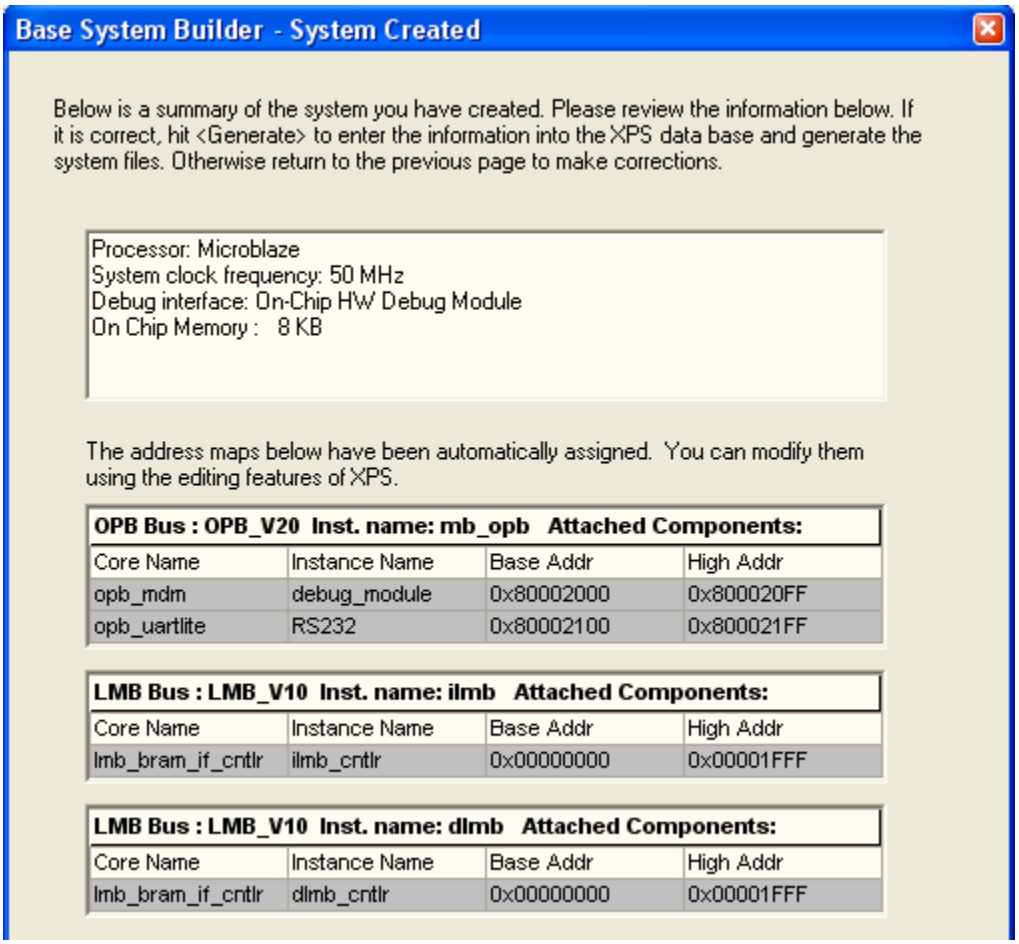
Σχήμα 5.5 – Παράμετροι επεξεργαστή

Στο παράθυρο διαλόγου με τις συσκευές εισόδου / εξόδου επιλέγουμε να προσθέσουμε στο σύστημα μόνο την σειριακή θύρα, αν και δεν θα χρησιμοποιηθεί από την Digilab 2, αφού ο βοηθός BSB απαιτεί να έχει επιλεχθεί τουλάχιστον ένα περιφερειακό.



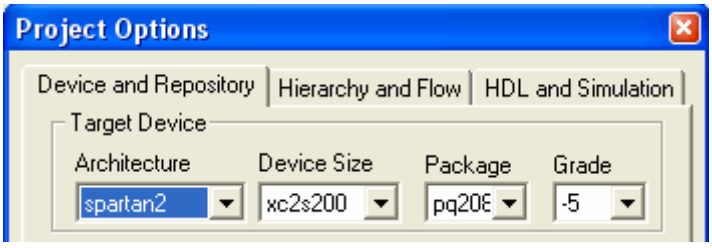
Σχήμα 5.6 – Προσθήκη της σειριακής θύρας

Δεν προσθέτουμε κανένα άλλο περιφερειακό, παρά πατάμε Next μέχρι να φτάσουμε στο παράθυρο διαλόγου Software Configuration και εκεί ορίζουμε την σειριακή θύρα RS232 ως πρότυπη είσοδο και έξοδο. Στο επόμενο παράθυρο διαλόγου βρίσκεται η περίληψη του συστήματος που δημιουργείται και που φαίνεται στο σχήμα 5.7.



Σχήμα 5.7 – Περίληψη συστήματος

Ολοκληρώνοντας τον οδηγό, έχουν δημιουργηθεί όλα τα απαραίτητα αρχεία του project, το οποίο εμφανίζεται σε δενδρική μορφή στο XPS. Το σύστημα όμως που έχει δημιουργηθεί στοχεύει στην κάρτα Spartan-3 Starter Kit και όχι στην Digilab 2, όπως εμείς επιθυμούμε. Για να έχουμε συμβατότητα υλικού και λογισμικού, θα πρέπει να αλλάξουν κάποιες ρυθμίσεις του project. Οι γενικές ρυθμίσεις για τον σκοπό αυτό, βρίσκονται στο παράθυρο διαλόγου Project Options που εμφανίζεται επιλέγοντας Options → Project Options. Στην καρτέλα Device and Repository του παραθύρου αυτού, εισάγουμε τις επιλογές που φαίνονται στο σχήμα 5.8.



Σχήμα 5.8 – Αλλαγή κάρτας

Με τις παραπάνω αλλαγές, πλέον σαν στόχος υλοποίησης είναι η κάρτα Digilab 2, όμως η περιγραφή της πλατφόρμας υλικού που έχει προκύψει από τον οδηγό BSB δεν συμφωνεί με αυτήν την κάρτα. Χρειάζονται ακόμα τρεις αλλαγές. Η πρώτη αφορά το αρχείο MHS. Κάνοντας διπλό-κλικ στο αρχείο system.mhs που βρίσκεται στην καρτέλα System του XPS, στο υπόδεντρο Project Files, αυτό ανοίγει για επεξεργασία στο κυρίως παράθυρο του XPS. Στο τέλος της γραμμής 26 του αρχείου αυτού προσθέτουμε την ιδιότητα SIGIS = CLK, ώστε να γράφει:

PORT sys_rst = sys_rst_s, DIR = input, SIGIS = CLK

Η παραπάνω αλλαγή γίνεται λόγω κάποιας ιδιομορφίας του Digilab 2. Ο ακροδέκτης 77, όπου συνδέεται ένας πιεστικός διακόπτης και όπου συνήθως αντιστοιχίζεται το σήμα reset του συστήματος, είναι ένας ακροδέκτης ρολογιού. Εμείς όμως στην προκειμένη περίπτωση δεν θα το χρησιμοποιήσουμε σαν ρολόι και έτσι το EDK που το λαμβάνει σαν ακροδέκτη σύνδεσης απλού σήματος θα χρησιμοποιήσει διαφορετικού είδους buffer. Στη δημιουργία όμως του netlist και του αρχείου bit, που θα ακολουθήσει παρακάτω, αυτό λαμβάνεται ως λάθος. Έτσι με μία προσθήκη στο αρχείο MHS δηλώνετε αυθαίρετα στα εργαλεία της σύνθεσης ότι το reset είναι μία σύνδεση τύπου ρολόι (CLK) ώστε να είναι σε συμφωνία με το είδος του ακροδέκτη.

Έχοντας σώσει και κλείσει το αρχείο MHS, επιλέγουμε στο ίδιο υπόδεντρο το αρχείο system.ucf. Σε αυτό αντιστοιχίζονται τα εξωτερικά σήματα του συστήματος με τους ακροδέκτες του FPGA. Οι αντιστοιχίσεις μπορούν να προκύψουν από την περιγραφή της κάρτας Digilab 2 στο κεφάλαιο τρία του παρόντος συγγράμματος και θεωρώντας ότι το σήμα μηδενισμού / αρχικοποίησης αντιστοιχεί στον πιεστικό διακόπτη της κάρτας Digilab 2. Αλλάζουμε το αρχείο UCF ώστε να έχει την παρακάτω μορφή:

```
Net sys_clk PERIOD = 20000 ps;
Net RS232_RX  LOC=p202;
Net RS232_TX  LOC=p201;
Net sys_clk       LOC=p80;
Net sys_rst       LOC=p77;
```

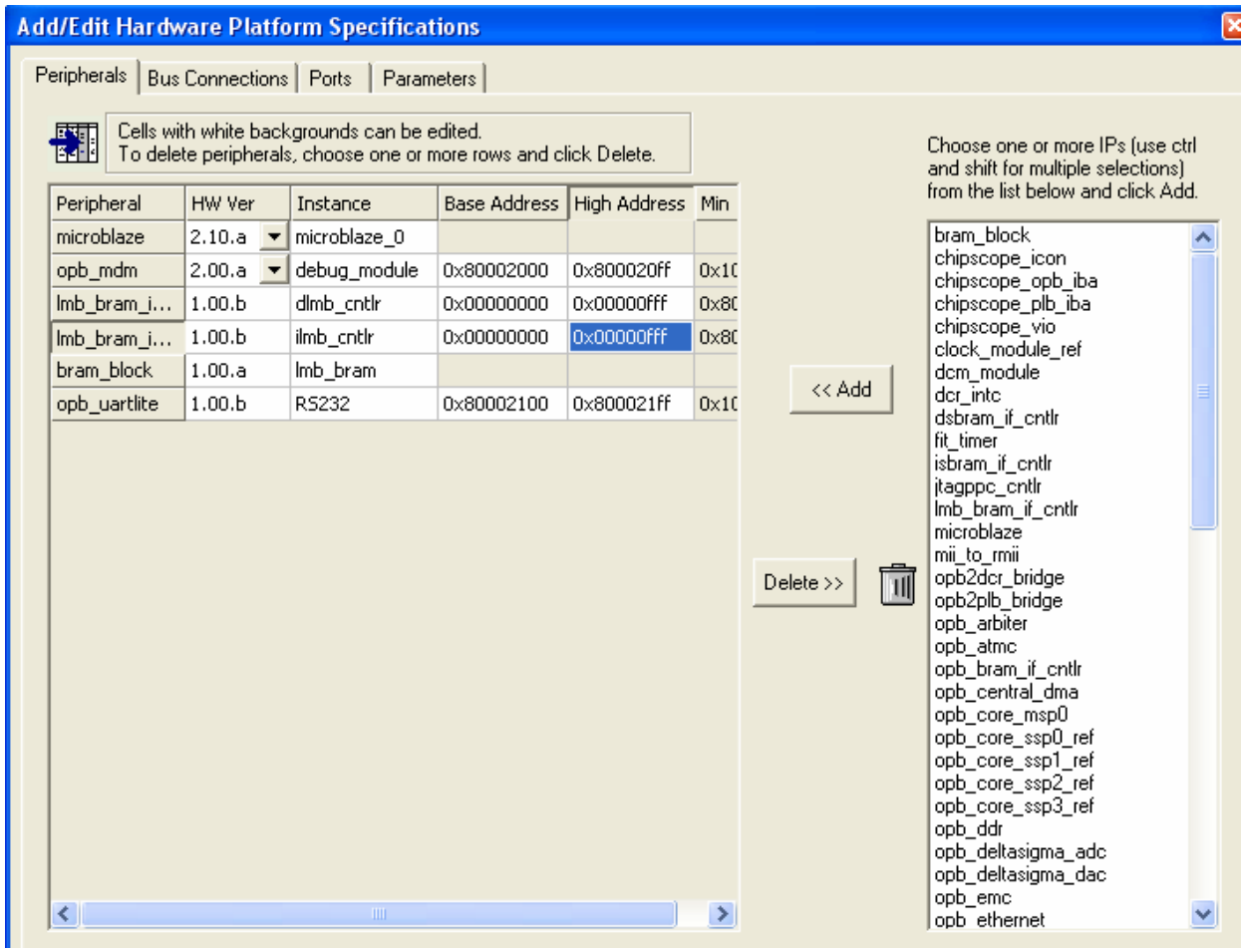
Σώζουμε και κλείνουμε το αρχείο αυτό.

Η πλατφόρμα υλικού έχει τυπικά προσαρμοστεί ώστε να είναι συμβατή με την κάρτα Digilab 2, όμως υπάρχει κάτι ακόμα που πρέπει να αλλάξει. Αν επιχειρήσουμε να δημιουργήσουμε το netlist του συστήματος, τότε το εργαλείο της σύνθεσης, δίνει το λάθος που ακολουθεί:

```
Check platform configuration ...
ERROR:MDT - lmb_bram_if_cntlr (ilmb_cntlr) -
  C:\xup\embedded\labs\diplo\labmb1\system.mhs:102 - address space needs to be
  of a fixed size!
  Only the following memory configurations are supported:
  ---------------------------------------------------------------------------------------
                              |  Memory (kBytes)          |  Memory (kBytes)
  Architecture                |    32-bit data            |    64-bit data
                              |    byte-write             |    byte-write
  ---------------------------------------------------------------------------------------
  Spartan-II                  |       2 4                 |         4
  Spartan-IIE                 |     2 4 8 16              |      4 8 16 32
  Spartan-3                   |    8 16 32 64             |     16 32 64 128
  QPro Virtex                 |     2 4 8 16              |      4 8 16 32
  QPro VirtexE                |     2 4 8 16              |      4 8 16 32
  QPro Virtex-II              |    8 16 32 64             |     16 32 64 128
  QPro-R Virtex               |     2 4 8 16              |      4 8 16 32
  QPro-R Virtex-II            |    8 16 32 64             |     16 32 64 128
  Virtex                      |     2 4 8 16              |      4 8 16 32
  VirtexE                     |     2 4 8 16              |      4 8 16 32
  Virtex-II                   |    8 16 32 64             |     16 32 64 128
  Virtex-II PRO               |    8 16 32 64             |     16 32 64 128
  Virtex-4                    | 2 4 8 16 32 64 128        | 4 8 16 32 64 128 256
```

Στις παραπάνω ενδείξεις επισημαίνεται ότι σε 32-bit λογική και αρχιτεκτονική Spartan-II, η μνήμη BRAM μπορεί να είναι μεγέθους είτε 2 είτε 4 KB. Εμείς επιλέγουμε μέγεθος 4 KB και αυτό επιτυγχάνεται με την βοήθεια του παραθύρου διαλόγου Add/Edit Hardware Platform Specifications, που εμφανίζεται όταν επιλέξουμε Project → Add/Edit Cores… στο μενού του XPS. Εκεί στην καρτέλα Peripherals και στο πεδίο High Address των dlmb_cntlr και ilmb_cntlr αλλάζουμε την τιμή από 0x00001fff σε 0x00000fff. Η αλλαγή αυτή υποδεικνύει ότι η μνήμη

BRAM θα είναι μεγέθους 4KB. Πατάμε Apply και OK και το σύστημα είναι πλέον συμβατό με την κάρτα Digilab 2.

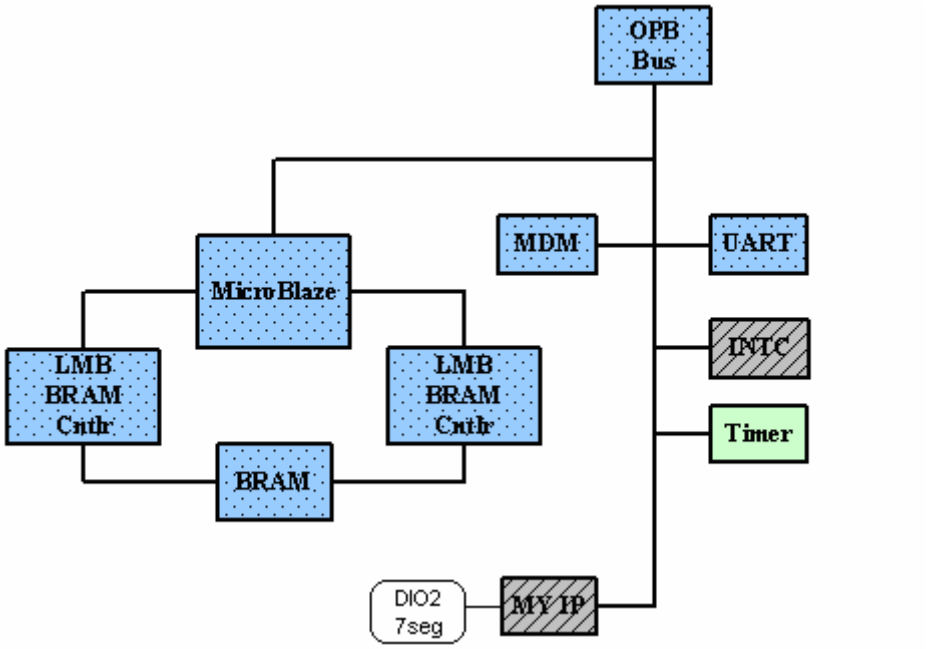


Σχήμα 5.9 – Αλλαγή του μεγέθους της μνήμης BRAM

Έχοντας τελειώσει με τις παραπάνω ρυθμίσεις, μπορούμε να επιλέξουμε Tools → Generate Netlist, για να δημιουργήσουμε το netlist του συστήματος, με την βοήθεια του εργαλείου PlatGen, και έπειτα να δημιουργήσουμε το αρχείο BIT που διαμορφώνει το FPGA, επιλέγοντας Tools → Generate Bitstream. Βέβαια δεν υπάρχει καμία χρησιμότητα στο να κατεβάσουμε το αρχείο bit στο FPGA, αφού ακόμα το σύστημα δεν εκτελεί ουσιαστικά καμία λειτουργία αλλά είναι μία βάση για επέκταση, όπως θα δούμε στα βήματα που θα ακολουθήσουν.

5.2.2 Βήμα 2^{ο} : Προσθήκη IP στην πλατφόρμα υλικού

Στο δεύτερο βήμα, της πρώτης εφαρμογής, θα γίνει η περιγραφή της διαδικασίας για την προσθήκη ενός επιπλέον IP, από αυτά που είναι διαθέσιμα από το EDK, στην πλατφόρμα υλικού. Η προσθήκη θα πραγματοποιηθεί τόσο με την βοήθεια παραθύρων διαλόγου όσο και με τροποποίηση του αρχείου MHS, προκειμένου να γίνει αναφορά και στους δύο τρόπους. Παράλληλα θα δείξουμε τις ενέργειες που πρέπει να γίνουν ώστε να υλοποιηθεί το σύστημα στο ISE της Xilinx, αντί να ακολουθηθεί η ροή του XPS.



Σχήμα 5.10 – Το σύστημα του δεύτερου βήματος

Όπως φαίνεται στο σχήμα 5.10, το IP που θα προστεθεί στο σύστημα είναι ένας χρονομέτρης και συγκεκριμένα ο:

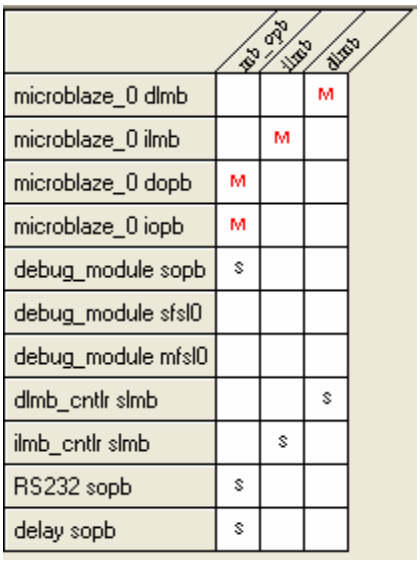
- OPB timer and counter for time delay

Επιλέγοντας Project → Add/Edit Cores από το μενού του XPS, εμφανίζεται το παράθυρο διαλόγου Add/Edit Hardware Platform Specifications. Εκεί στην καρτέλα Peripherals επιλέγουμε από την λίστα με τα IPs, που βρίσκεται δεξιά, το opb_timer, το προσθέτουμε στο σύστημα και μεταβάλουμε τα πεδία της ονομασίας του και των διευθύνσεών του όπως στο σχήμα 5.11.

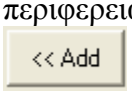
Peripheral	HW Ver	Instance	Base Address	High Address	Min
microblaze	2.10.a	microblaze_0			
opb_mdm	2.00.a	debug_module	0x80002000	0x800020ff	0x100
lmb_bram_i...	1.00.b	dlmb_cntlr	0x00000000	0x00000fff	0x800
lmb_bram_i...	1.00.b	ilmb_cntlr	0x00000000	0x00000fff	0x800
bram_block	1.00.a	lmb_bram			
opb_uartlite	1.00.b	RS232	0x80002100	0x800021ff	0x100
opb_timer	1.00.b	delay	0x80002200	0x800022ff	0x100

Σχήμα 5.11 – Το όνομα και η περιοχή μνήμης του opb_timer

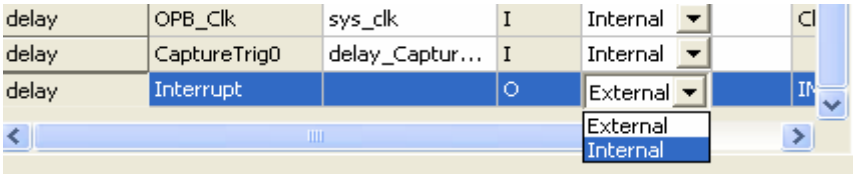
Η επόμενη καρτέλα του παραθύρου Add/Edit Hardware Platform Specifications είναι η καρτέλα Bus Connections. Σε αυτήν ορίζουμε την σύνδεση του νέου περιφερειακού με τον διάδρομο δεδομένων OPB, που έχει μόνο μία επιλογή, δηλαδή να είναι τύπου slave. Ταυτόχρονα παρατηρούμε και ότι μόνον οι συνδέσεις του MicroBlaze είναι τύπου master και όλες οι άλλες, συμπεριλαμβανομένου και του χρονομέτρη delay, είναι τύπου slave.



Σχήμα 5.12 – Οι συνδέσεις των IP στους διαδρόμους δεδομένων

Περνώντας στην καρτέλα Ports βλέπουμε τις θύρες επικοινωνίας των περιφερειακών που απαρτίζουν το σύστημα, μέχρι εκείνη την στιγμή. Πληκτρολογώντας δεξιά στο πεδίο Filter substring or instance την λέξη delay, εμφανίζονται όλες οι θύρες που είναι διαθέσιμες για το νέο περιφερειακό. Επιλέγουμε τις θύρες OPB_Clk, CaptureTrig0 και Interrupt και πατώντας 

τις προσθέτουμε στο σύστημα. Σβήνουμε το Net Name της θύρας Interrupt, αφού ακόμα δεν θα χρησιμοποιήσουμε διακοπές, και μετατρέπουμε τα σήματα όλων των θυρών από εξωτερικά (External) σε εσωτερικά (Internal), όπως στο σχήμα 5.13. Περισσότερα για τις θύρες του χρονομέτρη και για την λειτουργία του θα πούμε σε επόμενο βήμα όπου θα γίνει και χρήση του χρονομέτρη.



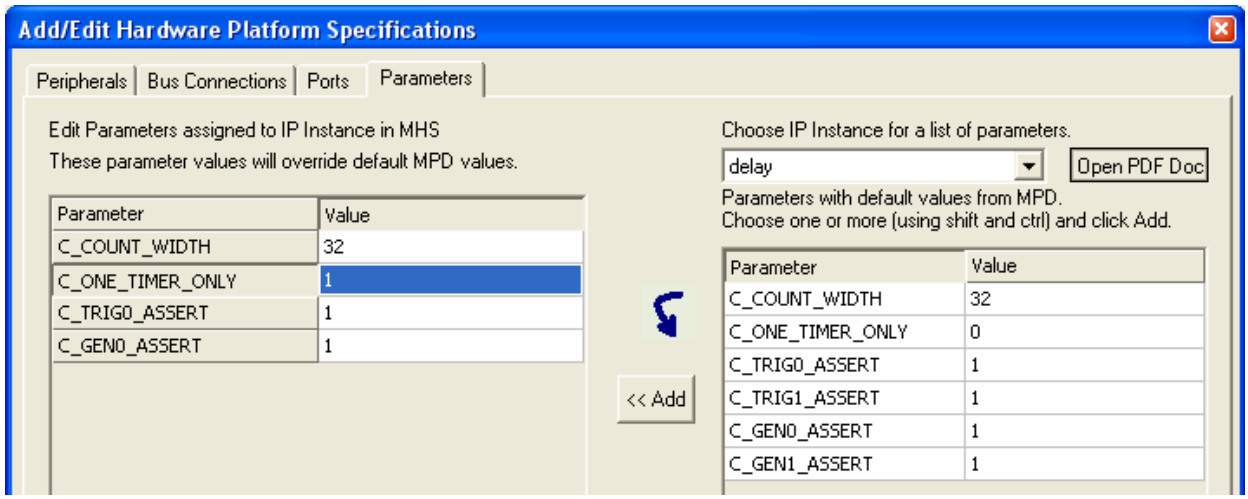
Σχήμα 5.13 – Μετατροπή των σημάτων σε εσωτερικά

Στην τελευταία καρτέλα, Parameters, γίνεται ανάθεση τιμών για τις παραμέτρους των περιφερειακών. Δεξιά στο πεδίο Choose IP instance for a list of parameters, επιλέγουμε delay και από τις παραμέτρους που εμφανίζονται επιλέγουμε αυτές που φαίνονται στον πίνακα 5.1 και πατάμε το κουμπί Add.

Πίνακας 5.1 – Παράμετροι του χρονομέτρη

Parameter	Default Value	Σχόλιο
C_COUNT_WIDTH	32	32-bit μετρητής / χρονομέτρης
C_ONE_TIMER_ONLY	0	'0' : Δύο μετρητές '1' : Ένας μετρητής
C_TRIG0_ASSERT	1	Είσοδος του CaptureTrig0 στο '1' (High)
C_GEN0_ASSERT	1	Έξοδος του GenerateOut0 στο '1' (High)

Από τις παραμέτρους του πίνακα αλλάζουμε την τιμή της C_ONE_TIMER_ONLY από 0 σε 1 αφού θα χρησιμοποιήσουμε μόνο έναν χρονομέτρη.



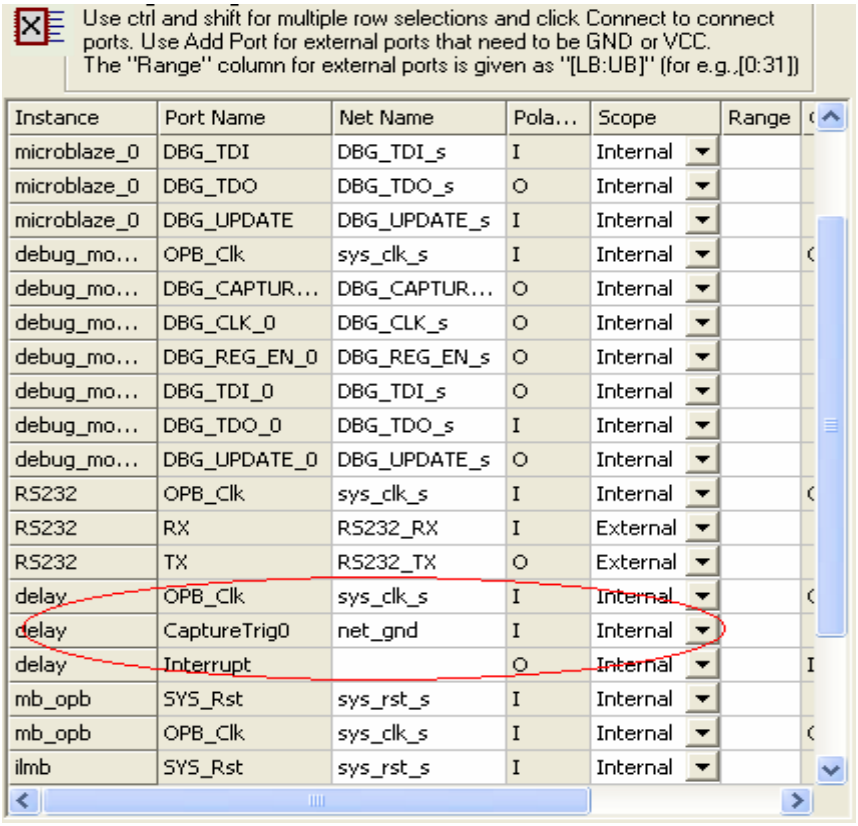
Σχήμα 5.14 – Τροποποίηση παραμέτρων

Πατώντας OK δεχόμαστε όλες τις αλλαγές που πραγματοποιήθηκαν στο παράθυρο διαλόγου και το κλείνουμε.

Όπως είδαμε στην καρτέλα Ports αλλάξαμε τα Net Names των θυρών του στοιχείου delay. Αυτή η αλλαγή όπως και όλες οι υπόλοιπες μπορούν να γίνουν μέσω της επεξεργασίας του αρχείου MHS. Στην καρτέλα System του XPS, στο υπόδεντρο project Files, κάνουμε διπλό-κλικ στο αρχείο system.mhs και αυτό ανοίγει στο κυρίως παράθυρο του XPS. Προς το τέλος του αρχείου, εντοπίζουμε δύο θύρες του στοιχείου delay. Τις τροποποιούμε όπως φαίνεται παρακάτω:

```
PORT OPB_Clk = sys_clk_s
PORT CaptureTrig0 = net_gnd
```

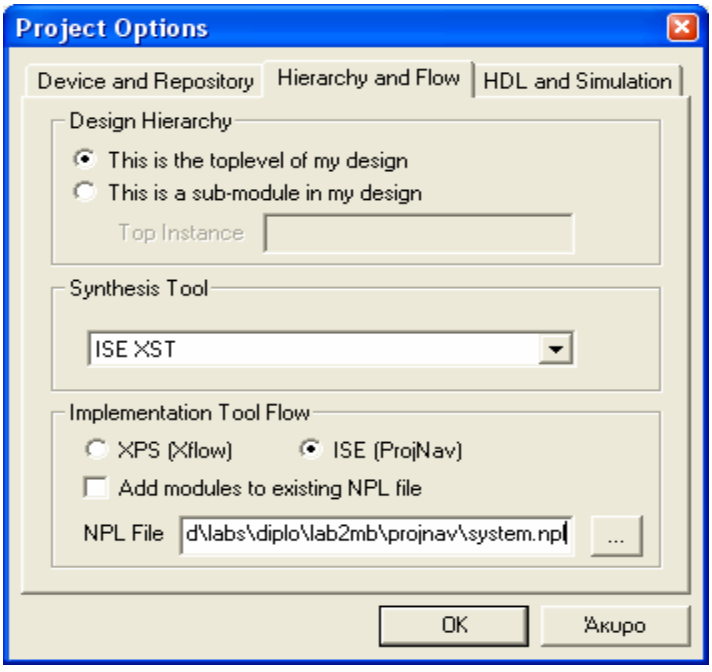
Προτού κλείσουμε και σώσουμε το αρχείο, παρατηρούμε την δομή του αρχείου MHS και μπορούμε να διακρίνουμε τα στοιχεία που αποτελούν το σύστημα, τις εξωτερικές και εσωτερικές συνδέσεις αυτών, καθώς και τον χάρτη διευθύνσεων. Συνεπώς κάθε αλλαγή στο αρχείο MHS αντιστοιχεί με μία αλλαγή στο παράθυρο διαλόγου Add/Edit Hardware Platform Specifications. Για να φανεί καλύτερα αυτό, αφού έχουμε σώσει και κλείσει το αρχείο system.mhs, επιλέγουμε Project → Add/Edit Cores και μεταβαίνουμε στην καρτέλα Ports. Εκεί βλέπουμε ότι πράγματι τα Net Names των δύο θυρών που αλλάξαμε στο αρχείο MHS, έχουν αλλάξει και σε αυτό το παράθυρο διαλόγου.



Instance	Port Name	Net Name	Pola...	Scope	Range
microblaze_0	DBG_TDI	DBG_TDI_s	I	Internal	
microblaze_0	DBG_TDO	DBG_TDO_s	O	Internal	
microblaze_0	DBG_UPDATE	DBG_UPDATE_s	I	Internal	
debug_mo...	OPB_Clk	sys_clk_s	I	Internal	
debug_mo...	DBG_CAPTUR...	DBG_CAPTUR...	O	Internal	
debug_mo...	DBG_CLK_0	DBG_CLK_s	O	Internal	
debug_mo...	DBG_REG_EN_0	DBG_REG_EN_s	O	Internal	
debug_mo...	DBG_TDI_0	DBG_TDI_s	O	Internal	
debug_mo...	DBG_TDO_0	DBG_TDO_s	I	Internal	
debug_mo...	DBG_UPDATE_0	DBG_UPDATE_s	O	Internal	
RS232	OPB_Clk	sys_clk_s	I	Internal	
RS232	RX	RS232_RX	I	External	
RS232	TX	RS232_TX	O	External	
delay	OPB_Clk	sys_clk_s	I	Internal	
delay	CaptureTrig0	net_gnd	I	Internal	
delay	Interrupt		O	Internal	
mb_opb	SYS_Rst	sys_rst_s	I	Internal	
mb_opb	OPB_Clk	sys_clk_s	I	Internal	
ilmb	SYS_Rst	sys_rst_s	I	Internal	

Σχήμα 5.15 – Η αλλαγή των Net Names

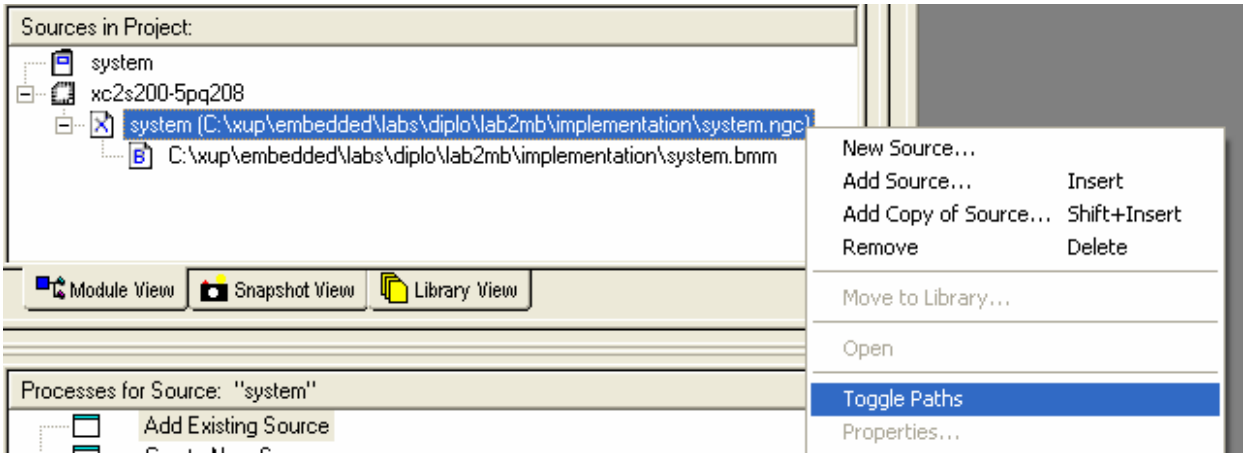
Λόγω του ότι το αρχείο MHS έχει αλλάξει, θα πρέπει να ξανά-δημιουργηθεί το netlist και το bitstream του συστήματος. Στο πρώτο βήμα, αυτό έγινε μέσα στο XPS, τώρα όμως θα δούμε και έναν άλλο τρόπο μέσω του ISE της Xilinx. Για να αλλάξει η ροή της υλοποίησης θα πρέπει να αλλάξουν οι ρυθμίσεις του project. Επιλέγουμε Options → Project Options για να εμφανιστεί το παράθυρο διαλόγου Project Options. Στην καρτέλα Hierarchy and Flow επιλέγουμε το εργαλείο υλοποίησης ISE (ProjNav) και το φάκελο όπου θα αποθηκευτεί το project για αυτό το εργαλείο. (διατηρούμε τον προεπιλεγμένο φάκελο).



Σχήμα 5.16 – Επιλογή εργαλείου υλοποίησης

Πατάμε OK για να αποθηκεύσουμε τις ρυθμίσεις και επιλέγουμε Tools → Generate Netlist για να δημιουργηθεί το netlist του συστήματος. Όταν ολοκληρωθεί το netlist επιλέγουμε Tools → Export to ProjNav. Με την επιλογή αυτή τα κατάλληλα αρχεία θα προστεθούν σε ένα project για το Project Navigator του ISE.

Κλείνουμε το XPS, ανοίγουμε το Project Navigator και μαζί το project system.npl, που έχει προέλθει από το XPS. Στο παράθυρο Sources in Project περικόπτουμε τις διαδρομές των αρχείων, ώστε να είναι σχετικές και όχι απόλυτες. Με τον τρόπο αυτό επιτρέπεται η μεταφορά του project σε μία διαφορετική δομή φακέλων.



Σχήμα 5.17 – Περικοπή διαδρομής αρχείου

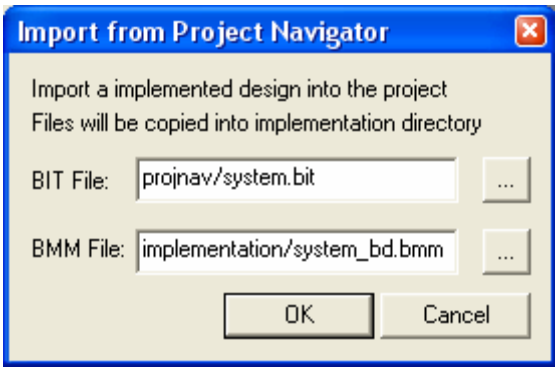
Επιλέγουμε Project → Add Source από το μενού του Project Navigator και προσθέτουμε το αρχείο UCF, system.ucf, από τον φάκελο ..\data. Έπειτα ακολουθούμε τα παρακάτω βήματα:

- Στο μενού του Project Navigator επιλέγουμε Edit → Preferences
- Στην καρτέλα Processes επιλέγουμε Advanced και πατάμε OK
- Στο παράθυρο Sources in Project επιλέγουμε system
- Στο κάτω παράθυρο, Processes for Source: "system", κάνουμε δεξί-κλικ στο Implement Design και επιλέγουμε Properties
- Επιλέγουμε την καρτέλα Translate Properties και επαληθεύουμε ότι το ..\implementation είναι ορισμένο σαν Macro Search Path
- Πατάμε OK
- Κάνουμε διπλό-κλικ στο Implement Design
- Κάνουμε διπλό-κλικ στο Generate Programming File για να δημιουργηθεί το αρχείο BIT

Αφού ολοκληρωθεί η παραπάνω διαδικασία δύο σημαντικά αρχεία θα έχουν δημιουργηθεί :

1. Το αρχείο system.bit. Αυτό το αρχείο BIT αποτελεί το bitstream του υλοποιημένου συστήματος.
2. Το αρχείο system_bd.bmm. Αυτό το αρχείο BMM περιέχει τους χωρικούς περιορισμούς για την BRAM που χρησιμοποιείται στο σύστημα. Το εργαλείο Data2BRAM το συμβουλεύεται για να ανανεώσει τα περιεχόμενα της BRAM με τον κώδικα των εφαρμογών του επεξεργαστή

Κλείνουμε τον Project Navigator και ανοίγουμε πάλι το project που επεξεργαζόμασταν στο XPS. Η ενέργεια της υλοποίησης έχει ολοκληρωθεί από ένα εργαλείο εκτός του XPS και τώρα τα αποτελέσματα της υλοποίησης πρέπει να επιστρέψουν σε αυτό. Επιλέγουμε Tools → Import from ProjNav, πατάμε OK και τα δημιουργημένα αρχεία μεταφέρονται στο project του XPS.

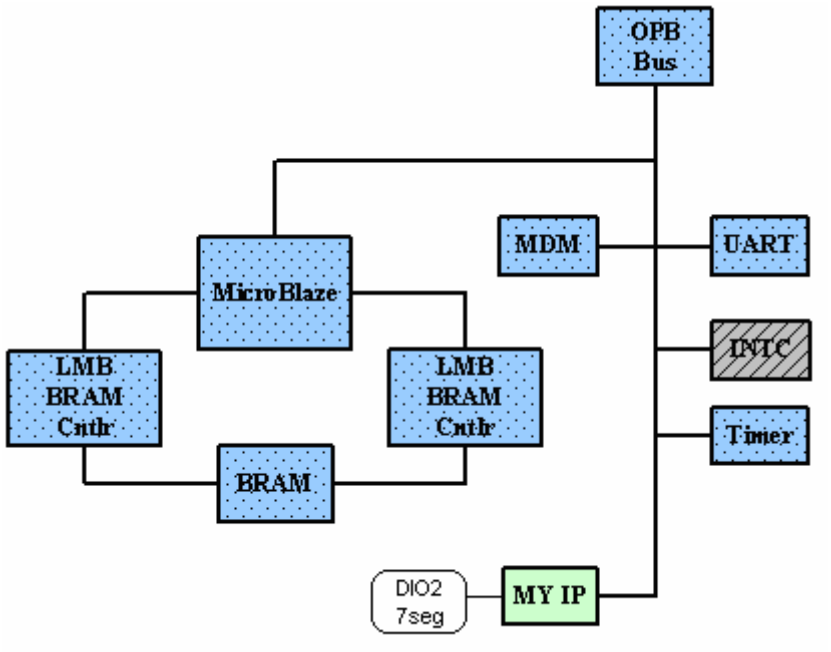


Σχήμα 5.18 – Εισαγωγή αρχείων από το Project Navigator

Το σύστημα είναι πλέον έτοιμο να κατέβει στο FPGA, όμως και πάλι δεν εκτελεί κάποια εργασία ώστε να γίνει η απαραίτητη δοκιμή. Αυτό θα γίνει στο επόμενο βήμα που ακολουθεί.

5.2.3 Βήμα 3ο : Προσθήκη ενός IP του χρήστη στην πλατφόρμα υλικού

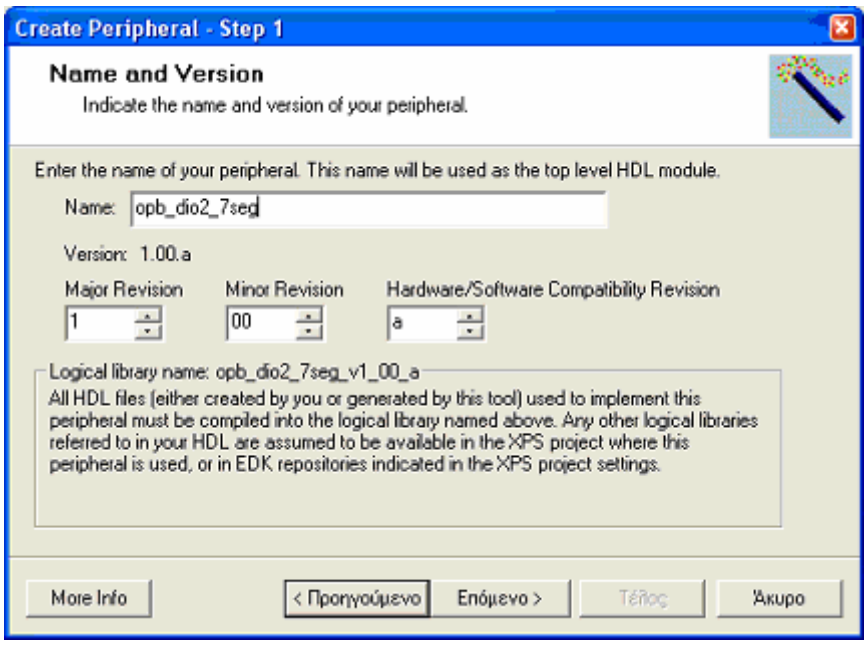
Στο βήμα αυτό σε αντιστοιχία με το προηγούμενο θα προσθέσουμε ένα IP στην πλατφόρμα υλικού, όμως αυτή την φορά το IP δεν θα προέρχεται από το EDK, αλλά θα το διαμορφώσουμε εμείς με την βοήθεια του οδηγού του XPS, Import Peripheral Wizard. Το IP μας θα ελέγχει την οθόνη επτά τμημάτων της κάρτας DIO2 μέσω της κάρτας Digilab 2. Το επεκταμένο σύστημα που θα προκύψει, φαίνεται στο σχήμα 5.19.



Σχήμα 5.19 – Το σύστημα του τρίτου βήματος

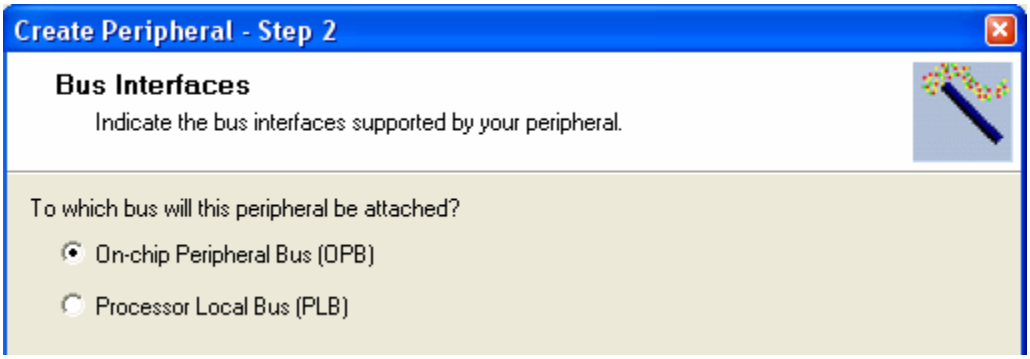
Η περιγραφή του ελεγκτή για το περιφερειακό μας θα γραφεί σε γλώσσα VHDL, όμως πριν από αυτό θα πρέπει να δημιουργηθούν τα κατάλληλα αρχεία, και σε κατάλληλη δομή, ώστε να αναγνωρίζει το EDK το περιφερειακό μας. Σε αυτό θα μας οδηγήσει ο βοηθός του XPS Import Peripheral Wizard, του οποίου η περαιτέρω ανάλυση βρίσκεται στο κεφάλαιο τέσσερα του παρόντος συγγράμματος.

Επιλέγουμε Tools → Import Peripheral Wizard στο μενού του XPS και εμφανίζεται το εισαγωγικό παράθυρο του βοηθού. Πατάμε Next και στο επόμενο παράθυρο διαλόγου επιλέγουμε Create templates for a new peripheral, για να δημιουργήσουμε τα αρχεία υποστήριξης για ένα νέο περιφερειακό. Στο επόμενο παράθυρο διαλόγου επιλέγουμε να αποθηκεύσουμε τα αρχεία του περιφερειακού στο ήδη υπάρχον project του συστήματος και στο επόμενο δίνουμε το όνομα και την έκδοση του περιφερειακού, όπως στο σχήμα 5.20.



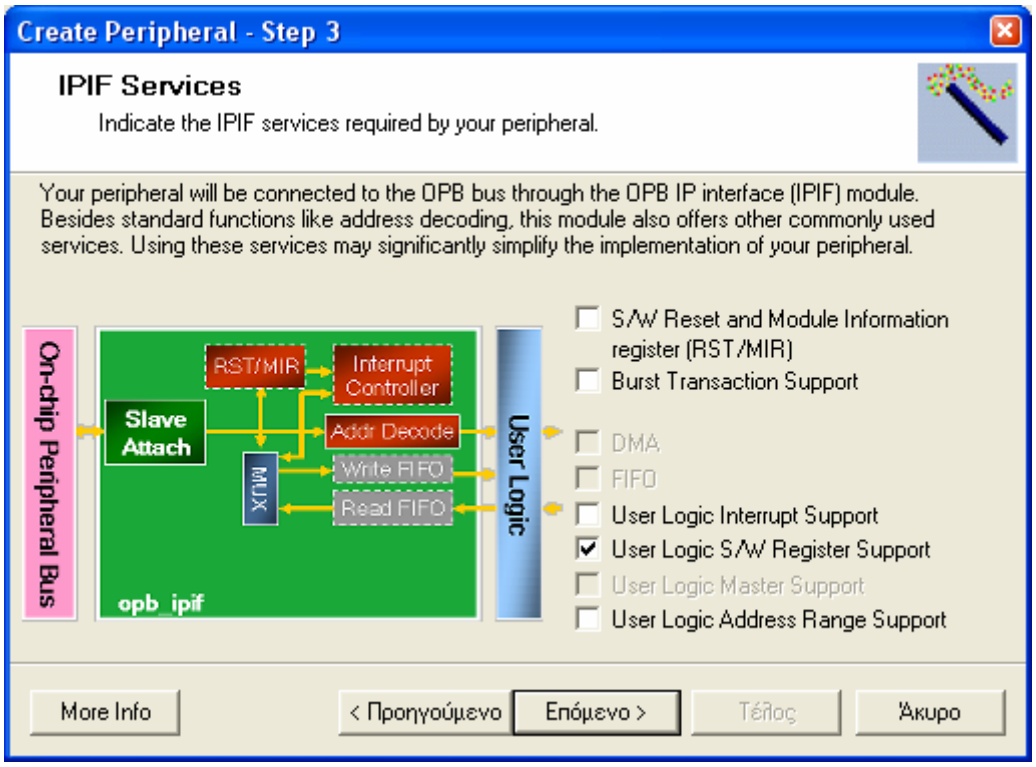
Σχήμα 5.20 – Εισαγωγή ονόματος και έκδοσης περιφερειακού

Πατάμε Next και επιλέγουμε την σύνδεση του περιφερειακού μας στον διάδρομο δεδομένων OPB.



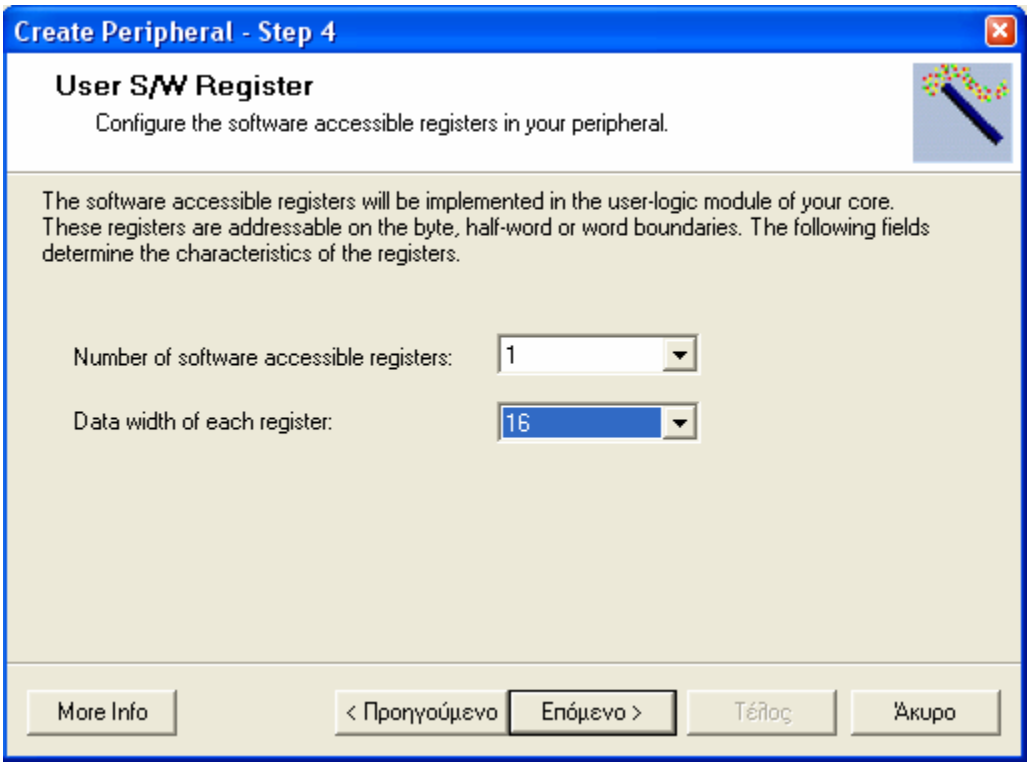
Σχήμα 5.21 – Επιλογή διαδρόμου δεδομένων

Στο επόμενο παράθυρο διαλόγου ζητούνται οι επιθυμητές υπηρεσίες που θα υποστηρίζει η διεπαφή IPIF. Ο ελεγκτής μας χρειάζεται μονάχα καταχωρητές ελεγχόμενους από το λογισμικό και αυτή είναι η επιλογή, όπως φαίνεται στο σχήμα 5.22.



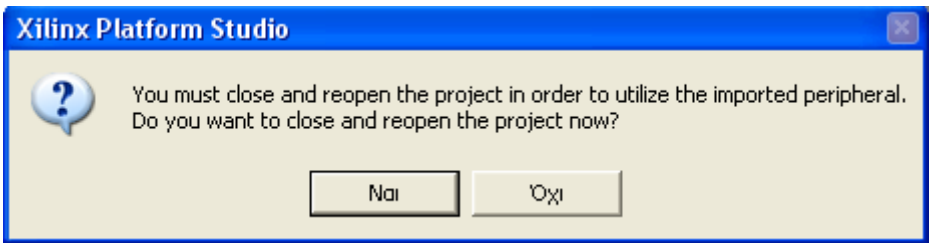
Σχήμα 5.22 – Επιλογή των υπηρεσιών της διεπαφής IPIF

Στο επόμενο παράθυρο διαλόγου ορίζονται οι ρυθμίσεις για την μοναδική υπηρεσία της διεπαφής IPIF που επιλέξαμε. Χρειαζόμαστε μονάχα έναν καταχωρητή των 16 bit για να αποθηκεύουμε τους τέσσερις αριθμούς των 4bit που θα είναι προς εμφάνιση στην οθόνη επτά τμημάτων. Επομένως ορίζουμε αυτόν τον καταχωρητή όπως στο σχήμα 5.23.



Σχήμα 5.23 – Αριθμός και μέγεθος καταχωρητών

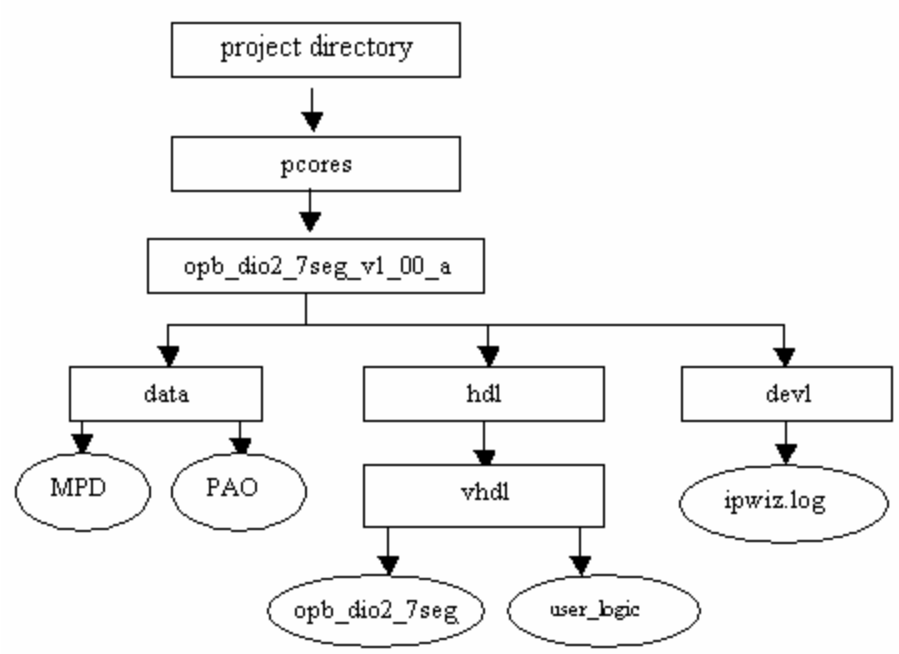
Πατώντας Next ο βοηθός μας ζητά να διαλέξουμε τα σήματα της διεπαφής IPIC που επιθυμούμε, με τα απαραίτητα να είναι προεπιλεγμένα. Τα προεπιλεγμένα σήματα μας αρκούν και συνεπώς πατώντας συνεχώς Next φτάνουμε στο τελευταίο παράθυρο του οδηγού. Εκεί πατάμε Finish και εμφανίζεται η ειδοποίηση του σχήματος 5.24, προκειμένου να κλείσουμε και να ξανά-ανοίξουμε το project ώστε το νέο μας περιφερειακό να ληφθεί υπόψη.



Σχήμα 5.24 – Προτροπή για κλείσιμο του project

Όμως υπάρχουν ακόμα κάποιες τροποποιήσεις που πρέπει να γίνουν, ώστε το περιφερειακό μας να είναι έτοιμο για χρήση, και συνεπώς στην προτροπή του XPS επιλέγουμε No.

Αν περιηγηθούμε στους φακέλους του project τότε θα διαπιστώσουμε την δομή φακέλων που φαίνεται στο σχήμα 5.25 και η οποία σχετίζεται με το νέο περιφερειακό.



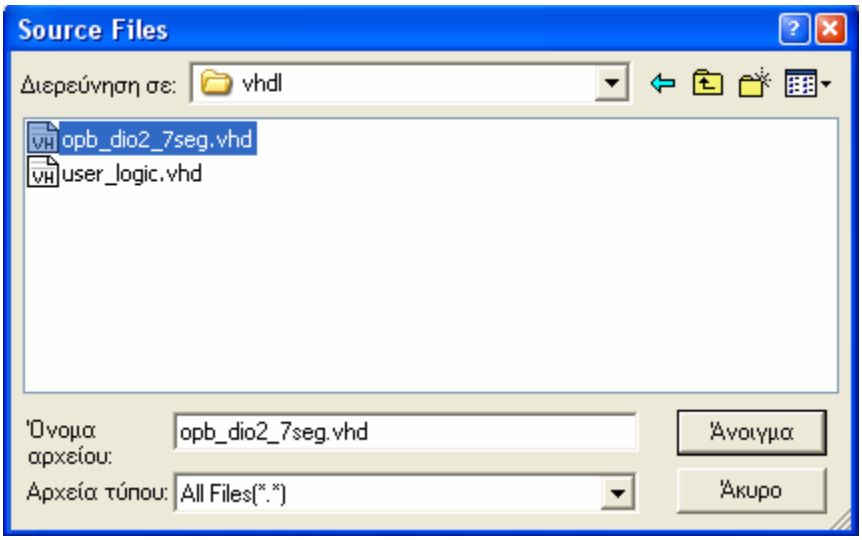
Σχήμα 5.25 – Η δομή φακέλων και αρχείων που δημιουργείται από τον βοηθό Import Peripheral

Από τα αρχεία που αναφέρονται στο σχήμα 5.25, και των οποίων το περιεχόμενο περιγράφεται αναλυτικά στο κεφάλαιο τέσσερα του παρόντος συγγράμματος, εμείς θα πρέπει να προσθέσουμε κάποιες δηλώσεις στο αρχείο opb_dio2_7seg.vhd, στο αρχείο opb_dio2_7seg_v2_1_0.mpd και κυρίως να επεξεργαστούμε το user_logic.vhd, που είναι και το αρχείο στο οποίο θα γίνει η περιγραφή του ελεγκτή μας. Το opb_dio2_7seg.vhd είναι το top-level αρχείο του περιφερειακού μας και συνεπώς θα πρέπει να δηλωθούν εκεί οι εξωτερικές θύρες αυτού. Οι θύρες που θα χρησιμοποιηθούν είναι οι εξής:

- Η θύρα led. Αυτή θα ελέγχει το LED της κάρτας Digilab 2. Θα ανάβει όταν έχει προγραμματιστεί σωστά το FPGA και όταν δεν είναι πατημένο το reset, δηλαδή όταν είναι πατημένος ο πιεστικός διακόπτης της Digilab 2

- Η θύρα oclk. Από αυτήν οδηγείται το ρολόι του CPLD, της κάρτας DIO2, που είναι 256 φορές πιο αργό από το ρολόι του επεξεργαστή
- Η θύρα cs. Είναι το σήμα επιλογής της κάρτας DIO2 και πρέπει να οδηγείται στο λογικό '1' για να λειτουργήσει η κάρτα
- Η θύρα we. Οδηγεί το σήμα εγγραφής του CPLD. Οι εγγραφές πραγματοποιούνται στην αρνητική ακμή του
- Η θύρα oe. Η επίτρεψη εξόδου. Όταν είναι στο λογικό '1' μπορούμε να διαβάσουμε από το CPLD
- Ο διάδρομος διευθύνσεων addr, των έξι bit. Καθορίζει την προσπέλαση των συσκευών της κάρτας DIO2
- Ο διάδρομος δεδομένων διπλής κατεύθυνσης data, των οκτώ bit. Λαμβάνει τα δεδομένα ανάγνωσης και αποστέλλει τα δεδομένα εγγραφής. Η συμβάσεις του EDK απαιτούν το σήμα αυτό να χειρίζεται στην VHDL σαν τρία σήματα. Το data_I των οκτώ bit είναι τα δεδομένα ανάγνωσης ενώ το data_O είναι αντίστοιχα τα δεδομένα εγγραφής. Το τρίτο σήμα είναι το data_T. Αυτό είτε είναι των οκτώ bit και καθορίζει εάν κάθε bit του διαδρόμου δεδομένων συμπεριφέρεται σαν είσοδος (τιμή λογικό '1') ή έξοδος (τιμή λογικό '0'), είτε είναι του ενός bit και καθορίζει το εάν όλα τα bit του διαδρόμου συμπεριφέρονται σαν είσοδος (τιμή λογικό '1') ή έξοδος (τιμή λογικό '0'). Εμείς χρησιμοποιήσαμε το data_T του ενός bit και το αναθέσαμε στο σήμα oe, αφού έχουν την ίδια συμπεριφορά.

Περισσότερα για τα σήματα και την λειτουργία της κάρτας DIO2 αναφέρονται στο κεφάλαιο τρία του παρόντος συγγράμματος.



Σχήμα 5.26 – Άνοιγμα του VHDL αρχείου

Αφού επιλέξουμε File → Open και περιηγηθούμε στον φάκελο που περιέχει το αρχείο opb_dio2_7seg.vhd, συμβουλευόμενοι και το σχήμα 5.25, μεταβαίνουμε στην γραμμή 121 αυτού του αρχείου. Εκεί προσθέτουμε τις παρακάτω δηλώσεις:

```
-- ADD USER PORTS BELOW THIS LINE ------------------
led       : out   std_logic;
oclk      : out   std_logic;
cs        : out   std_logic;
we        : out   std_logic;
oe        : out   std_logic;
addr       : out   std_logic_vector(0 to 5);
data_I    : in        std_logic_vector(0 to 7);
data_O    : out       std_logic_vector(0 to 7);
```

```
data_T   : out      std_logic;
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

Έπειτα στην γραμμή 379 του ίδιου αρχείου προσθέτουμε τις παρακάτω δηλώσεις:

```
-- MAP USER PORTS BELOW THIS LINE ------------------
led       => led,
oclk      => oclk,
cs        => cs,
we        => we,
oe        => oe,
addr      => addr,
data_I    => data_I,
data_O    => data_O,
data_T    => data_T,
-- MAP USER PORTS ABOVE THIS LINE ------------------
```

Με την ολοκλήρωση αυτών των αλλαγών αυτών σώζουμε και κλείνουμε το αρχείο. Όπως γίνεται κατανοητό οι παραπάνω θύρες αντιστοιχούν σε κάποιες ομώνυμες κάποιας άλλης οντότητας. Αυτή η οντότητα ορίζεται στο αρχείο user_logic.vhd. Ανοίγουμε το αρχείο αυτό και στην γραμμή 100 προσθέτουμε τις ανάλογες δηλώσεις:

```
-- ADD USER PORTS BELOW THIS LINE ------------------
led       : out   std_logic;
oclk      : out   std_logic;
cs        : out   std_logic;
we        : out   std_logic;
oe        : out   std_logic;
addr      : out   std_logic_vector(0 to 5);
data_I    : in    std_logic_vector(0 to 7);
data_O    : out   std_logic_vector(0 to 7);
data_T    : out   std_logic;
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

Στο τέλος του παραπάνω αρχείου υπάρχει το τμήμα architecture. Εκεί αντιγράφουμε τον κώδικα που ακολουθεί:

```
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

        ---------------------------------------------------------------------------
        -- User logic signals and S/W accessible registers
        ---------------------------------------------------------------------------
        signal counter              : natural range 0 to 255;     --driver for the generation of oclk
        signal reg_oclk             : std_logic;
        signal reg_cs               : std_logic;
        signal reg_addr             : std_logic_vector(0 to 5);
        signal reg_data             : std_logic_vector(0 to 7);
        signal reg_we               : std_logic;
        signal reg_oe               : std_logic;
        signal phase                : natural range 0 to 31;      --phases for the polling procedure
        signal sevsegments          : std_logic_vector(0 to 15);  --the digits for appearance
        signal reg_write_select     : std_logic_vector(0 to 0);
        signal reg_read_select      : std_logic_vector(0 to 0);

begin
```

```
	-----------------------------------------------------------------------
	-- Map user logic S/W register read/write select signal
	-----------------------------------------------------------------------
	reg_write_select <= Bus2IP_WrCE(0 to 0);
	reg_read_select  <= Bus2IP_RdCE(0 to 0);

	-----------------------------------------------------------------------
	-- User logic S/W accessible registers write action
	-----------------------------------------------------------------------
	REG_WRITE_PROC : process( Bus2IP_Clk ) is
	begin

	  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
	    if Bus2IP_Reset = '1' then
	      sevsegments <= "0001100110000010";   --reset value 1982
	    else
	      case reg_write_select is
	        when "1" =>
	          for byte_index in 0 to (C_DWIDTH/8)-1 loop
	            if ( Bus2IP_BE(byte_index) = '1' ) then
	              sevsegments(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
	            end if;
	          end loop;
	            when others => null;
	          end case;
	    end if;
	  end if;

	end process REG_WRITE_PROC;

	-----------------------------------------------------------------------
	-- User logic S/W accessible registers read action
	-----------------------------------------------------------------------
	REG_READ_PROC : process( reg_read_select, sevsegments ) is
	begin

		case reg_read_select is
			when "1" => IP2Bus_Data <= sevsegments;
			when others => IP2Bus_Data <= (others => '0');
		end case;

	end process REG_READ_PROC;

	-----------------------------------------------------------------------
	-- Counter for DIO2's clock
	-----------------------------------------------------------------------
	COUNTER_PROC : process( Bus2IP_Clk ) is
	begin
		if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
			counter <= counter + 1;
		end if;
	end process COUNTER_PROC;

	-----------------------------------------------------------------------
	-- DIO2's clock generation
	-----------------------------------------------------------------------
	DIO2_CLOCK_PROC : process( Bus2IP_Clk ) is
	begin
		if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
			if counter = 0 then
				reg_oclk <= '0';
```

```
                        elsif counter = 128 then
                                reg_oclk <= '1';
                        end if;
                end if;
        end process DIO2_CLOCK_PROC;

        ------------------------------------------------------------------------
        -- Activation of DIO2 and creation of phases for polling
        ------------------------------------------------------------------------
        reg_cs          <= '1';
        phase           <= counter / 8;
        ------------------------------------------------------------------------
        -- Sevensegments' display
        ------------------------------------------------------------------------
        DIO2_7SEG_PROC : process( Bus2IP_Clk ) is

        begin

          if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
            if Bus2IP_Reset = '1' then
                reg_addr<= (others => '0');
                reg_data <= (others => 'Z');
                reg_we          <= '0';
                reg_oe          <= '0';
            else
             case phase is
             -- Write seven segments digits 2 and 1
             when 24 => reg_oe <= '0'; reg_we <= '0'; reg_addr <= "000110"; reg_data <= sevsegments(4 to 7) &
sevsegments(0 to 3)          ;
              when 25 => reg_oe <= '0'; reg_we <= '1';
             when 26 => reg_oe <= '0'; reg_we <= '1';
             when 27 => reg_oe <= '0'; reg_we <= '0';
             -- Write seven segments digits 4 and 3
             when 28 => reg_oe <= '0'; reg_we <= '0'; reg_addr <= "000111"; reg_data <= sevsegments(12 to 15) &
sevsegments(8 to 11);
             when 29 => reg_oe <= '0'; reg_we <= '1';
             when 30 => reg_oe <= '0'; reg_we <= '1';
             when 31 => reg_oe <= '0'; reg_we <= '0';
             when others  => null;
            end case;
           end if;
          end if;
        end process DIO2_7SEG_PROC;

        ------------------------------------------------------------------------
        -- Code to drive IP to Bus signals
        ------------------------------------------------------------------------
        led                             <= not Bus2IP_Reset;
        oclk                    <= reg_oclk;
        cs                              <= reg_cs;
        we                              <= reg_we;
        oe                              <= reg_oe;
        addr                    <= reg_addr;
        data_O                  <= reg_data;
        data_T                  <= reg_oe;
        IP2Bus_Ack      <= Bus2IP_WrCE(0) or Bus2IP_RdCE(0); --acknowledge when read or write
        IP2Bus_Error    <= '0';
        IP2Bus_Retry    <= '0';
        IP2Bus_ToutSup <= '0';


end IMP;
```

Το κομμάτι της εγγραφής στην οθόνη των επτά τμημάτων, μέσω της διαδικασίας DIO2_7SEG_PROC, είναι εμπνευσμένο από κομμάτι VHDL που χειρίζεται όλες τις συσκευές με polling, διαδοχική ανάγνωση και εγγραφή, και για τον λόγο αυτό το ρολόι του CPLD έχει χωριστεί σε 32 φάσεις όπου οι 24 έως 31 ανήκουν στην οθόνη επτά τμημάτων. Όταν γίνεται εγγραφή στον 16-bit καταχωρητή τότε ο επεξεργαστής στέλνει ένα σήμα, Bus2IP_WrCE, και ο διάδρομος δεδομένων Bus2IP_Data περνάει τα δεδομένα που στέλνει ο επεξεργαστής στον καταχωρητή sevsegments. Όλα αυτά συμβαίνουν στην διαδικασία REG_WRITE_PROC, ενώ αντίστοιχα εκτελείται και η ανάγνωση του καταχωρητή στην REG_READ_PROC. Βέβαια μέχρι στιγμής δεν έχουμε γράψει κάποια εφαρμογή λογισμικού που να ελέγχει τον συγκεκριμένο καταχωρητή και συνεπώς αυτό που θα απεικονίζει η οθόνη επτά τμημάτων εάν κατεβάσουμε το σύστημα στο FPGA θα είναι η τιμή μηδενισμού / αρχικοποίησης 1982, που έχει αυτήν την τυχαία τιμή για να μην συγχέεται με την μηδενική κατάσταση στην οποία μπορεί να περιέλθει η οθόνη επτά τμημάτων μετά την τροφοδοσία της.

Οι αλλαγές στο user_logic.vhd έχουν ολοκληρωθεί και μπορούμε να σώσουμε και να κλείσουμε το αρχείο. Όμως το περιφερειακό μας δεν είναι ακόμα έτοιμο για να συμπεριληφθεί στο σύστημα. Θα πρέπει θύρες του περιφερειακού να δηλωθούν και στο αρχείο opb_dio2_7seg_v2_1_0.mpd. Ανοίγουμε το αρχείο αυτό και στο κομμάτι όπου βρίσκονται οι δηλώσεις PORT προσθέτουμε τις παρακάτω γραμμές:

```
## Ports
PORT led = "", DIR = O
PORT oclk = "", DIR = O
PORT cs = "", DIR = O
PORT we = "", DIR = O
PORT oe = "", DIR = O
PORT addr = "", DIR = O, VEC = [0:5]
PORT data = "", DIR = INOUT, VEC = [0:7], ENABLE = SINGLE, THREE_STATE = TRUE
```

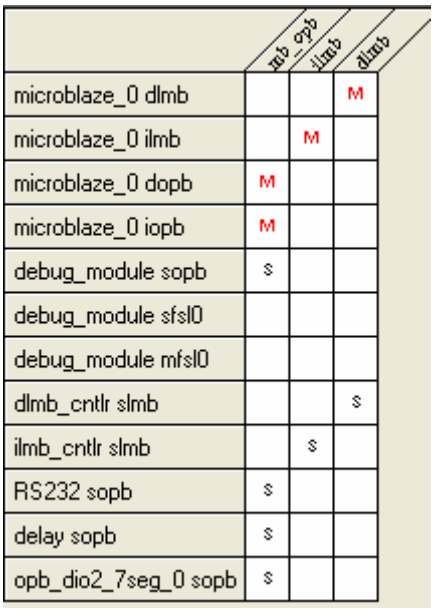
Παρατηρήστε ότι στην δήλωση της data γνωστοποιείται ότι πρέπει να χρησιμοποιηθεί τρικατάστατος καταχωρητής που θα έχει ένα bit επιλογής κατεύθυνσης.

Αφού σώσουμε και κλείσουμε το παραπάνω αρχείο, κλείνουμε όλο το project του XPS και το ανοίγουμε πάλι, προκειμένου το XPS να ανιχνεύσει την δημιουργία ενός νέου περιφερειακού. Το νέο μας περιφερειακό είναι έτοιμο να προστεθεί στο σύστημα σαν ένα προϋπάρχον IP. Επιλέγουμε Project → Add/Edit Cores. Στην καρτέλα Peripherals βρίσκουμε και προσθέτουμε το opb_dio2_7seg και ορίζουμε το πεδίο διευθύνσεων του όπως στο σχήμα 5.27.

Peripheral	HW Ver	Instance	Base Address	High Address	Min
microblaze	2.10.a	microblaze_0			
opb_mdm	2.00.a	debug_module	0x80002000	0x800020ff	0x100
lmb_bram_i...	1.00.b	dlmb_cntlr	0x00000000	0x00000fff	0x800
lmb_bram_i...	1.00.b	ilmb_cntlr	0x00000000	0x00000fff	0x800
bram_block	1.00.a	lmb_bram			
opb_uartlite	1.00.b	RS232	0x80002100	0x800021ff	0x100
opb_timer	1.00.b	delay	0x80002200	0x800022ff	0x100
opb_dio2_7...	1.00.a	opb_dio2_7s...	0x80002300	0x800023ff	0x100

Σχήμα 5.27 – Αλλαγή διευθύνσεων του περιφερειακού

Στην καρτέλα Bus Connections συνδέουμε το περιφερειακό στον διάδρομο δεδομένων OPB.



Σχήμα 5.28 – Σύνδεση του στοιχείου στον OPB

Στην επόμενη καρτέλα Ports προσθέτουμε τις θύρες led, oclk, cs, we, oe, addr, data, OPB_CLK στο σύστημα. Ορίζουμε το εύρος των addr και data και μετατρέπουμε το OPB_Clk σε εσωτερική θύρα (Internal). Όλες οι παραπάνω ενέργειες φαίνονται στο σχήμα 5.29.

Instance	Port Name	Net Name	Pola...	Scope	Range	Class
opb_dio2_...	led	led	O	External		
opb_dio2_...	oclk	oclk	O	External		
opb_dio2_...	cs	cs	O	External		
opb_dio2_...	we	we	O	External		
opb_dio2_...	oe	oe	O	External		
opb_dio2_...	addr	addr	O	External	[0:5]	
opb_dio2_...	data	data	IO	External	[0:7]	
opb_dio2_...	OPB_Clk	sys_clk_s	I	Internal		Clk

Σχήμα 5.29 – Οι θύρες του νέου περιφερειακού

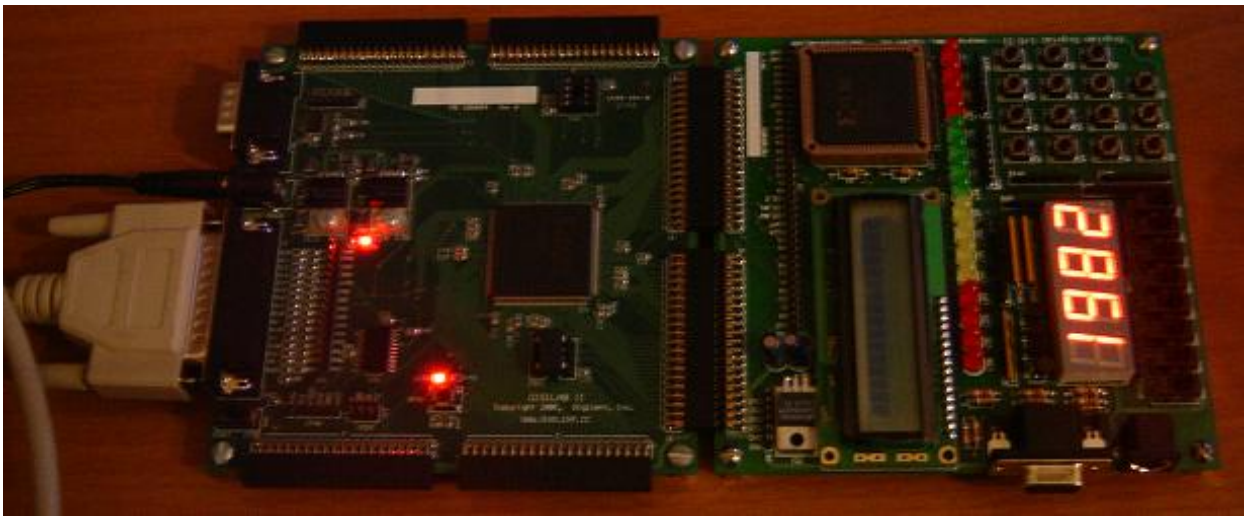
Κλείνουμε το παράθυρο διαλόγου πατώντας OK. Το περιφερειακό έχει συνδεθεί στο σύστημα, όμως από την στιγμή που διαθέτει εξωτερικές θύρες, θα πρέπει αυτές να συνδεθούν με συγκεκριμένους ακροδέκτες του FPGA. Οι κατάλληλοι ακροδέκτες μπορούν εύκολα να προκύψουν από την ανάλυση των καρτών Digilab2 και DIO2 στο κεφάλαιο τρία του παρόντος συγγράμματος. Η αντιστοίχιση των σημάτων με τους ακροδέκτες θα γίνει στο αρχείο UCF. Στην καρτέλα System του XPS, στο υπόδεντρο Project Files, κάνουμε διπλό-κλικ στο αρχείο system.ucf, ώστε αυτό να ανοίξει στο κυρίως παράθυρο του XPS. Στο τέλος του αρχείου προσθέτουμε τις παρακάτω γραμμές:

```
Net led        LOC=p71;
Net oclk       LOC=p173;
Net cs         LOC=p166;
Net we         LOC=p165;
Net oe         LOC=p168;

Net addr<0>    LOC=p180;
Net addr<1>    LOC=p176;
```

```
Net addr<2>     LOC=p178;
Net addr<3>     LOC=p174;
Net addr<4>     LOC=p175;
Net addr<5>     LOC=p172;

Net data<0>      LOC=p163;
Net data<1>     LOC=p164;
Net data<2>     LOC=p161;
Net data<3>     LOC=p162;
Net data<4>     LOC=p154;
Net data<5>     LOC=p160;
Net data<6>     LOC=p151;
Net data<7>     LOC=p152;
```

Σώζουμε και κλείνουμε το αρχείο.

Στο προηγούμενο βήμα, της εφαρμογής, είχαμε υλοποιήσει το σύστημα στο Project Navigator του ISE, τώρα όμως πηγαίνουμε στο παράθυρο διαλόγου Project Options και επιλέγουμε ξανά σαν εργαλείο σύνθεσης το XPS. Επιλέγουμε Tools → Generate Bitstream, το σύστημα υλοποιείται και δημιουργείται το αρχείο BIT. Για να κατεβάσουμε το σύστημα επιλέγουμε Tools → Download και πράγματι η τιμή μηδενισμού / αρχικοποίησης, 1982, εμφανίζεται στην οθόνη επτά τμημάτων. Παρόλα αυτά δεν μπορούμε να αλλάξουμε την τιμή αυτή, αφού δεν έχουμε δημιουργήσει μία αντίστοιχη εφαρμογή λογισμικού, κάτι που θα πραγματοποιηθεί στο επόμενο βήμα.



Σχήμα 5.30 – Το κατέβασμα του συστήματος και η εμφάνιση στην οθόνη επτά τμημάτων

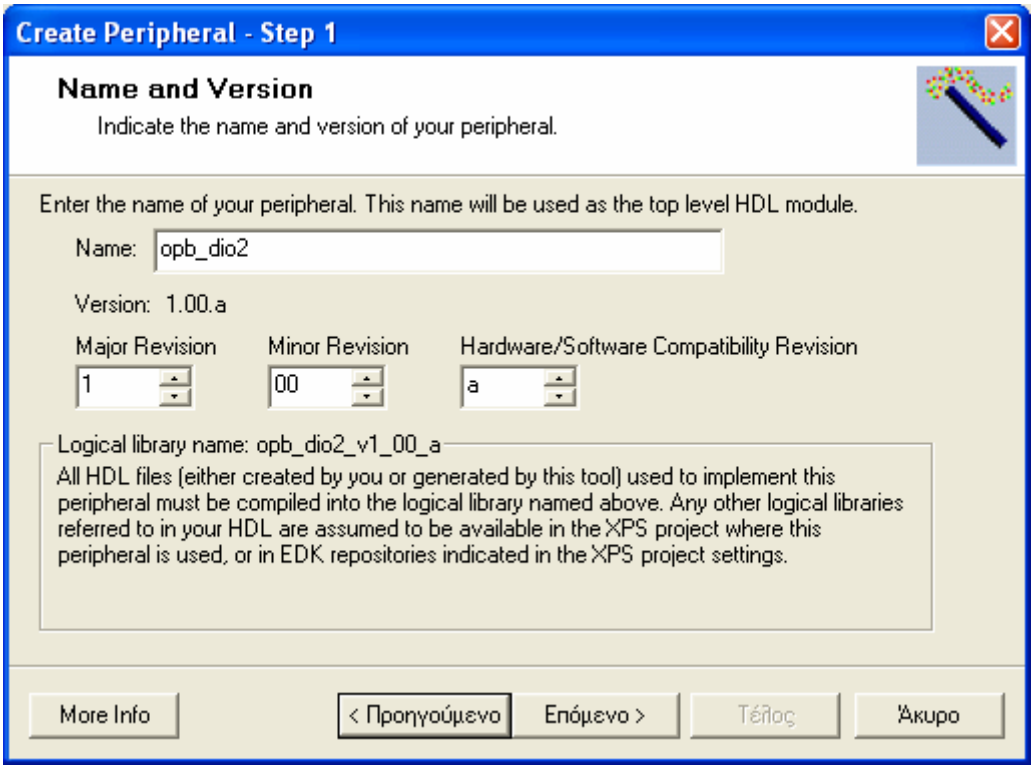
5.2.4 Βήμα 4ο : Συγγραφή απλής εφαρμογής λογισμικού

Στο τέταρτο βήμα της πρώτης εφαρμογής, θα επεκτείνουμε την χρήση της κάρτας DIO2 δημιουργώντας ένα νέο περιφερειακό, που θα ελέγχει εκτός από την οθόνη επτά τμημάτων τόσο τα LEDs όσο και τους διακόπτες, και θα συγγράψουμε μία εφαρμογή λογισμικού που θα χειρίζεται τα LEDs και την οθόνη επτά τμημάτων με βάση τις τιμές των διακοπτών.

Σε αυτό λοιπόν το βήμα θα ασχοληθούμε και με το λογισμικό του συστήματος, αλλά πριν από αυτό θα πρέπει να επεκτείνουμε την περιγραφή του ελεγκτή της κάρτας DIO2, ώστε να συμπεριλάβει τους διακόπτες και τα LEDs. Για τον λόγο αυτό επιλέγουμε Project → Add/Edit Cores ώστε να εμφανιστεί το παράθυρο διαλόγου Add/Edit Hardware Platform Specifications

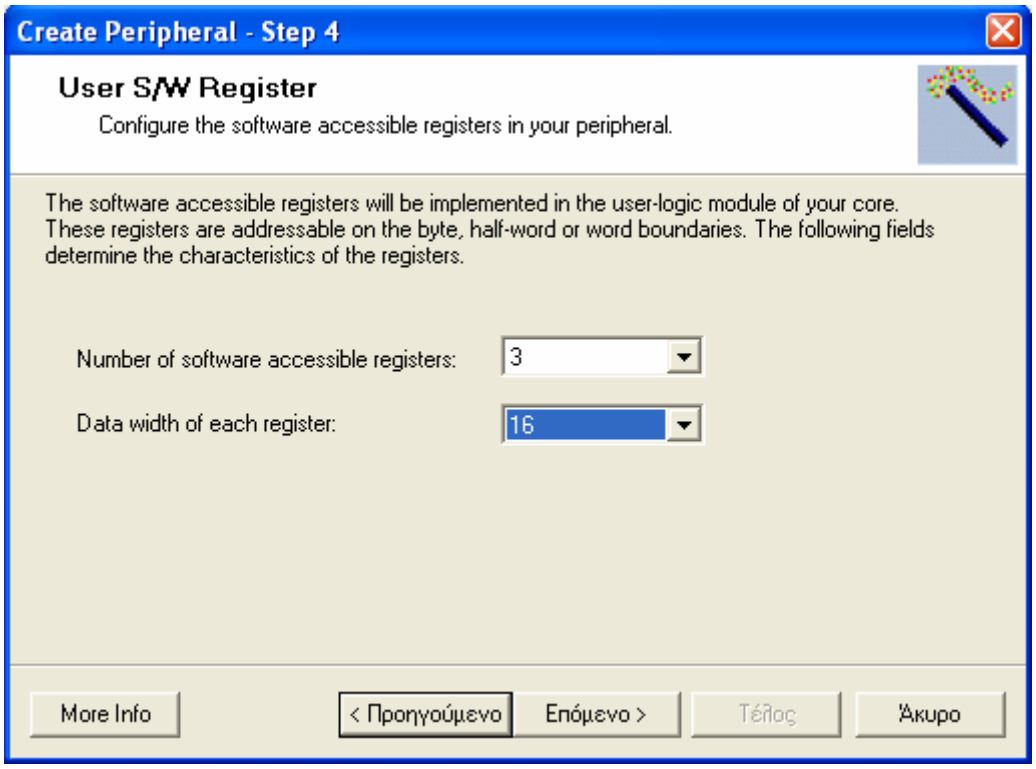
όπου θα διαγράψουμε το παλαιό περιφερειακό μας από το σύστημα. Στην καρτέλα Ports επιλέγουμε όλες τις θύρες του opb_dio2_7seg και πατάμε Del για να διαγραφούν οι θύρες αυτές. Στην καρτέλα Bus Connections αφαιρούμε την σύνδεση του opb_dio2_7seg με τον διάδρομο δεδομένων OPB και τέλος στην καρτέλα Peripherals επιλέγουμε το περιφερειακό και πατάμε Delete. Πατάμε OK για να κλείσουμε το παράθυρο διαλόγου και πλέον το περιφερειακό opb_dio2_7seg δεν υπάρχει στο σύστημα όπως μπορούμε να διαπιστώσουμε και από το υπόδεντρο System BSP στο αριστερό παράθυρο του XPS, όπου δεν γίνεται πλέον αναφορά στο περιφερειακό μας.

Κατά τα γνωστά επιλέγουμε Tools → Import Peripheral Wizard για να εισάγουμε το νέο, βελτιωμένο, περιφερειακό για την κάρτα DIO2. Στα πρώτα παράθυρα διαλόγου κάνουμε τις ίδιες επιλογές με αυτές του προηγούμενου βήματος (σελ. 127-128), μέχρι που φτάνουμε στην ονομασία του περιφερειακού όπου χρησιμοποιούμε το πιο γενικό όνομα opb_dio2, όπως φαίνεται στο σχήμα 5.31.



Σχήμα 5.31 – Η ονομασία του νέου περιφερειακού

Στην συνέχεια επιλέγουμε τη σύνδεση του περιφερειακού μας με τον διάδρομο δεδομένων OPB και από τις υπηρεσίες της διεπαφής IPIF διαλέγουμε μόνο την χρήση καταχωρητών ελεγχόμενων από το λογισμικό. Η μόνη διαφορά της υπηρεσίας IPIF σε σχέση με το παλιό περιφερειακό θα είναι ότι θα χρησιμοποιήσουμε τρεις καταχωρητές, αντί για έναν, μεγέθους και πάλι 16 bit. Οι δύο επιπλέον καταχωρητές θα χρειαστούν για την μεταφορά των δεδομένων των LEDs και των διακοπτών.



Σχήμα 5.32 – Επιλογές των καταχωρητών του νέου περιφερειακού

Η συνέχεια είναι η ίδια με αυτήν του προηγούμενου βήματος (σελ. 130) και κατά την έξοδό μας από τον βοηθό αρνούμαστε να επανεκκινήσουμε το project, αφού κάτι τέτοιο θα κάνουμε αφού γράψουμε τα VHDL κομμάτια του περιφερειακού μας.

Αφού περιηγηθούμε στους φακέλους με τα αρχεία του νέου μας περιφερειακού ανοίγουμε το αρχείο opb_dio2.vhd. Οι θύρες του νέου περιφερειακού παραμένουν οι ίδιες με αυτές του προηγούμενου και τις προσθέτουμε στα δύο γνωστά σημεία του αρχείου opb_dio2.vhd, όπως κάναμε και στο προηγούμενο βήμα (σελ. 131-132). Αφού σώσουμε και κλείσουμε το παραπάνω αρχείο, ανοίγουμε το user_logic.vhd και προσθέτουμε και εδώ τις θύρες του περιφερειακού στο γνωστό σημείο (σελ. 132). Στην συνέχεια αντικαθιστούμε το κομμάτι architecture με τον παρακάτω κώδικα:

```
---------------------------------------------------------------------------
-- Architecture section
---------------------------------------------------------------------------

architecture IMP of user_logic is

  ---------------------------------------------------------------
  -- User signal declarations and signals for user logic
  ---------------------------------------------------------------
  signal sevsegments     : std_logic_vector(0 to 15);
  signal leds                    : std_logic_vector(0 to 15);          --the leds to be appeared
  signal switches          : std_logic_vector(0 to C_DWIDTH-1); --the values of the switches
  signal slv_reg_write_select    : std_logic_vector(0 to 2);
  signal slv_reg_read_select     : std_logic_vector(0 to 2);
  signal slv_ip2bus_data         : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_read_ack    : std_logic;          --the acknowledge of the read actions of the IP
  signal slv_write_ack   : std_logic;         --the acknowledge of the write actions of the IP
  signal reg_oe          : std_logic;
  signal phase           : natural range 0 to 31;  --phases for the polling procedure
  signal counter                 : natural range 0 to 255; --driver for the generation of oclk
```

```
begin

  ---------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic, you
  -- are basically operating on the following memory mapped registers:
  --
  --    Bus2IP_WrCE or   Memory Mapped
  --      Bus2IP_RdCE   Register
  --           "1000"   C_BASEADDR + 0x0
  --           "0100"   C_BASEADDR + 0x4
  --           "0010"   C_BASEADDR + 0x8
  --           "0001"   C_BASEADDR + 0xC
  --
  ---------------------------------------
  ---------------------------------------------------------------------
  --Map user logic S/W register read/write select signal
  ---------------------------------------------------------------------
  slv_reg_write_select <= Bus2IP_WrCE(0 to 2);
  slv_reg_read_select  <= Bus2IP_RdCE(0 to 2);

  ---------------------------------------
  --Acknowledge signals
  ---------------------------------------
  slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
  slv_read_ack         <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);

  ---------------------------------------------------------------------------------------------
  --User logic S/W accessible registers write action (only sevsegments and leds)
  ---------------------------------------------------------------------------------------------
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        sevsegments <= (others => '0');
        leds <= (others => '0');
      else
        case slv_reg_write_select is
          when "100" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                sevsegments(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when "010" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                leds(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when others => null;
        end case;
      end if;
    end if;
  end process SLAVE_REG_WRITE_PROC;
```

```
------------------------------------------------------------------------------
-- User logic S/W accessible registers read action (only switches)
------------------------------------------------------------------------------
SLAVE_REG_READ_PROC : process( slv_reg_read_select, switches ) is
begin

      case slv_reg_read_select is
            when "001" => slv_ip2bus_data <= switches;
            when others => slv_ip2bus_data <= (others => '0');
      end case;

end process SLAVE_REG_READ_PROC;

------------------------------------------------------------------------------
-- Counter for DIO2's clock
------------------------------------------------------------------------------
COUNTER_PROC : process( Bus2IP_Clk ) is
begin
      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
            counter <= counter + 1;
      end if;
end process COUNTER_PROC;

------------------------------------------------------------------------------
-- DIO2's clock generation
------------------------------------------------------------------------------
DIO2_CLOCK_PROC : process( Bus2IP_Clk ) is
begin
      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
            if counter = 0 then
                  oclk <= '0';
            elsif counter = 128 then
                  oclk <= '1';
            end if;
      end if;
end process DIO2_CLOCK_PROC;

------------------------------------------------------------------------------
-- Activation of DIO2 and creation of phases for polling
------------------------------------------------------------------------------
cs          <= '1';
phase       <= counter / 8;

------------------------------------------------------------------------------
-- DIO2's functions/actions
------------------------------------------------------------------------------
DIO2_7SEG_PROC : process( Bus2IP_Clk ) is

------------------------------------------------------------------------------
--Function that reverses the order of bits in a logic_vector
------------------------------------------------------------------------------
function fliplr(x : in std_logic_vector) return std_logic_vector is
    variable y : std_logic_vector(x'range);
 begin
    for i in x'range loop
       y(i) := x(x'right -i + x'left);
    end loop;
    return y;
 end fliplr;

begin
```

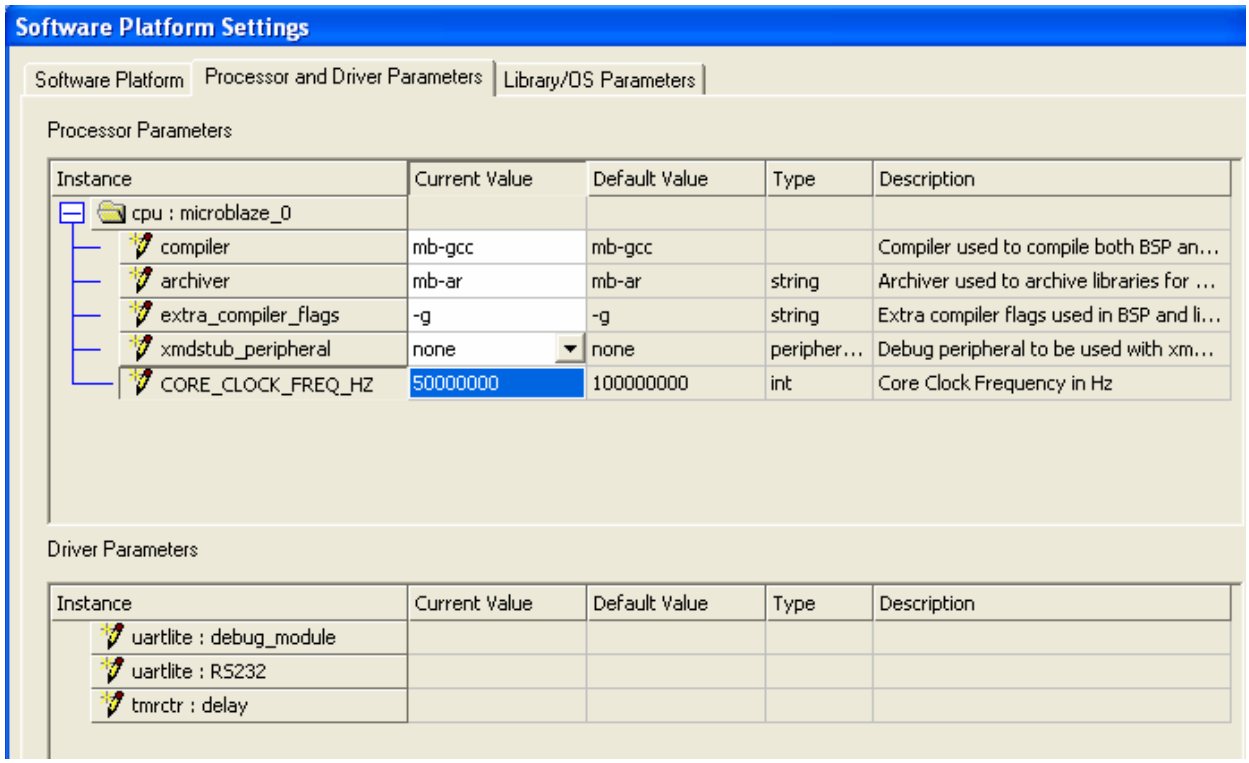
```
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        addr      <= (others => '0');
        data_O    <= (others => 'Z');
        we                <= '0';
        reg_oe            <= '0';
      else
     case phase is
        -- Read sw0 to sw7
        when  8 => reg_oe <= '0'; we <= '0'; addr <= "000010";
        when  9 => reg_oe <= '1'; we <= '0'; switches(8 to 15) <= data_I;
        when 10 => reg_oe <= '1'; we <= '0';
        when 11 => reg_oe <= '0'; we <= '0';
        -- Idle time
        when 12 => reg_oe <= '0'; we <= '0';
        when 13 => reg_oe <= '0'; we <= '0';
        when 14 => reg_oe <= '0'; we <= '0';
        when 15 => reg_oe <= '0'; we <= '0';
        -- Write ld0 to ld7
        when 16 => reg_oe <= '0'; we <= '0'; addr <= "000100";
        when 17 => reg_oe <= '0'; we <= '1'; data_O <= fliplr(leds(0 to 7));
        when 18 => reg_oe <= '0'; we <= '1';
        when 19 => reg_oe <= '0'; we <= '0';
        -- Write ld8 to ldF
        when 20 => reg_oe <= '0'; we <= '0'; addr <= "000101";
        when 21 => reg_oe <= '0'; we <= '1'; data_O <= fliplr(leds(8 to 15));
        when 22 => reg_oe <= '0'; we <= '1';
        when 23 => reg_oe <= '0'; we <= '0';
        -- Write seven segments digits 2 and 1
        when 24 => reg_oe <= '0'; we <= '0'; addr <= "000110";
        when 25 => reg_oe <= '0'; we <= '1'; data_O <= sevsegments(4 to 7) & sevsegments(0 to 3);
        when 26 => reg_oe <= '0'; we <= '1';
        when 27 => reg_oe <= '0'; we <= '0';
        -- Write seven segments digits 4 and 3
        when 28 => reg_oe <= '0'; we <= '0'; addr <= "000111";
        when 29 => reg_oe <= '0'; we <= '1'; data_O <= sevsegments(12 to 15) & sevsegments(8 to 11);
        when 30 => reg_oe <= '0'; we <= '1';
        when 31 => reg_oe <= '0'; we <= '0';
        when others  => null;
          end case;
      end if;
    end if;
  end process DIO2_7SEG_PROC;

  ------------------------------------------------
  -- Code to drive IP to Bus signals
  ------------------------------------------------
  led                                 <= not Bus2IP_Reset;
  oe                                  <= reg_oe;
  data_T                              <= reg_oe;
  IP2Bus_Data    <= slv_ip2bus_data;

  IP2Bus_Ack         <= slv_write_ack or slv_read_ack;
  IP2Bus_Error       <= '0';
  IP2Bus_Retry       <= '0';
  IP2Bus_ToutSup     <= '0';

end IMP;
```

Όπως φαίνεται και παραπάνω ο κώδικας είναι λίγο πιο προσεκτικά γραμμένος, τα σήματα των θυρών, εκτός του oe λόγω του ότι οδηγεί και το data_T, αναθέτονται χωρίς την μεσολάβηση προσωρινών καταχωρητών και στην διαδικασία εγγραφής στους καταχωρητές

συμμετέχουν μόνο τα δεδομένα της οθόνης επτά τμημάτων και των LEDs, ενώ στην αντίστοιχη της ανάγνωσης συμμετέχει μόνο ο καταχωρητής με τα δεδομένα των διακοπτών. Η επικοινωνία με τις συσκευές της κάρτας DIO2 γίνεται με polling, όπως είχε αναφερθεί και στο τρίτο βήμα, και εδώ έχουμε περισσότερες φάσεις ενεργές, και τέσσερις άεργες φάσεις για τον διαχωρισμό μεταξύ αναγνώσεων και εγγραφών. Σώζουμε και κλείνουμε το αρχείο.

Αυτό που απομένει είναι η προσθήκη των θυρών του περιφερειακού μας στο αρχείο opb_dio2_v2_1_0.mpd και η σύνδεση του περιφερειακού στο σύστημα. Η εγγραφή στο αρχείο είναι πανομοιότυπη με αυτήν του τρίτου βήματος (σελ. 135), ενώ για την σύνδεση του περιφερειακού ακολουθείται ακριβώς η ίδια διαδικασία με την βοήθεια του παραθύρου διαλόγου Add/Edit Hardware Platform Specifications που περιγράφεται στις σελίδες 135-136 του τρίτου βήματος. Το αρχείο UCF δεν χρειάζεται επεξεργασία, καθώς οι αντιστοιχίσεις των σημάτων των εξωτερικών θυρών με τους ακροδέκτες του FPGA παραμένουν οι ίδιες.

Πλέον από την πλευρά του υλικού το περιφερειακό μας είναι έτοιμο να λειτουργήσει. Αυτό όμως που λείπει είναι μία εφαρμογή λογισμικού για να ελέγξει το περιφερειακό. Πριν από την δημιουργία της εφαρμογής, θα πρέπει να εισαχθούν οι κατάλληλες ρυθμίσεις για την πλατφόρμα του λογισμικού. Επιλέγουμε Project → Software Platform Settings και εμφανίζεται το ομώνυμο παράθυρο διαλόγου. Στην καρτέλα Software Platform διατηρούμε τις προεπιλεγμένες τιμές για τους οδηγούς των συσκευών και το λειτουργικό σύστημα (δεν χρησιμοποιούμε). Στην καρτέλα Processor and Driver Parameters αλλάζουμε την συχνότητα λειτουργίας του MicroBlaze στα 50 MHz όπως στο σχήμα 5.33.



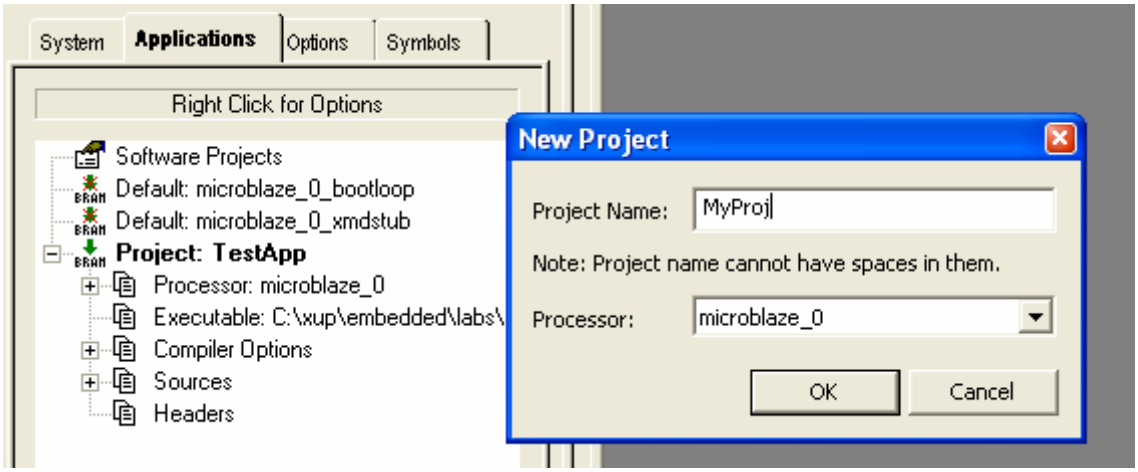
Σχήμα 5.33 – Οι παράμετροι για τον επεξεργαστή MicroBlaze

Στην τελευταία καρτέλα, Library/OS Parameters, δεν ορίζουμε καμία πρότυπη είσοδο ή έξοδο, αφού όπως είδαμε παρότι έχουμε προσθέσει την σειριακή θύρα αυτή δεν είναι σε θέση να λειτουργήσει σωστά , και αφήνουμε την τιμή του πεδίου need_xil_malloc false, αφού στην εφαρμογή που θα υλοποιήσουμε δεν θα χρειαστούμε συνάρτηση κατανομής μνήμης. Πατάμε OK για να αποθηκεύσουμε τις ρυθμίσεις. Η ενέργειες που ακολουθήσαμε στο προηγούμενο παράθυρο διαλόγου, αντικατοπτρίζονται στο αρχείο system.mss, αφού αν το ανοίξουμε θα δούμε στην δομή του όλα τα χαρακτηριστικά για την πλατφόρμα λογισμικού τα οποία ορίσαμε.

Προκειμένου να δημιουργήσουμε την βάση για την λειτουργία της πλατφόρμας λογισμικού επιλέγουμε Tools → Generate Libraries. Με τον τρόπο αυτό το εργαλείο LibGen δεχόμενο ως κύρια είσοδο το αρχείο system.mss, δημιουργεί μία δομή τεσσάρων φακέλων, μέσα σε έναν άλλο φάκελο microblaze_0, που περιέχουν τα εξής:

- Φάκελος code. Εκεί θα τοποθετηθεί το εκτελέσιμο αρχείο που θα παραχθεί παρακάτω
- Φάκελος include. Εδώ περιέχονται τα αρχεία επικεφαλίδων που αφορούν τα περιφερειακά και τις συναρτήσεις που δύνανται να χρησιμοποιηθούν στις εφαρμογές
- Φάκελος lib. Περιέχει τις βιβλιοθήκες του συστήματος
- Φάκελος libsrc. Εδώ περιέχονται τα αρχεία επικεφαλίδων με τις συναρτήσεις για τους οδηγούς των περιφερειακών του συστήματος

Έχοντας δημιουργήσει τα αρχεία υποστήριξης για την πλατφόρμα λογισμικού, μπορούμε να προχωρήσουμε στην δημιουργία μιας εφαρμογής. Στο παράθυρο του XPS που βρίσκεται αριστερά, στην καρτέλα Applications, κάνουμε διπλό-κλικ στο Software Projects. Στο παράθυρο που εμφανίζεται τοποθετούμε το όνομα, MyProj, της εφαρμογής μας και πατάμε OK.



Σχήμα 5.34 – Δημιουργία νέας εφαρμογής

Αμέσως στην καρτέλα Applications εμφανίζεται ένα υπόδεντρο με το όνομα της εφαρμογής μας. Για να γράψουμε τον πηγαίο κώδικα της εφαρμογής μας επιλέγουμε από το μενού του XPS File → New και ένα νέο έγγραφο προς επεξεργασία ανοίγει στο κυρίως παράθυρο του XPS. Εκεί γράφουμε τον παρακάτω κώδικα:

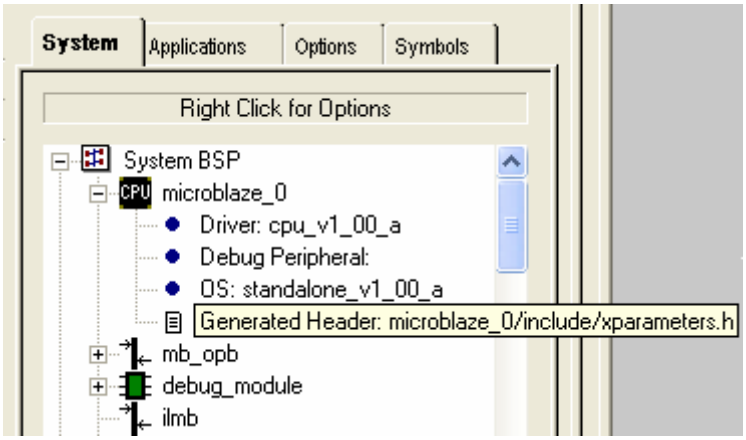
```
#include "xbasic_types.h"
#include "xparameters.h"

main()
{
Xuint32 i=0;   //counter for infinite loop
Xuint16 j=0;   //the value written on the seven segments
Xuint16 k=0;  //the value from the switches that is written on the leds
Xuint16  *ledptr;  //a pointer to the base address of our peripheral, for data of 16 bits

    ledptr = (Xuint16 *)XPAR_OPB_DIO2_0_BASEADDR; //the address of the pointer
    k = *(ledptr + 2);  //read the switches
    *(ledptr + 1) = k;  //write on the leds
    *ledptr = 0;          //write on seven segments
```

```
    while (1)
        {
        j = j + k;
        k = *(ledptr + 2);
        *ledptr = j;
        *(ledptr + 1) = k;
        for (i=0; i<1000000; i++);  //delay for display
        }
}
```

Ο παραπάνω απλός κώδικας αποτελείται από κάποια βασικά σημεία. Το περιφερειακό που δημιουργήσαμε χειρίζεται με την διεύθυνση βάσης του. Με την βοήθεια αυτής γράφονται και διαβάζονται οι τρεις καταχωρητές. Η διαδικασία ελέγχου των επεξεργαστών είναι γραμμένη και στον κώδικα VHDL που αφορούσε το περιφερειακό μας και είναι η εξής:

- Ο επεξεργαστής χειρίζεται τόσα σήματα ελέγχου όσοι και οι καταχωρητές, δηλαδή στην περίπτωσή μας τρία
- Όταν στην εφαρμογή λογισμικού γίνεται μία εγγραφή / ανάγνωση με διεύθυνση την διεύθυνση βάσης του περιφερειακού, τότε ελέγχεται ο πρώτος καταχωρητής (η αντίστοιχη έξοδος των σημάτων ελέγχου είναι ‘100’)
- Όταν θέλουμε να ελέγξουμε τον δεύτερο καταχωρητή επιχειρούμε εγγραφή / ανάγνωση στην επόμενη διεύθυνση *(pointer + 1), που εξαρτάται από το μέγεθος των καταχωρητών, και που στην περίπτωσή μας, αφού έχουμε 16-bit καταχωρητές, είναι η [Διεύθυνση Βάσης + 0x2] (τα σήματα ελέγχου είναι ‘010’)
- Ομοίως ισχύει για τους υπόλοιπους σε σειρά καταχωρητές

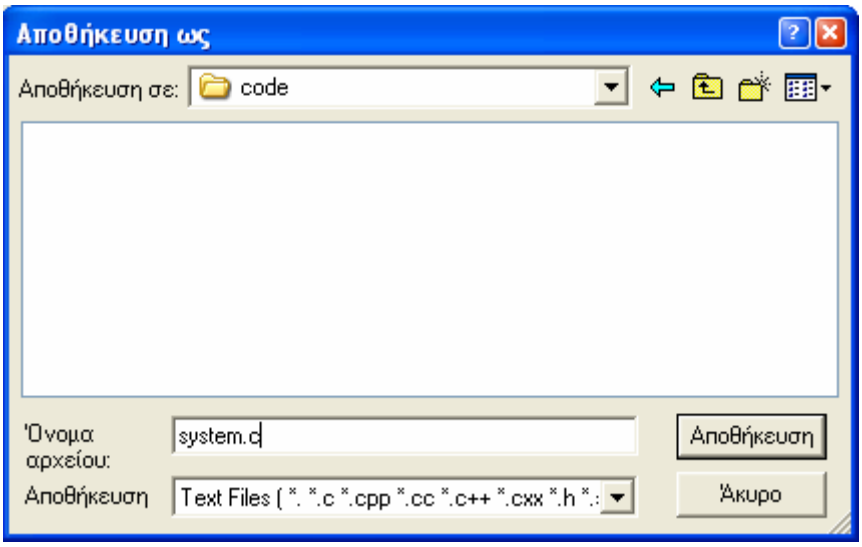
Με τον τρόπο αυτό οι καταχωρητές ελέγχονται από το λογισμικό σαν απλές θέσεις μνήμης, ενώ η διεύθυνση βάσης, XPAR_OPB_DIO2_0_BASEADDR, παρέχεται σαν σταθερά από την επικεφαλίδα xparametetrs.h που ανοίγει με διπλό-κλικ στο υπόδεντρο microblaze_0, της καρτέλας System του αριστερού παραθύρου του XPS.



Σχήμα 5.35 – Άνοιγμα της επικεφαλίδας xparameters.h

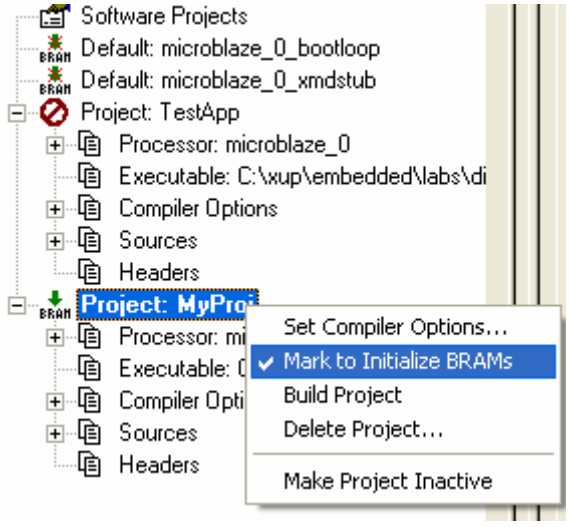
Γνωρίζοντας όλα τα παραπάνω ο πηγαίος κώδικας περιγράφει μια εφαρμογή όπου, σε έναν ατέρμονο βρόχο, διαβάζεται η τιμή των διακοπτών, απεικονίζεται στα LEDs, και στην οθόνη επτά τμημάτων απεικονίζεται ένας μετρητής που αυξάνει με βήμα όσο είναι η τιμή των διακοπτών. Οι αρχικές εγγραφές / αναγνώσεις των καταχωρητών πραγματοποιούνται για να καθαρίσουν οι διάδρομοι δεδομένων.

Από το μενού του XPS επιλέγουμε File → Save As για να σώσουμε το αρχείο του πηγαίου κώδικα με την ονομασία που επιθυμούμε (προτιμάμε την system.c) και σε οποιαδήποτε τοποθεσία μέσα στους φακέλους του project.



Σχήμα 5.36 – Αποθήκευση του αρχείου C

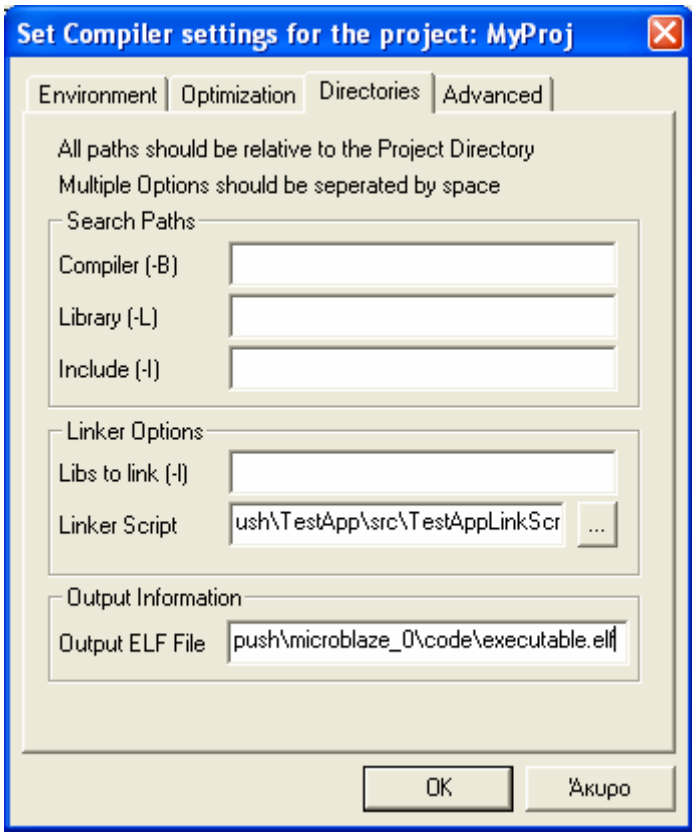
Προτού να είμαστε σε θέση να υλοποιήσουμε το τελικό σύστημα θα πρέπει να ρυθμίσουμε τις παραμέτρους των εφαρμογών (υπάρχει και η TestApp που έχει δημιουργήσει ο βοηθός BSB). Στην καρτέλα Applications κάνουμε δεξί-κλικ στο Project: TestApp και επιλέγουμε Make Project Inactive προκειμένου η εφαρμογή αυτή, που δεν μας χρειάζεται, να μην συμμετέχει στην δημιουργία του λογισμικού. Αντίστοιχα βεβαιωνόμαστε ότι η εφαρμογή μας είναι ενεργή και θα διαμορφώσει την μνήμη BRAM του συστήματος, κάνοντας δεξί-κλικ στο Project: MyProj και φροντίζοντας το Mark to Initialize BRAMs να είναι επιλεγμένο.



Σχήμα 5.37 – Επιλογή ενημέρωσης της μνήμης BRAM με τον κώδικα της εφαρμογής

Στο υπόδεντρο της MyProj κάνουμε διπλό-κλικ στο Add File προκειμένου να επισυνάψουμε το αρχείο C που γράψαμε παραπάνω στην εφαρμογή. Περιηγούμαστε στους φακέλους του project, βρίσκουμε το αρχείο και το επιλέγουμε. Έπειτα κάνουμε διπλό-κλικ στο Compiler Options για να εμφανιστεί το παράθυρο διαλόγου Set Compiler settings for the project: MyProj. Στην καρτέλα Directories, στο πεδίο Linker Script επιλέγουμε το αντίστοιχο αρχείο της

εφαρμογής TestApp, TestAppLinkScr, που βρίσκεται στον φάκελο Project Directory\TestApp\src\. Στην ίδια καρτέλα, στο πεδίο Output ELF File ορίζουμε τον φάκελο προορισμού για το εκτελέσιμο αρχείο ELF της εφαρμογής μας, ως Project Directory\microblaze_0\code\executable.elf. Πατάμε OK για να αποδεχτούμε τις αλλαγές.



Σχήμα 5.38 – Ρυθμίσεις φακέλων του μεταγλωττιστή

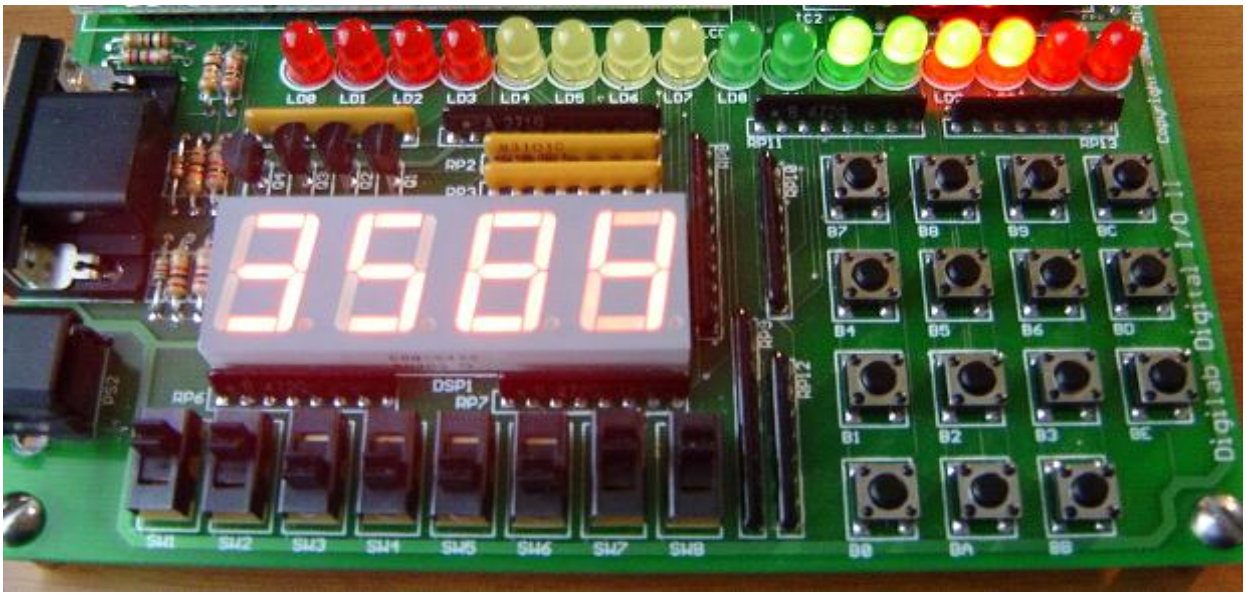
Αν αναπτύξουμε το υπόδεντρο Compiler Options της εφαρμογής μας θα διαπιστώσουμε ότι υπάρχει συντόμευση για το linker script. Κάνουμε διπλό-κλικ επάνω της για να ανοίξουμε το αρχείο στο κυρίως παράθυρο του XPS. Εκεί στην γραμμή 6 αλλάζουμε την τιμή LENGTH από 0x1fff σε 0x0fff, ώστε να αντιστοιχεί στο μέγεθος μνήμης που χρησιμοποιούμε. Σώζουμε και κλείνουμε το αρχείο.

Πλέον είμαστε έτοιμοι να μεταγλωττίσουμε το αρχείο, πατώντας  στην μπάρα με τα εργαλεία του XPS. Το αποτέλεσμα φαίνεται στο σχήμα 5.39.

```
mb-size microblaze_0/code/executable.elf
   text    data     bss     dec     hex filename
    620      16       4     640     280 microblaze_0/code/executable.elf
Done.
```

Σχήμα 5.39 – Αποτέλεσμα της μεταγλώττισης του πηγαίου κώδικα της εφαρμογής

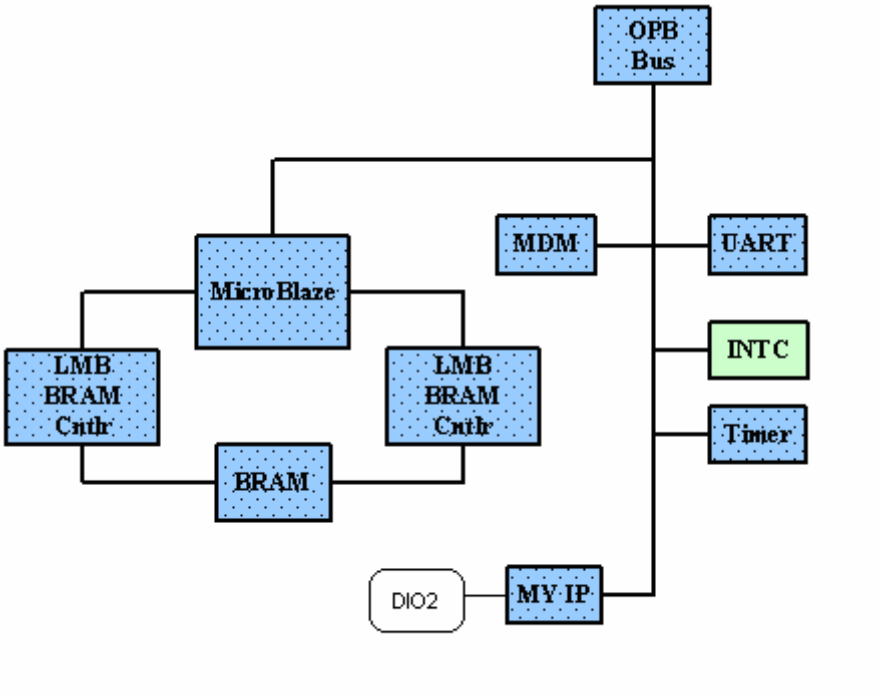
Έχοντας έτοιμο τον εκτελέσιμο κώδικα επιλέγουμε Tools → Generate Bitstream για να δημιουργηθεί το netlist του νέου συστήματος και το bitstream. Για να ενσωματωθεί το λογισμικό στο σύστημα επιλέγουμε Tools → Update Bitstream και το αρχείο BIT ενημερώνεται με τον κώδικα της εφαρμογής. Τέλος επιλέγουμε Tools → Download για να κατεβάσουμε το αρχείο BIT. Παρατηρούμε ότι στην οθόνη επτά τμημάτων εμφανίζεται ένας μετρητής που αυξάνει τόσο όσο είναι η τιμή των διακοπτών, η οποία εμφανίζεται στα LEDs.



Σχήμα 5.40 – Στιγμιότυπο από την λειτουργία του συστήματος του τέταρτου βήματος

5.2.5 Βήμα 5ο : Συγγραφή προχωρημένης εφαρμογής λογισμικού

Στο προηγούμενο βήμα υλοποιήσαμε ένα ολοκληρωμένο σύστημα με μία απλή εφαρμογή. Εδώ θα προχωρήσουμε στην συγγραφή μιας πιο σύνθετης εφαρμογής όπου θα γίνει χρήση και του χρονομέτρη, ενώ θα διαμορφωθεί το σύστημα ώστε να υποστηρίζει και διακοπές. Η αναγκαία προσθήκη στην πλατφόρμα υλικού θα είναι ο ελεγκτής διακοπών opb_intc ώστε να δημιουργηθεί το σύστημα του σχήματος 5.41.



Σχήμα 5.41 – Το ολοκληρωμένο σύστημα του πέμπτου βήματος

Για να προσθέσουμε τον ελεγκτή επιλέγουμε Project → Add/Edit Cores. Στο παράθυρο που εμφανίζεται, στην καρτέλα Peripherals, προσθέτουμε το στοιχείο opb_intc και ορίζουμε την διεύθυνση βάσης στην τιμή 0x80002400 και το πεδίο high address στην τιμή 0x800025ff. Στην

καρτέλα Bus Connections συνδέουμε το περιφερειακό με τον διάδρομο δεδομένων OPB και στην καρτέλα Ports προσθέτουμε στο σύστημα τις θύρες του opb_intc OPB_Clk, Intr και Irq. Επίσης προσθέτουμε την θύρα INTERRUPT του στοιχείου microblaze_0. Προτού συνεχίσουμε με την επεξεργασία των συνδέσεων των θυρών θα εξηγήσουμε την διαδικασία των διακοπών. Εμείς επιθυμούμε μετά από ένα περιοδικό φαινόμενο που θα συμβαίνει στον χρονομέτρη delay να γίνεται μία αίτηση διακοπής. Για τον σκοπό αυτό προσθέτουμε στο σύστημα την θύρα Interrupt του στοιχείου delay. Αλλάζουμε το πεδίο Net Name σε timer1 και μετατρέπουμε το σήμα σε εσωτερικό, Internal. Η αίτηση διακοπής καταλήγει στον ελεγκτή διακοπών opb_intc επομένως αλλάζουμε το Net Name της θύρας Intr σε timer1 για να συμφωνεί με αυτό της αίτησης διακοπής από τον χρονομέτρη. Επίσης αλλάζουμε το Net Name του OPB_Clk σε sys_clk_s για να είναι συμβατό με το Net Name του ρολογιού του συστήματος. Ο ελεγκτής διακοπών όταν δεχτεί την αίτηση για διακοπή την μεταβιβάζει στην κατάλληλη θύρα του επεξεργαστή MicroBlaze. Συνεπώς αλλάζουμε το Net Name της θύρας Irq του ελεγκτή διακοπών σε interrupt και το ίδιο κάνουμε με το Net Name της INTERRUPT του στοιχείου microblaze_0. Με τον τρόπο αυτό το σήμα για την αίτηση διακοπής που προέρχεται από τον ελεγκτή διακοπών είναι το ίδιο με αυτό που καταλήγει στην είσοδο των διακοπών του επεξεργαστή. Όλα τα παραπάνω σήματα χαρακτηρίζονται ως εσωτερικά, Internal, και τελικά η καρτέλα Ports θα πρέπει να έχει διαμορφωθεί όπως στο σχήμα 5.42.

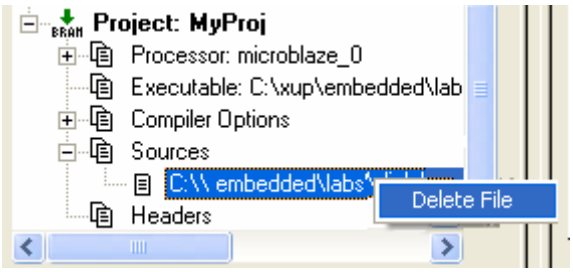
Instance	Port Name	Net Name	Pola...	Scope	Range	Class
microblaze_0	CLK	sys_clk_s	I	Internal		CLK
microblaze_0	DBG_CAPTURE	DBG_CAPTUR...	I	Internal		
microblaze_0	DBG_CLK	DBG_CLK_s	I	Internal		
microblaze_0	DBG_REG_EN	DBG_REG_EN_s	I	Internal		
microblaze_0	DBG_TDI	DBG_TDI_s	I	Internal		
microblaze_0	DBG_TDO	DBG_TDO_s	O	Internal		
microblaze_0	DBG_UPDATE	DBG_UPDATE_s	I	Internal		
microblaze_0	INTERRUPT	interrupt	I	Internal		INTERR.
debug_mo...	OPB_Clk	sys_clk_s	I	Internal		CLK
debug_mo...	DBG_CAPTUR...	DBG_CAPTUR...	O	Internal		
debug_mo...	DBG_CLK_0	DBG_CLK_s	O	Internal		
debug_mo...	DBG_REG_EN_0	DBG_REG_EN_s	O	Internal		
debug_mo...	DBG_TDI_0	DBG_TDI_s	O	Internal		
debug_mo...	DBG_TDO_0	DBG_TDO_s	I	Internal		
debug_mo...	DBG_UPDATE_0	DBG_UPDATE_s	O	Internal		
RS232	OPB_Clk	sys_clk_s	I	Internal		CLK
RS232	RX	RS232_RX	I	External		
RS232	TX	RS232_TX	O	External		
delay	OPB_Clk	sys_clk_s	I	Internal		Clk
delay	CaptureTrig0	net_gnd	I	Internal		
delay	Interrupt	timer1	O	Internal		INTERR.
opb_intc_0	OPB_Clk	sys_clk_s	I	Internal		CLK
opb_intc_0	Intr	timer1	I	Internal		INTERR.
opb_intc_0	Irq	interrupt	O	Internal		INTERR.
opb_dio2_0	led	led	O	External		
opb_dio2_0	oclk	oclk	O	External		
opb_dio2_0	cs	cs	O	External		

Σχήμα 5.42 – Η τελική διαμόρφωση των θυρών του συστήματος

Πατώντας OK για να αποδεχτούμε τις ρυθμίσεις και να κλείσουμε το παραπάνω παράθυρο διαλόγου έχουμε ολοκληρώσει τις αλλαγές που χρειάζονται στην πλατφόρμα υλικού. Επόμενο

βήμα είναι η προσαρμογή της πλατφόρμας λογισμικού στα νέα δεδομένα, με την ύπαρξη διακοπών, και κυρίως ο ορισμός της ρουτίνας εξυπηρέτησης διακοπών για τον χρονομέτρη delay. Επιλέγουμε Project → Software Platform Settings, για να ανοίξουμε το ομώνυμο παράθυρο. Στην καρτέλα Processor and Driver Parameters και στο πεδίο interrupt_handler του στοιχείου delay ορίζουμε την τιμή timer_int_handler, που είναι το όνομα της ρουτίνας χειρισμού διακοπών για τον χρονομέτρη. Ο κώδικας σε C της ρουτίνας θα ακολουθήσει. Πατάμε OK για να αποδεχτούμε τις αλλαγές και επιλέγουμε Tools → Generate Libraries and BSPs για να ανανεώσουμε τις βιβλιοθήκες και τα αρχεία επικεφαλίδων.

Όπως είδαμε η ρουτίνα που θα χειρίζεται τις διακοπές του χρονομέτρη θα είναι η timer_int_handler. Η ρουτίνα αυτή θα πρέπει να περιέχεται σε μία εφαρμογή λογισμικού. Για τον λόγο αυτό μεταβαίνουμε στην καρτέλα Applications του αριστερού παραθύρου του XPS προκειμένου να προσαρμόσουμε την εφαρμογή μας, MyProj. Ο πηγαίος κώδικας (system.c) της εφαρμογής του τέταρτου βήματος δεν μας χρειάζεται πλέον και επομένως μπορούμε να παραλείψουμε το συγκεκριμένο αρχείο, κάνοντας δεξί-κλικ επάνω του και επιλέγοντας Delete File.



Σχήμα 5.43 – Αφαίρεση αρχείου από την εφαρμογή

Πλέον καλούμαστε να γράψουμε τον νέο κώδικα για την εφαρμογή μας. Για τον σκοπό αυτό ανοίγουμε ένα νέο αρχείο στον επεξεργαστή κειμένου του XPS και γράφουμε τον κώδικα που ακολουθεί:

```
#include <xtmrctr_l.h>     /* headers for the timer */
#include <xintc_l.h>       /* headers for the interrupts*/
#include <xbasic_types.h>
#include <xparameters.h>

/* the global counter of the interrupts */
Xuint32 timer_count = 0;


/* the interrupt handler */
void timer_int_handler(void * baseaddr_p)
{

/* the address of the peripheral that the handler controls */
Xuint32 baseaddr = (int)baseaddr_p;
Xuint32 csr;

    /* read the status register */
    csr = XTmrCtr_mGetControlStatusReg(baseaddr, 0);

    /* check if there is an interrupt and increase the counter */
    if (csr & XTC_CSR_INT_OCCURED_MASK)
```

```
        {
        timer_count++;
        }

    /* show the counter */
    dispLED(timer_count);

    /* clear the interrupt */
    XTmrCtr_mSetControlStatusReg(baseaddr, 0, csr);

}

void main()
{
  /* enable the interrupt controller */
  XIntc_mMasterEnable(XPAR_OPB_INTC_0_BASEADDR);

  /* set the load register with the value that the timer will count down before it interrupts */
  XTmrCtr_mSetLoadReg(XPAR_DELAY_BASEADDR, 0, 50000000 - 2);

  /* clear the interrupts and reset the timer */
  XTmrCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0,
XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK );

  /* enable the timer interrupts in the interrupt controller */
  XIntc_mEnableIntr(XPAR_OPB_INTC_0_BASEADDR,
XPAR_DELAY_INTERRUPT_MASK);

  /* enable the only counter, its interrupt, to reload from the load register and to count down */
  XTmrCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0,
XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_ENABLE_INT_MASK |
XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);

  /* enable MicroBlaze interrupts */
  microblaze_enable_interrupts();


  /* wait for the interrupts */
  while (1);

}
```

Η ρουτίνα timer_int_handler, η οποία καλείται αυτόματα όταν πρόκειται να χειριστεί μια διακοπή που πηγάζει από τον χρονομέτρη, παίρνει ως όρισμα την διεύθυνση της συσκευής που αφορά η διακοπή. Σε μια σύντομη περιγραφή του παραπάνω κώδικα, της λειτουργίας του χρονομέτρη και των σταθερών αυτού που χρησιμοποιούνται, μπορούμε να πούμε ότι ο χρονομέτρης, με βάση τις ρυθμίσεις που του έχουμε δώσει από το δεύτερο βήμα, βασίζεται σε τρεις καταχωρητές των 32 bit. Ο ένας είναι ο Load Register στον οποίο φορτώνεται η τιμή από την οποία αρχίζει να μετράει ο χρονομέτρης. Η προεπιλεγμένη λειτουργία του χρονομέτρη είναι να μετράει έως ότου μηδενιστεί εάν μετράει προς τα κάτω ή υπερχειλίσει εάν μετράει προς τα πάνω. Εμείς χρησιμοποιούμε την προς τα κάτω μέτρηση. Αρχικά με την εντολή XTmrCtr_mSetLoadReg(XPAR_DELAY_BASEADDR, 0, 50000000 - 2) θέτουμε στον Load

Register κατάλληλη τιμή ώστε να συμβαίνει μία διακοπή ανά δευτερόλεπτο. Η τιμή αυτή προκύπτει από τον τύπο που δίνεται στο βιβλίο δεδομένων του χρονομέτρη:

TIMING_INTERVAL = (TLRx + 2) X OPB_CLOCK_PERIOD =>
TLRx = (TIMING_INTERVAL X OPB_CLOCK_FREQUENCY) – 2

όπου το επιθυμητό TIMING_INTERVAL είναι 1 δευτερόλεπτο, TLRx είναι η τιμή που πρέπει να πάρει ο Load Register και φυσικά 50000000 Hz η συχνότητα του OPB. Κάθε φορά που πρόκειται να ξεκινήσει μια νέα διαδικασία μέτρησης, ο Load Register αντιγράφεται στον δεύτερο καταχωρητή, τον Timer/Counter Register όπου και πραγματοποιείται η μέτρηση. Ο τρίτος καταχωρητής είναι ο Control/Status Register ο οποίος περιέχει όλες τις πληροφορίες που θέλουμε να ξέρουμε για τον χρονομέτρη ή να εγγράψουμε σε αυτόν. Διαβάζουμε με την XTmrCtr_mGetControlStatusReg(BaseAddress, TmrCtrNumber) και γράφουμε με την XTmrCtr_mSetControlStatusReg(BaseAddress, TmrCtrNumber, RegisterValue). Όπου το TmrCtrNumber δείχνει σε ποιον από τους δύο μετρητές, που υποστηρίζει ο χρονομέτρης, αναφερόμαστε, τον 0 ή τον 1, και συνεπώς στην περίπτωσή μας που έχουμε μόνο έναν έχει την τιμή 0. Με την βοήθεια του καταχωρητή αυτού,

```
csr = XTmrCtr_mGetControlStatusReg(baseaddr, 0);

if (csr & XTC_CSR_INT_OCCURED_MASK) …,
```

μπορούμε να εξετάσουμε αν έχει μηδενιστεί ο μετρητής, οπότε και προκαλείται διακοπή, καθώς και να εκτελέσουμε πλήθος άλλων ενεργειών όπως να ελέγξουμε την φορά μέτρησης του χρονομέτρη και να εκκινήσουμε την διαδικασία μέτρησης. Τέλος σημαντικό είναι και το γεγονός ότι πρέπει να ενεργοποιηθούν οι διακοπές στον χρονομέτρη, τον ελεγκτή διακοπών και φυσικά στον MicroBlaze.

Σώζουμε τον κώδικα που έχουμε γράψει στο κυρίως παράθυρο του XPS με όνομα system_timer.c και ανοίγουμε ένα νέο έγγραφο. Από το system_timer.c προκύπτει ότι ο χρονομέτρης προκαλεί μία διακοπή κάθε δευτερόλεπτο, η ρουτίνα χειρισμού αυτής αυξάνει έναν (global) μετρητή, τον περνάει σαν όρισμα σε μία συνάρτηση dispLED και καθαρίζει την μάσκα των διακοπών. Η συνάρτηση dispLED αντιστοιχεί στον παρακάτω κώδικα:

```
#include <xbasic_types.h>
#include <xparameters.h>

void dispLED( Xuint32 Val)
{
Xuint16  *ledptr;

  ledptr = (Xuint16 *)XPAR_OPB_DIO2_0_BASEADDR;
  *ledptr = Val;

}
```

όπου φαίνεται ότι ο αριθμός του (global) μετρητή, δηλαδή του αριθμού των διακοπών που έχουν συμβεί, εμφανίζεται στην οθόνη επτά τμημάτων. Σώζουμε και αυτό το αρχείο με το όνομα 7segled.c.

Έχοντας γράψει τον κώδικα της εφαρμογής μας σε δύο αρχεία, τα επισυνάπτουμε σε αυτήν με την επιλογή Add File, στο υπόδεντρο Project: MyProj, και είμαστε έτοιμοι να επιβεβαιώσουμε το σύστημα. Επιλέγουμε Tools → Generate Netlist και το netlist του συστήματος θα ξαναδημιουργηθεί, αφού έχει αλλάξει με την προσθήκη του ελεγκτή διακοπών.

Στην συνέχεια επιλέγουμε Tools → Generate Bitstream για να δημιουργηθεί το αρχείο BIT του συστήματος, το οποίο όμως δεν περιλαμβάνει το λογισμικό. Για τον λόγο αυτό μεταγλωττίζουμε τον κώδικα της εφαρμογής μας πατώντας  από την μπάρα με τα εργαλεία του XPS και επιλέγουμε Tool → Update Bitstream, για να ανανεωθεί το αρχείο BIT με το λογισμικό. Τέλος με Tools → Download κατεβάζουμε το σύστημα στο FPGA και παρατηρούμε έναν μετρητή στην οθόνη επτά τμημάτων που αυξάνει κατά ένα κάθε δευτερόλεπτο.

5.2.6 Βήμα 6^{ο} : Εκσφαλμάτωση λογισμικού και υλικού

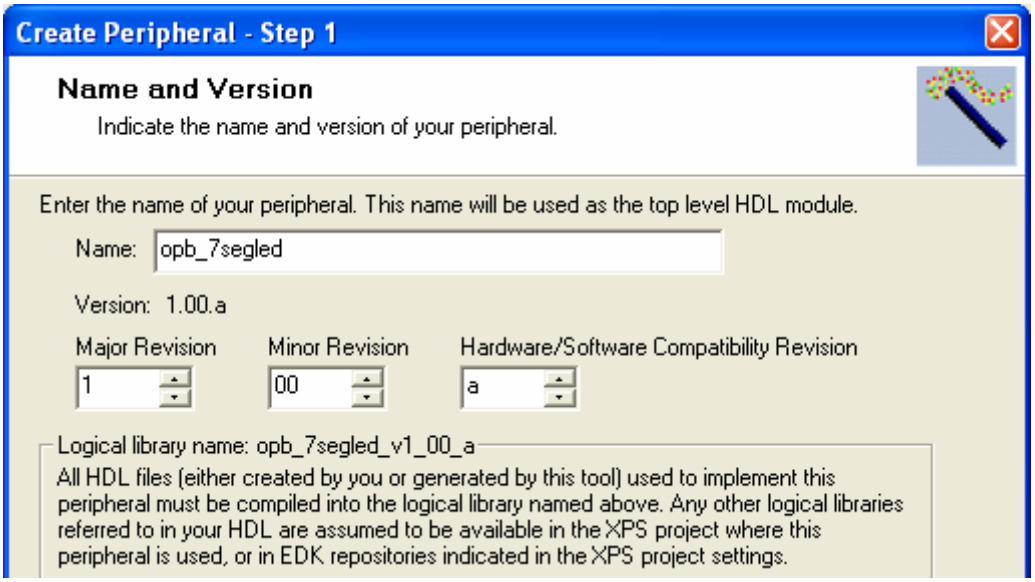
5.2.6.1 Υλοποίηση του νέου συστήματος

Στο τελευταίο βήμα της πρώτης εφαρμογής θα μας δοθεί η δυνατότητα να εκτελέσουμε τα βασικά βήματα για την εκσφαλμάτωση ενός συστήματος. Λόγω του ότι τα IPs που θα χρησιμοποιήσουμε για την εκσφαλμάτωση υλικού δεν μπορούν να συμπεριληφθούν στο FPGA που διαθέτει η κάρτα Digilab 2, θα χρησιμοποιήσουμε την κάρτα Spartan-3 Starter Kit.

Όπως ακριβώς κάναμε και στο πρώτο βήμα αυτής της εφαρμογής θα χρησιμοποιήσουμε τον βοηθό BSB για να δημιουργήσουμε το αρχικό μας σύστημα που θα περιέχει τα εξής IP:

- MicroBlaze
- LMB BRAM controllers for BRAM
- OPB bus
- OPB MDM (μονάδα εκσφαλμάτωσης)
- OPB UART

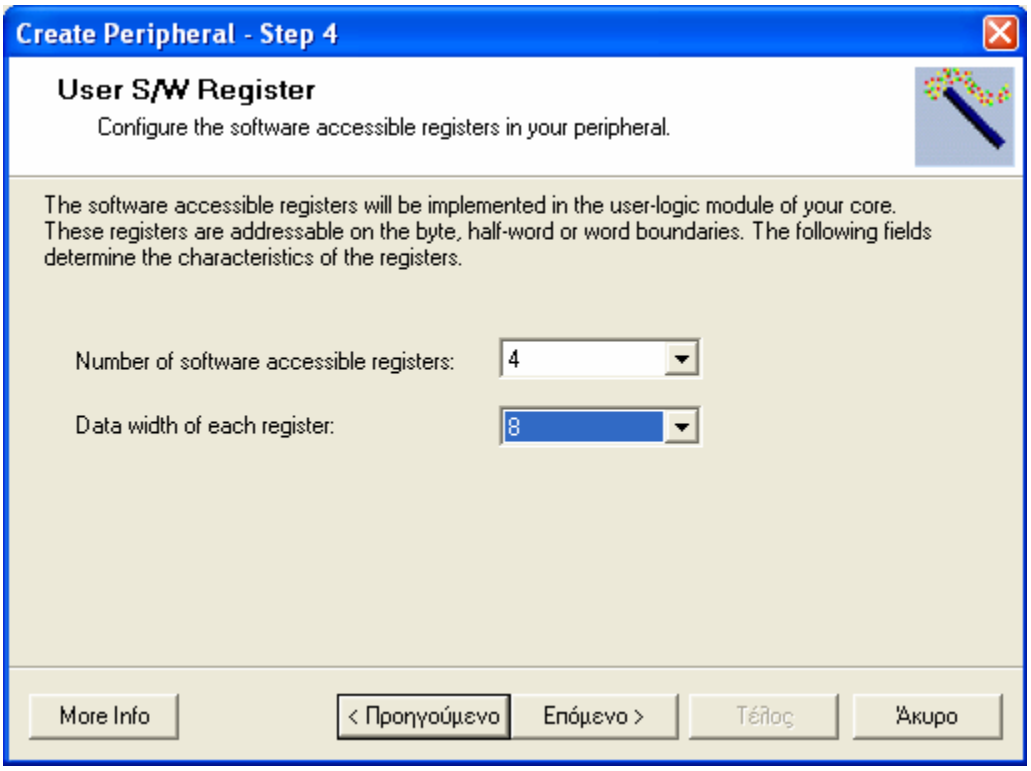
Η διαδικασία που ακολουθούμε με τον βοηθό είναι ακριβώς η ίδια με αυτήν που περιγράψαμε στις σελίδες 115-118. Στο σύστημα που προκύπτει πρέπει να προσθέσουμε ένα περιφερειακό που θα ελέγχει την οθόνη επτά τμημάτων της κάρτας Spartan-3 Starter Kit. Κατά τα γνωστά εκκινούμε τον βοηθό δημιουργίας περιφερειακού (Tools → Import Peripheral Wizard) και αφού περάσουμε τα αρχικά παράθυρα διαλόγου του δίνουμε την ονομασία opb_7segled.



Σχήμα 5.44 – Η ονομασία του νέου περιφερειακού

Στην συνέχεια συνδέουμε το περιφερειακό με τον διάδρομο δεδομένων OPB, επιλέγουμε την υπηρεσία των καταχωρητών που ελέγχονται από το λογισμικό για την διεπαφή IPIF και

ορίζουμε τέσσερις καταχωρητές των οκτώ bit. Κάθε ένας από αυτούς τους καταχωρητές θα φιλοξενεί κατάλληλα διαμορφωμένες τιμές για τους τέσσερις χαρακτήρες προς εμφάνιση.



Σχήμα 5.45 – Ορισμός των καταχωρητών του περιφερειακού

Αφού ολοκληρώσουμε τον οδηγό, περιηγούμαστε στους φακέλους του περιφερειακού και ανοίγουμε στον επεξεργαστή κειμένου του XPS το αρχείο opb_7segled.vhd. Η λειτουργία του ελεγκτή για το περιφερειακό μας στηρίζεται σε δύο θύρες, με βάση την λειτουργία της οθόνης επτά τμημάτων όπως αυτή περιγράφεται στο κεφάλαιο τρία. Στην μία θύρα, digit, οδηγούνται τέσσερα σήματα που το καθένα αντιστοιχεί στους τέσσερις χαρακτήρες προς εμφάνιση, με LSB το σήμα του δεξιότερου χαρακτήρα (AN0). Όταν το σήμα του κάθε χαρακτήρα έχει τιμή λογικό ‘0’ τότε αυτός εμφανίζεται. Στην δεύτερη θύρα, segment, οδηγούνται οκτώ σήματα που αντιστοιχούν στα επτά τμήματα και την τελεία (DP) του κάθε χαρακτήρα, που ανάβουν με τιμή λογικό ‘0’, με LSB το τμήμα a. Επομένως στην γραμμή 121 του αρχείου opb_7segled.vhd προσθέτουμε τις παρακάτω δηλώσεις:

```
-- ADD USER PORTS BELOW THIS LINE ------------------
digit       : out std_logic_vector(0 to 3);
segment     : out std_logic_vector(0 to 7);
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

Επίσης στην γραμμή 372 προσθέτουμε τα παρακάτω:

```
-- MAP USER PORTS BELOW THIS LINE ------------------
digit   => digit,
segment => segment,
-- MAP USER PORTS ABOVE THIS LINE ------------------
```

Αφού σώσουμε και κλείσουμε το αρχείο ανοίγουμε το user_logic.vhd. Στην γραμμή 100 προσθέτουμε τις δηλώσεις των θυρών:

```
-- ADD USER PORTS BELOW THIS LINE ------------------
digit       : out std_logic_vector(0 to 3);
segment     : out std_logic_vector(0 to 7);
-- ADD USER PORTS ABOVE THIS LINE ------------------
```

Αντικαθιστούμε το τμήμα architecture του κώδικα με αυτό που ακολουθεί:

```
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  ---------------------------------------------
  -- User logic S/W accessible registers
  ---------------------------------------------
  signal reg0              : std_logic_vector(0 to C_DWIDTH-1);
  signal reg1              : std_logic_vector(0 to C_DWIDTH-1);
  signal reg2              : std_logic_vector(0 to C_DWIDTH-1);
  signal reg3              : std_logic_vector(0 to C_DWIDTH-1);
  signal reg_write_select          : std_logic_vector(0 to 3);
  signal reg_read_select : std_logic_vector(0 to 3);
  signal digit_select        : std_logic_vector(0 to 1);
  signal digit_i       : std_logic_vector(0 to 3);
  signal segment_i      : std_logic_vector(0 to 7);
 signal mhertz_count        : std_logic_vector(5 downto 0);
 signal khertz_count        : std_logic_vector(9 downto 0);
 signal mhertz_en           : std_logic;
 signal khertz_en           : std_logic;

begin

  ---------------------------------------------------------------
  -- Map user logic S/W register read/write select signal
  ---------------------------------------------------------------
  reg_write_select <= Bus2IP_WrCE;
  reg_read_select <= Bus2IP_RdCE;

  ---------------------------------------------------------------
  -- User logic S/W accessible registers write action
  ---------------------------------------------------------------
  REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        reg0 <= (others => '0');
        reg1 <= (others => '0');
        reg2 <= (others => '0');
        reg3 <= (others => '0');
      else
        case reg_write_select is
          when "1000" => reg0 <= Bus2IP_Data;
          when "0100" => reg1 <= Bus2IP_Data;
          when "0010" => reg2 <= Bus2IP_Data;
          when "0001" => reg3 <= Bus2IP_Data;
          when others => null;
        end case;
      end if;
    end if;
  end process REG_WRITE_PROC;

  ---------------------------------------------------------------
```

```
  -- User logic S/W accessible registers read action
  ---------------------------------------------------------------
  REG_READ_PROC : process( reg_read_select, reg0, reg1, reg2, reg3 ) is
  begin
       case reg_read_select is
                when "1000" => IP2Bus_Data <= reg0;
                when "0100" => IP2Bus_Data <= reg1;
                when "0010" => IP2Bus_Data <= reg2;
                when "0001" => IP2Bus_Data <= reg3;
                when others => IP2Bus_Data <= (others => '0');
       end case;
  end process REG_READ_PROC;

  ----------------------------------------
  -- Clock Dividers
  ----------------------------------------
GEN_1MHZ : process (Bus2IP_Clk) is
begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
      mhertz_count <= (others => '0');
      mhertz_en <= '0';
         else
      mhertz_count <= mhertz_count + 1;
      if mhertz_count = "110010" then
           mhertz_en <= '1' ;
           mhertz_count <= (others => '0');
      else
           mhertz_en <= '0';
      end if;
    end if;
  end if;
end process GEN_1MHZ;

GEN_1KHZ : process (Bus2IP_Clk) is
begin
  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
       if Bus2IP_Reset = '1' then
      khertz_count <= (others => '0');
      khertz_en <= '0';
    else
      if mhertz_en = '1' then
           khertz_count <= khertz_count + 1;
           if khertz_count = "1111101000" then
                khertz_en <= '1';
                khertz_count <= (others => '0');
                     end if;
        else
                khertz_en <= '0';
        end if;
    end if;
  end if;
end process GEN_1KHZ;

  ----------------------------------------
  -- User logic display update
  ----------------------------------------
  CYC_DISP_PROC : process( Bus2IP_Clk ) is
  begin
       if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                if Bus2IP_Reset = '1' then
                     digit_select <= (others => '0');
```

```
                        digit_i     <= (others => '1');
                        segment_i   <= (others => '1');
                else
            if khertz_en = '1' then
                digit_select <= digit_select + 1 ;
                        case digit_select is
                                when "00" => segment_i <= reg0;  digit_i <= "1110";
                                when "01" => segment_i <= reg1;  digit_i <= "1101";
                                when "10" => segment_i <= reg2;  digit_i <= "1011";
                                when "11" => segment_i <= reg3;  digit_i <= "0111";
                                when others => null;
                        end case;
                  end if;
                end if;
        end if;
    end process CYC_DISP_PROC;

    ------------------------------------------------------
    -- Drive the IP2Bus signals
    ------------------------------------------------------
    IP2Bus_Ack    <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3) or
Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3);
    IP2Bus_Error  <= '0';
    IP2Bus_Retry  <= '0';
    IP2Bus_ToutSup <= '0';

    digit  <= digit_i;
    segment <= segment_i;

end IMP;
```

Αν και ο παραπάνω κώδικας δεν παρουσιάζει κάποια ιδιαίτερη δυσκολία, μπορούμε να πούμε σαν μια σύντομη περιγραφή ότι στους τέσσερις καταχωρητές (reg0-reg3) που ελέγχονται από το λογισμικό, με τρόπο που έχουμε εξηγήσει στα προηγούμενα βήματα, διαβιβάζεται σε κατάλληλη δυαδική μορφή η πληροφορία για το ποια τμήματα θα ανάψουν για κάθε χαρακτήρα, παράγεται ένα περιοδικό σήμα, khertz_en, που αλλάζει με συχνότητα 1 KHz και με αυτήν την συχνότητα οι τέσσερις χαρακτήρες εμφανίζονται διαδοχικά. Επομένως μόνο ένας χαρακτήρας εμφανίζεται κάθε στιγμή αλλά η πολύ γρήγορη εναλλαγή τους ξεγελάει το μάτι και φαίνεται σαν να εμφανίζονται ταυτόχρονα.

Αφού σώσουμε και κλείσουμε το παραπάνω αρχείο προχωράμε στην τελευταία ενέργεια για την προετοιμασία του περιφερειακού μας, δηλαδή στην δήλωση των θυρών αυτού στο αρχείο opb_7segled.mpd. Ανοίγουμε το αρχείο και στην γραμμή 27 προσθέτουμε τις παρακάτω δηλώσεις:

```
## Ports
PORT digit  =  "", DIR = O, VEC = [0:3]
PORT segment  =  "", DIR = O, VEC = [0:7]
```

Σώζουμε και κλείνουμε το αρχείο και πλέον το περιφερειακό μας είναι έτοιμο να προστεθεί στο σύστημα. Για τον λόγο αυτό κλείνουμε και ανοίγουμε πάλι το project ώστε αν γίνει αντιληπτή η ύπαρξη του νέου περιφερειακού. Επιλέγουμε Project → Add/Edit Cores και στην καρτέλα Peripherals προσθέτουμε το περιφερειακό στο σύστημα, όπως φαίνεται στο σχήμα 5.46.

Peripheral	HW Ver	Instance	Base Address	High Address	Min
microblaze	2.10.a	microblaze_0			
opb_mdm	2.00.a	debug_module	0x80002000	0x800020ff	0x100
lmb_bram_i...	1.00.b	dlmb_cntlr	0x00000000	0x00001fff	0x800
lmb_bram_i...	1.00.b	ilmb_cntlr	0x00000000	0x00001fff	0x800
bram_block	1.00.a	lmb_bram			
opb_uartlite	1.00.b	RS232	0x80002100	0x800021ff	0x100
opb_7segled	1.00.a	opb_7segled_0	0x80002200	0x800022ff	0x100

Σχήμα 5.46 – Προσθήκη του περιφερειακού για τον έλεγχο της οθόνης επτά τμημάτων

Στην καρτέλα Bus Connections συνδέουμε το περιφερειακό με τον διάδρομο δεδομένων OPB, ενώ στην καρτέλα Ports προσθέτουμε τις θύρες digit, segment και OPB_Clk του στοιχείου opb_7segled_0 στο σύστημα και τις διαμορφώνουμε όπως στο σχήμα 5.47.

opb_7segled_0	digit	digit	O	External	[0:3]	
opb_7segled_0	segment	segment	O	External	[0:7]	
opb_7segled_0	OPB_Clk	sys_clk_s	I	Internal		Clk

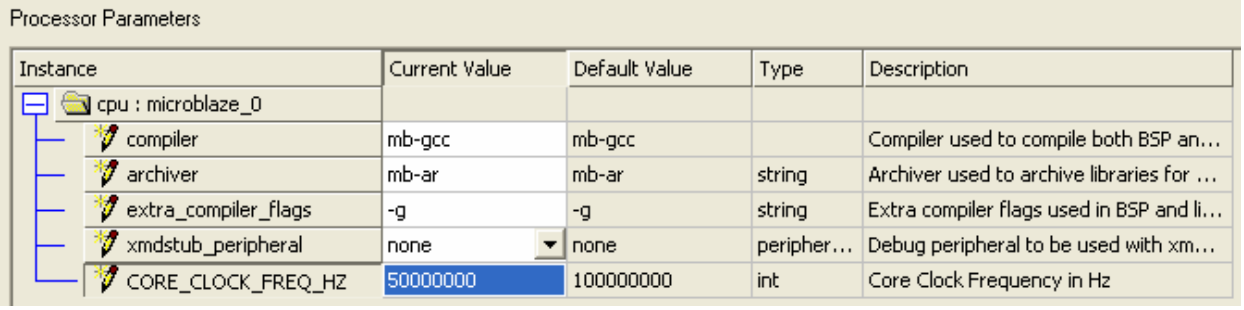
Σχήμα 5.47 – Διαμόρφωση των θυρών του στοιχείου opb_7segled_0

Πατάμε OK για να αποδεχτούμε τις αλλαγές που κάναμε στο παράθυρο διαλόγου και ανοίγουμε το αρχείο system.ucf για αντιστοιχίσουμε τα σήματα των θυρών του περιφερειακού με τους κατάλληλους ακροδέκτες του FPGA, όπως αυτοί προκύπτουν από την περιγραφή της κάρτας Spartan-3 Starter Kit που βρίσκεται στο κεφάλαιο τρία. Στο τέλος του αρχείου προσθέτουμε τις παρακάτω γραμμές:

```
NET  digit<3>          LOC=d14;
NET  digit<2>          LOC=g14;
NET  digit<1>          LOC=f14;
NET  digit<0>          LOC=e13;

NET  segment<7>          LOC=e14;
NET  segment<6>          LOC=g13;
NET  segment<5>          LOC=n15;
NET  segment<4>          LOC=p15;
NET  segment<3>          LOC=r16;
NET  segment<2>          LOC=f13;
NET  segment<1>          LOC=n16;
NET  segment<0>          LOC=p16;
```

Σώζοντας και κλείνοντας το αρχείο, έχουμε ολοκληρώσει της αλλαγές που αφορούν την πλατφόρμα του υλικού.

Προτού προχωρήσουμε στην δημιουργία της εφαρμογής που θα χειρίζεται την οθόνη επτά τμημάτων, φροντίζουμε να ορίσουμε τις βασικές παραμέτρους της πλατφόρμας λογισμικού. Για τον σκοπό αυτό επιλέγουμε Project → Software Platform Settings και διαμορφώνουμε την καρτέλα Processor and Driver Parameters όπως στο σχήμα 5.48.



Σχήμα 5.48 – Ρυθμίσεις επεξεργαστή

Επίσης στην καρτέλα Library/OS Parameters βεβαιωνόμαστε ότι η σειριακή θύρα έχει προεπιλεγεί ως πρότυπη είσοδος / έξοδος και το πεδίο need_xil_malloc έχει την τιμή false. Εδώ σε αντίθεση με όλα τα προηγούμενα βήματα η σειριακή θύρα λειτουργεί κανονικά και μπορούμε να το διαπιστώσουμε κάνοντας μία σύνδεση με το πρόγραμμα Hyper Terminal των Windows. Πατάμε OK για να αποδεχτούμε τις παραπάνω ρυθμίσεις.

Στην συνέχεια μεταβαίνουμε στην καρτέλα Applications του αριστερού παραθύρου του XPS όπου καθιστούμε ανενεργή την εφαρμογή TestApp και δημιουργούμε μία νέα εφαρμογή MyProj. Ανοίγουμε ένα νέο έγγραφο στον επεξεργαστή κειμένου του XPS και γράφουμε τον κώδικα που ακολουθεί:

```
#include <xbasic_types.h>
#include <xparameters.h>

/* Global variable */
Xuint32 timer_count=0;

int main()
{
int i=1;

  /* Print to serial port */
  xil_printf("ChipScope and GDB lab\n\r");

  while (1)
        {
        for (i=1; i<400000; i++);
        dispLED(timer_count, 1);
        timer_count++;
        }
}
```

Ο παραπάνω κώδικας απλώς αυξάνει συνεχώς έναν (global) μετρητή και τον στέλνει σαν όρισμα στην συνάρτηση dispLED. Σώζουμε το αρχείο ως system_delay.c και ανοίγουμε ένα νέο έγγραφο όπου θα περιγράψουμε την συνάρτηση dispLED γράφοντας τα παρακάτω:

```
#include <xbasic_types.h>
#include <xparameters.h>

Xuint8 Digit[] =
  {
  0xC0,  /* digit 0 */
```

```
  0xF9,  /* 1 */
  0xA4,  /* 2 */
  0xB0,  /* 3 */
  0x99,  /* 4 */
  0x92,  /* 5 */
  0x82,  /* 6 */
  0xF8,  /* 7 */
  0x80,  /* 8 */
  0x90  /* 9 */
  };



void dispLED( Xuint32 Val, Xuint8 DP )
{
Xuint32 i;
Xuint8  *ledptr, dig;

  ledptr = (Xuint8 *)XPAR_OPB_7SEGLED_0_BASEADDR;

  for ( i=0; i<4; i++ )
     {
     dig = Digit[Val % 10]; /* the less significant digit */
      if ( i == DP )
         dig &= 0x7F;    /* the MSB '0' means that the dot (DP) appears */
     *(ledptr + i) = dig;
      Val = Val / 10;  /* get rid of the less significant digit */
      }
}
```

Η συνάρτηση dispLED παίρνει δύο ορίσματα. Το πρώτο είναι ένας αριθμός. Από τον αριθμό αυτό κρατούνται για εμφάνιση τα τέσσερα LSB δεκαδικά ψηφία του με το δεξιότερο στην οθόνη επτά τμημάτων αν αντιστοιχεί στο LSB. Τα ψηφία απεικονίζονται με βάση έναν πίνακα, Digit[], ο οποίος περιέχει τις κατάλληλες τιμές στα bit του ώστε να σχηματίζονται από τα επτά τμήματα τα ψηφία 0 έως 9. Το δεύτερο όρισμα, του οποίου οι έγκυρες τιμές είναι από 0 έως 3, καθορίζει ποια από τις τελείες (DP) των ψηφίων θα ανάψει, αν ανάψει κάποια. Εμείς στην εφαρμογή μας ανάβουμε την τελεία του δεύτερου ψηφίου ώστε η οθόνη επτά τμημάτων να έχει την μορφή ενός χρονομέτρου.

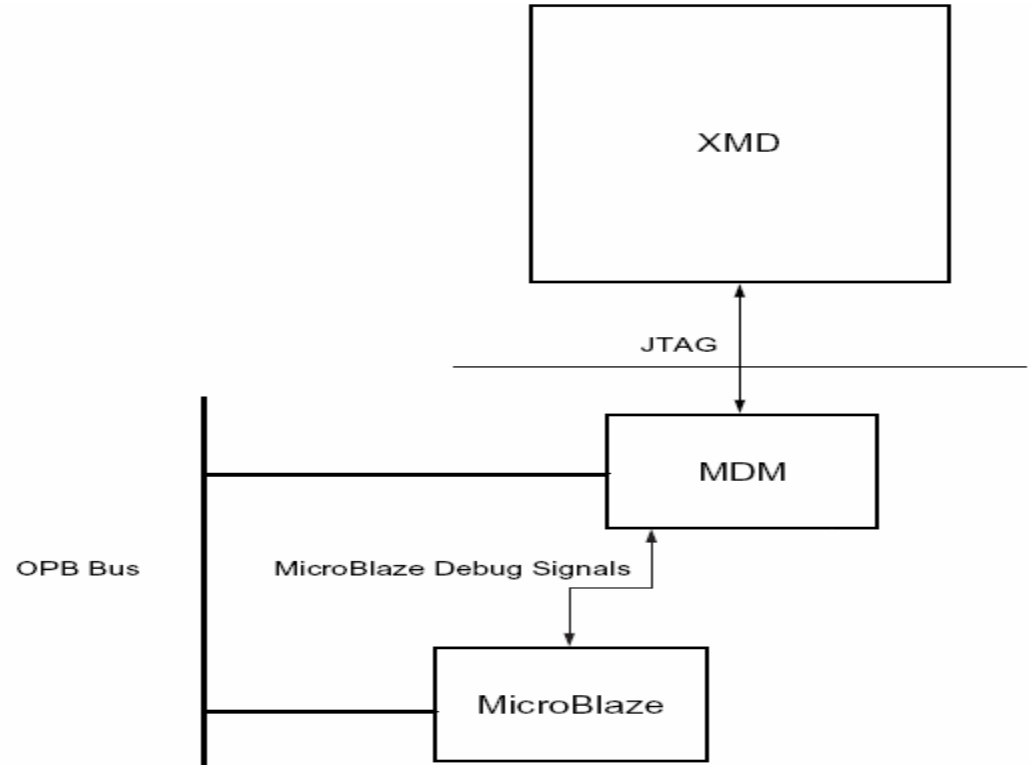
Το νέο μας σύστημα είναι έτοιμο, μπορούμε να επιλέξουμε Tools → Download και θα γίνουν αυτόματα όλες οι ενέργειες, δημιουργία του netlist, δημιουργία του bitstream, μεταγλώττιση κώδικα, ενοποίηση λογισμικού και υλικού και το κατέβασμα στο FPGA.

Σχήμα 5.49 – Στιγμιότυπο από την λειτουργία του νέου συστήματος

5.2.6.2 Εκσφαλμάτωση του συστήματος

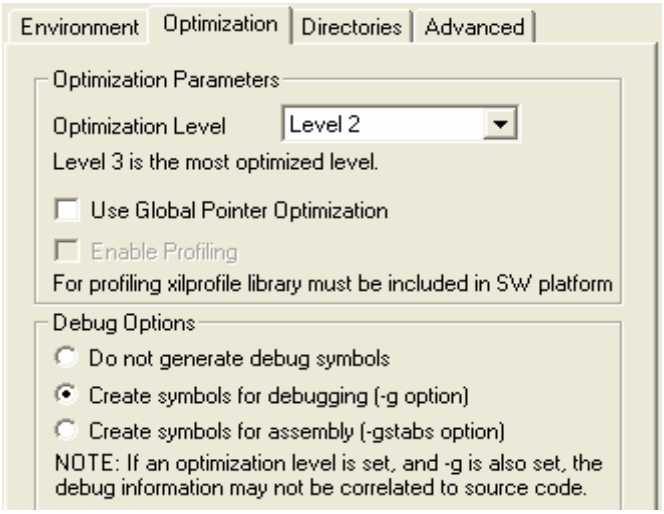
5.2.6.2.1 Εκσφαλμάτωση λογισμικού

Η εκσφαλμάτωση λογισμικού χρησιμοποιεί τον Xilinx Microprocessor Debugger (XMD), ο οποίος συνδέεται μέσω της αλυσίδας JTAG με το σύστημα. Ο XMD δεν συνδέεται απευθείας με τον επεξεργαστή, αλλά με την μονάδα εκσφαλμάτωσης MicroBlaze Debug Module (MDM), η οποία παρέχει τα κατάλληλα σήματα για την εκσφαλμάτωση του MicroBlaze.



Σχήμα 5.50 – Η αλυσίδα για την εκσφαλμάτωση του λογισμικού

Προτού προχωρήσουμε στην σύνδεση του MicroBlaze με τον XMD μεταβαίνουμε στην καρτέλα Applications του αριστερού παραθύρου του XPS και κάνουμε διπλό-κλικ στην εφαρμογή μας, MyProj, για να εμφανιστεί το παράθυρο Set Compiler settings for the project: MyProj. Εκεί στην καρτέλα Optimization επιλέγουμε Create Symbols for debugging (-g option) στον πλαίσιο Debug Options και πατάμε OK.



Σχήμα 5.51 – Επιλογές εκσφαλμάτωσης

Με την επιλογή αυτή η εκσφαλμάτωση που θα ακολουθήσει θα μπορεί να γίνει και σε επίπεδο κώδικα C, ενώ σε αντίθετη περίπτωση θα γινόταν μόνο σε επίπεδο assembly.

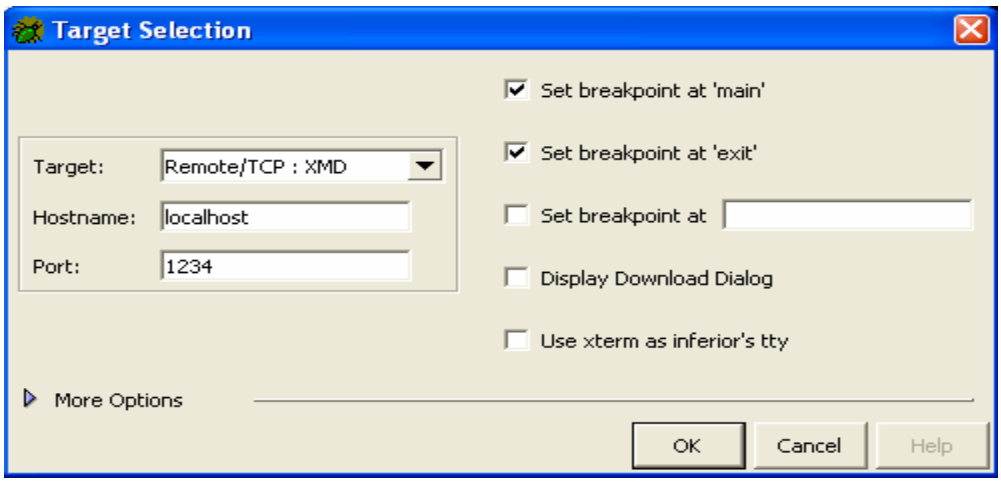
Έχοντας κατεβάσει το σύστημα στο FPGA, επιλέγουμε Tools → XMD και ανοίγει ένα παράθυρο τύπου DOS στο οποίο πληκτρολογούμε mbconnect ώστε να γίνει η σύνδεση του XMD με το σύστημα του MicroBlaze μέσω της θύρας JTAG. Αμέσως παγώνει η εκτέλεση του μετρητή στην οθόνη επτά τμημάτων και στο παράθυρο DOS εμφανίζονται οι πληροφορίες σύνδεσης του XMD με το σύστημα.

```
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 6.2.2 Build EDK_Gm.13.6
Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
XMD% mbconnect mdm
Connecting to cable (Parallel Port - LPT1).
Checking cable driver.
 Driver windrvr6.sys version = 6.0.3.0. LPT base address = 0378h.
 ECP base address = FFFFFFFFh.
Cable connection established.

JTAG chain configuration
--------------------------------------------------
Device    ID Code         IR Length    Part Name
 1        01414093            6        XC3S200
 2        05045093            8        XCF02S
Assuming, Device No: 1 contains the MicroBlaze system
Connected to the JTAG MicroBlaze Debug Module (MDM)
No of processors = 1

MicroBlaze Processor 1 Configuration :
--------------------------------------
Version.............................2.10.a
No of PC Breakpoints................2
No of Read Addr/Data Watchpoints...1
No of Write Addr/Data Watchpoints..1
Instruction Cache Support..........off
Data Cache Support..................off
JTAG MDM Connected to Mircoblaze 1
Connected to MicroBlaze "mdm" target. id = 0
Starting GDB server for "mdm" target (id = 0) at TCP port no 1234
XMD%
```

Σχήμα 5.52 – Πληροφορίες σύνδεσης του XMD με τον MDM

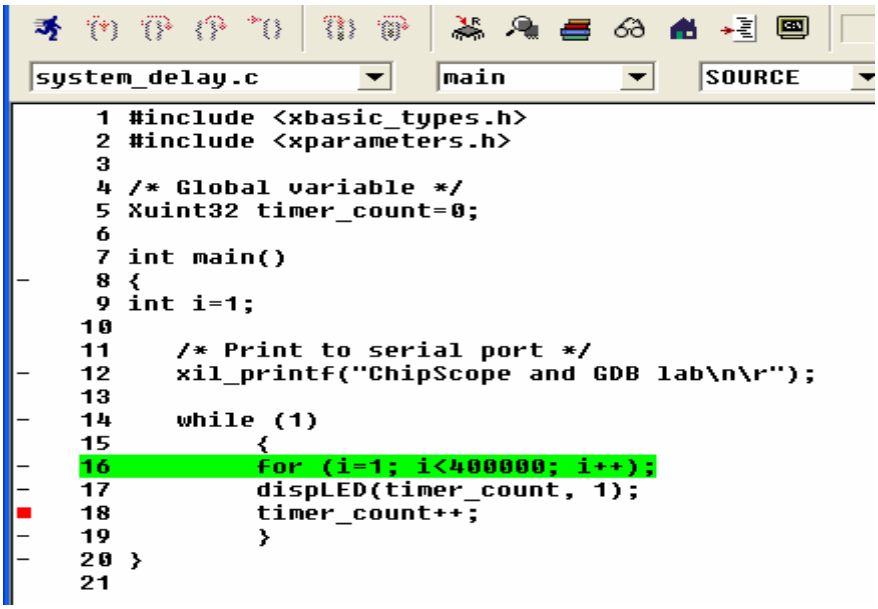
Όπως γίνεται φανερό ο XMD έχει ανιχνεύσει τόσο τον MDM όσο και τον MicroBlaze και παρατηρώντας προσεκτικά την προτελευταία γραμμή θα δούμε ότι μπορεί να δεχθεί συνδέσεις από τον GNU Debugger (GND) σε μία TCP θύρα με αριθμό 1234.

Με βάση τα στοιχεία αυτά επιστρέφουμε στο XPS και ανοίγουμε τον εκσφαλματωτή λογισμικού GNU Debugger επιλέγοντας Tools → Software Debugger και στο παράθυρο που αναδύεται επιλέγουμε την εφαρμογή MyProj. Στο νέο παράθυρο που εμφανίζεται, αναγράφεται ο κώδικας του αρχείου system_delay.c και η πρώτη εντολή της ρουτίνας main() είναι τονισμένη με μοβ χρώμα. Επιλέγουμε Run → Connect to target για να συνδέσουμε τον εκσφαλματωτή με τον XMD και στο παράθυρο διαλόγου Target Selection που εμφανίζεται εισάγουμε τις τιμές του σχήματος 5.53.



Σχήμα 5.53 – Οι επιλογές σύνδεσης του GND με τον XMD

Όπως γίνεται φανερό χρησιμοποιείται για την σύνδεση η θύρα με τον αριθμό 1234, την οποία επισημάναμε παραπάνω. Πατώντας OK ο κώδικας του λογισμικού κατεβαίνει πάλι στην κάρτα και η εκτέλεση του προγράμματος σταματά σε μια εντολή της ρουτίνας main(), που αυτήν τη φορά είναι τονισμένη με πράσινο χρώμα που σημαίνει ότι ο εκσφαλματωτής είναι συνδεδεμένος με το σύστημα. Πατώντας με το ποντίκι στα αριστερά της εντολής timer_count++ εισάγουμε ένα breakpoint.

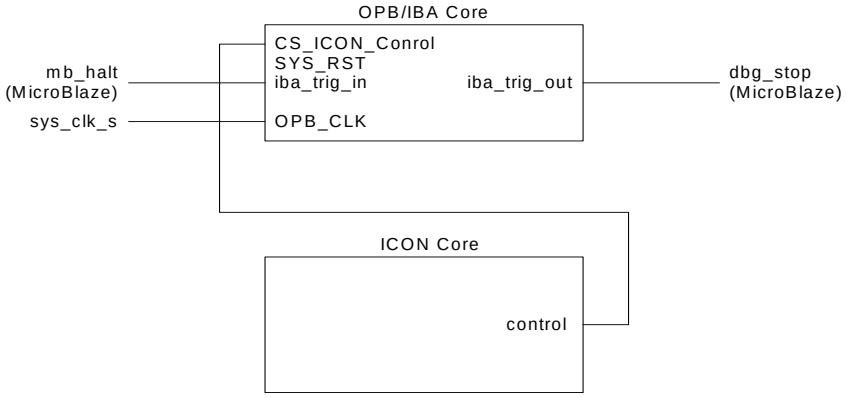


Σχήμα 5.54 – Εισαγωγή breakpoint

Παρά το ότι η οθόνη επτά τμημάτων δείχνει μία παλιά τιμή το πρόγραμμα έχει ξεκινήσει από την αρχή, απλά δεν έχει φτάσει στην εντολή dispLED(timer_count, 1) και επομένως δεν έχει ανανεωθεί η απεικόνιση. Πράγματι επιλέγοντας Control → Continue ο βρόγχος εκτελείται μέχρι την εντολή timer_count++ και μηδενίζεται η απεικόνιση. Κάθε φορά που επιλέγουμε Control → Continue εκτελείται μία φορά ο βρόγχος και η απεικόνιση αυξάνει κατά ένα. Εάν αφαιρέσουμε το breakpoint και συνεχίσουμε πάλι την εκτέλεση ο μετρητής αρχίζει να μετρά χωρίς να μπορούμε να τον σταματήσουμε

5.2.6.2.2 Εκσφαλμάτωση υλικού

Για τις ενέργειες της εκσφαλμάτωσης υλικού θα πρέπει να προσθέσουμε δύο νέα IPs στο σύστημά μας, το chipscope_icon και το chipscope_opb_iba. Και τα δύο είναι στοιχεία απαραίτητα για την λειτουργία του προγράμματος ChipScope Pro της Xilinx. Οι συνδέσεις των δύο αυτών στοιχείων φαίνονται στο σχήμα 5.55.



Σχήμα 5.55 – Οι συνδέσεις των στοιχείων του ChipScope

Επιλέγουμε Project → Add/Edit Cores και στην καρτέλα Peripherals προσθέτουμε τα δύο IPs, χωρίς να ορίζουμε διευθύνσεις αφού στην ουσία δεν αποτελούν ενεργό μέρος του συστήματος, αλλά είναι βοηθητικά στοιχεία για την ανάλυσή του. Στην καρτέλα Bus Connections συνδέουμε το στοιχείο chipscope_opb_iba στον διάδρομο δεδομένων OPB σαν BA, αναλυτή διαδρόμου δεδομένων (Bus Analyzer).

	mb_opb	ilmb	dlmb
microblaze_0 dlmb			M
microblaze_0 ilmb		M	
microblaze_0 dopb	M		
microblaze_0 iopb	M		
debug_module sopb	S		
debug_module sfsl0			
debug_module mfsl0			
dlmb_cntlr slmb			S
ilmb_cntlr slmb		S	
RS232 sopb	S		
opb_7segled_0 sopb	S		
chipscope_opb_iba_0 mon_opb	BA		

Σχήμα 5.56 – Σύνδεση του chipscope_opb_iba στον διάδρομο δεδομένων OPB

Στην καρτέλα Ports προσθέτουμε τις αναγκαίες θύρες και τις διαμορφώνουμε κατάλληλα με βάση το σχήμα 5.55. Το αποτέλεσμα φαίνεται στο σχήμα 5.57

chipscope_icon_0	control0	chipscope_control0	O	Internal
chipscope_opb_iba_0	OPB_Clk	sys_clk_s	I	Internal
chipscope_opb_iba_0	chipscope_icon_control	chipscope_control0	I	Internal
chipscope_opb_iba_0	iba_trig_in	mb_halted	I	Internal
chipscope_opb_iba_0	iba_trig_out	dbg_stop	O	Internal

Σχήμα 5.57 – Οι θύρες των στοιχείων του ChipScope

Στην καρτέλα Parameters επιλέγουμε το στοιχείο chipscope_icon_0 στο επάνω δεξιά κουτί επιλογής IP. Από τις παραμέτρους του που εμφανίζονται επιλέγουμε την SYSTEM_CONTAINS_MDM και πατάμε το << ADD για να την προσθέσουμε στην λίστα αριστερά. Θέτουμε την τιμή αυτής της παραμέτρου 1 για να δηλώσουμε την ύπαρξη του στοιχείου opb_mdm, το οποίο έχουμε συμπεριλάβει στο σύστημά μας και το οποίο έχουμε ήδη χρησιμοποιήσει. Στην συνέχεια επιλέγουμε το στοιχείο chipscope_opb_iba_0 στο επάνω δεξιά κουτί επιλογής IP και προσθέτουμε στην λίστα αριστερά τις παρακάτω παραμέτρους ορίζοντας και τις τιμές τους:

- C_NUM_DATA_SAMPLES = 512 (αριθμός δειγμάτων σε κάθε ερέθισμα, trigger)
- C_ENABLE_TRIGGER_OUT = 1 (επίτρεψη σήματος iba_trig_out στην τιμή λογικό ‘1’)
- C_CONTROL_UNITS = 1 (μία μονάδα σημάτων ελέγχου)
- C_ADDR_UNITS = 1 (μία μονάδα διευθύνσεων)
- C_DATA_UNITS = 1 (μία μονάδα δεδομένων)
- C_GENERIC_TRIGGER_UNITS = 1 (μία μονάδα ερεθισμάτων, trigger)
- C_TRIGGER_UNIT_MATCH_TYPE = basic with edges (επιλογή τύπου σύγκρισης γεγονότων για τον καθορισμό της ύπαρξης ή όχι ερεθίσματος, η basic with edges προσφέρει ανίχνευση μετάβασης στην τιμή του ερεθίσματος)
- C_GENERIC_TRIGGER_IN_WIDTH = 1 (το μήκος του διαδρόμου δεδομένων για τα ερεθίσματα, εμείς έχουμε μόνο ένα)
- C_DISABLE_RPM = 1

Η παράμετρος C_DISABLE_RPM χρησιμοποιείται ειδικά για το Spartan-3, άλλωστε δεν υπάρχει καν στο βιβλίο δεδομένων του chipscope_opb_iba, και εισάγεται για να αποτρέψει λάθη που προκύπτουν στην σχεδίαση (mapping), όπως μας υποδεικνύουν οι σημειώσεις για την έκδοση του EDK.

Πατώντας OK έχει επιτευχθεί η σύνδεση των στοιχείων του ChipScope στο σύστημα, όμως στην καρτέλα Ports ορίσαμε τα σήματα των θυρών iba_trig_in και iba_trig_out, mb_halted και dbg_stop αντίστοιχα, χωρίς να έχουμε ορίσει την προέλευση και τον προορισμό αυτών των σημάτων. Τα σήματα αυτά πρέπει να συνδεθούν με τον MicroBlaze. Για τον λόγο αυτό ανοίγουμε το αρχείο MHS, system.mhs, του συστήματος και στις θύρες του MicroBlaze προσθέτουμε τις:

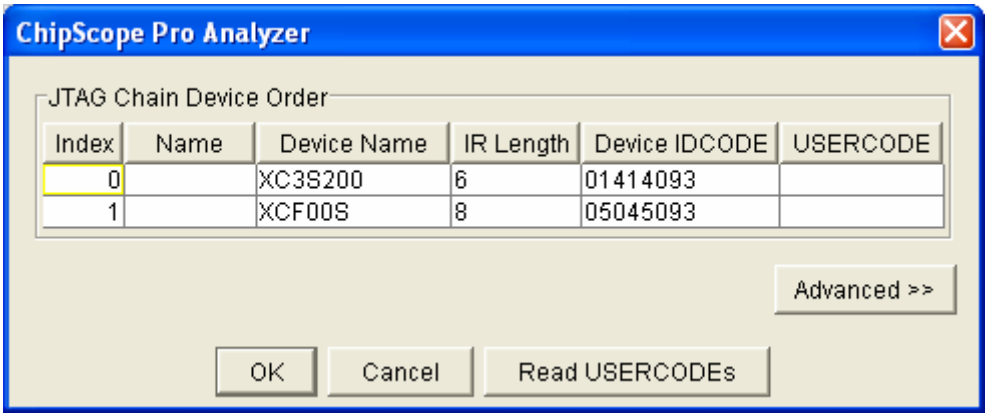
- PORT MB_Halted = mb_halted
- PORT DBG_STOP = dbg_stop

```
026  BEGIN microblaze
027   PARAMETER INSTANCE = microblaze_0
028   PARAMETER HW_VER = 2.10.a
029   PARAMETER C_DEBUG_ENABLED = 1
030   PARAMETER C_NUMBER_OF_PC_BRK = 2
031   PARAMETER C_NUMBER_OF_RD_ADDR_BRK = 1
032   PARAMETER C_NUMBER_OF_WR_ADDR_BRK = 1
033   BUS_INTERFACE DOPB = mb_opb
034   BUS_INTERFACE IOPB = mb_opb
035   BUS_INTERFACE DLMB = dlmb
036   BUS_INTERFACE ILMB = ilmb
037   PORT CLK = sys_clk_s
038   PORT DBG_CAPTURE = DBG_CAPTURE_s
039   PORT DBG_CLK = DBG_CLK_s
040   PORT DBG_REG_EN = DBG_REG_EN_s
041   PORT DBG_TDI = DBG_TDI_s
042   PORT DBG_TDO = DBG_TDO_s
043   PORT DBG_UPDATE = DBG_UPDATE_s
044   PORT MB_Halted = mb_halted
045   PORT DBG_STOP = dbg_stop
046  END
```

Σχήμα 5.58 – Προσθήκη των θυρών του MicroBlaze

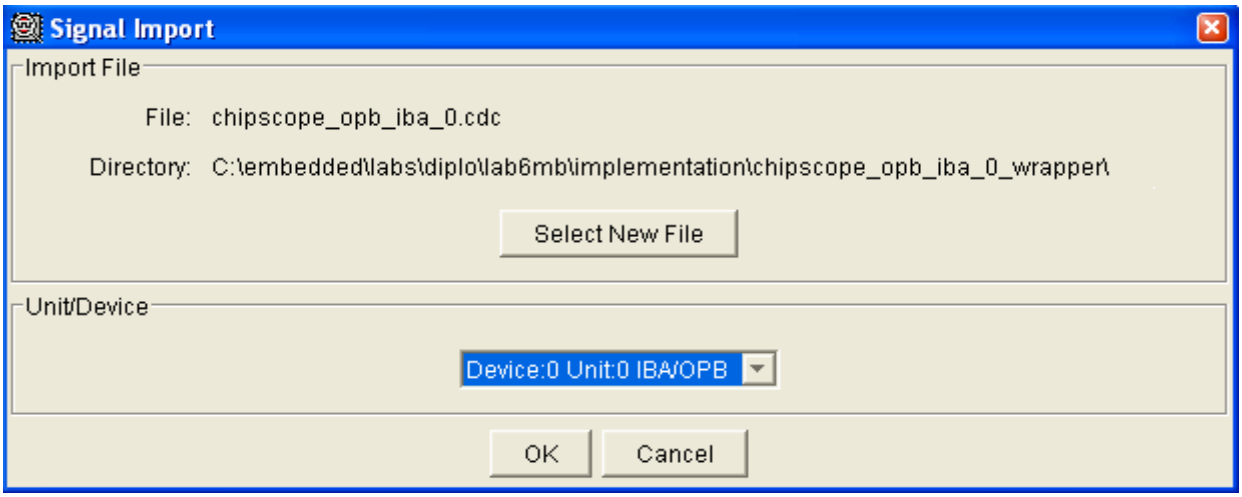
Σώζουμε και κλείνουμε το αρχείο MHS.

Επιλέγουμε Tools → Download και αναμένουμε όλες τις διαδικασίες σύνθεσης και υλοποίησης μέχρι το ολοκληρωμένο σύστημα να κατέβει στο FPGA. Στο σημείο αυτό τελειώνει ο ρόλος του XPS και ανοίγουμε το πρόγραμμα ChipScope Pro Analyzer. Στο περιβάλλον εργασίας του πατάμε το εικονίδιο Open Cable/Search JTAG Chain  . Με τον τρόπο αυτό ανιχνεύονται οι συσκευές που βρίσκονται στην αλυσίδα JTAG.



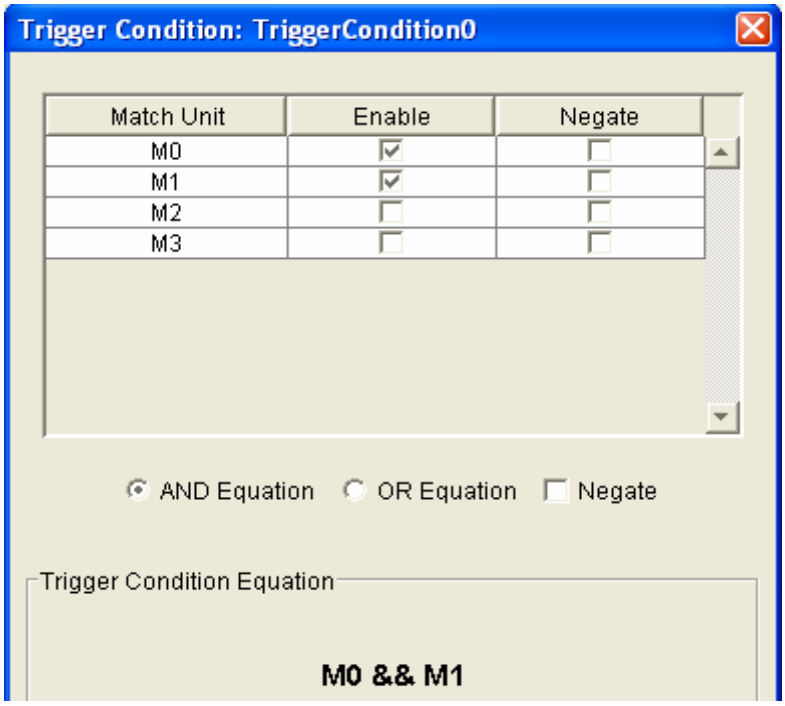
Σχήμα 5.59 – Ανίχνευση συσκευών στην αλυσίδα JTAG

Πατάμε OK. Το ChipScope Pro Analyzer ανοίγει δύο παράθυρα, το Trigger Setup και το Waveform. Οι ρυθμίσεις των παραμέτρων του chipscope_opb_iba που ορίσαμε στο XPS και οι ομάδες σημάτων βρίσκονται στους φακέλους του project με την μορφή ενός αρχείου CDC. Επιλέγουμε File → Import και στο παράθυρο διαλόγου Signal Import που αναδύεται περιηγούμαστε στον φάκελο \implementation\chipscope_opb_iba_0_wrapper και επιλέγουμε την εισαγωγή των απαραίτητων σημάτων από το αρχείο chipscope_opb_iba_0.cdc.



Σχήμα 5.60 – Εισαγωγή σημάτων στο ChipScope

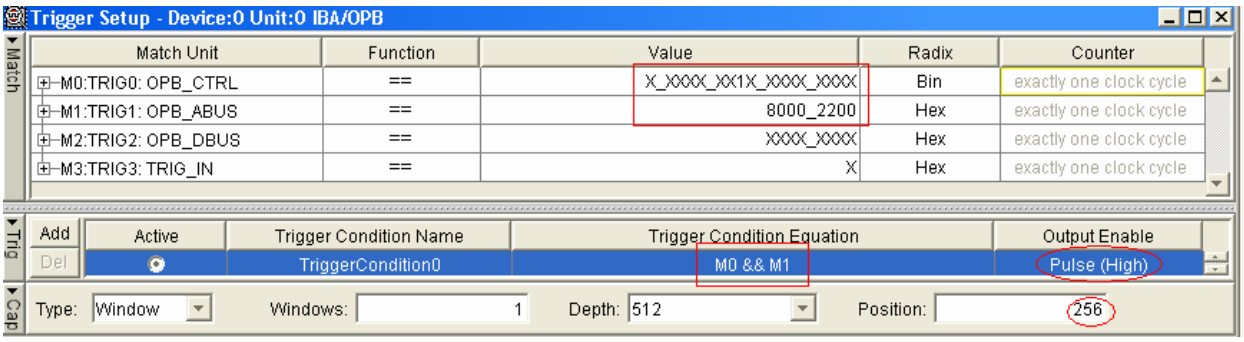
Τα σήματα του συστήματος εμφανίζονται στα παράθυρα Trigger Setup και Waveform. Στο παράθυρο Trigger Setup αλλάζουμε την βάση, Radix, των M1, M2 και M3 από δυαδική (Bin) σε δεκαεξαδική (Hex). Θέτουμε στο σήμα M1:TRIG1:OPB_ABUS την τιμή 80002200 που αντιστοιχεί στην διεύθυνση βάσης του περιφερειακού που ελέγχει την οθόνη επτά τμημάτων. Αναπτύσσοντας το δέντρο των σημάτων M0:TRIG1:OPB_CTRL θέτουμε το bit του OPB_xferAck στο λογικό '1'. Το bit αυτό αντιστοιχεί στην επιβεβαίωση της μεταφοράς δεδομένων. Κάτω από τα σήματα πατάμε επάνω στο πεδίο Trigger Condition Equation και μαζί με το M0 που είναι προεπιλεγμένο επιλέγουμε και το M1, αφού και σε αυτό έχουμε δώσει τιμή.



Σχήμα 5.61 – Ορισμός συνάρτησης ερεθίσματος

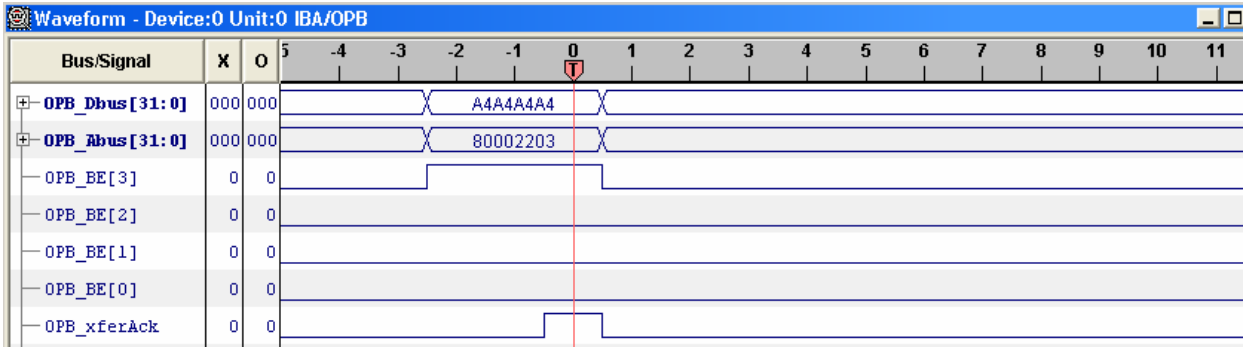
Αν η συνάρτηση M0 && M1 κατά την διάρκεια της εκτέλεσης του συστήματος ισούται με το αποτέλεσμα της ίδιας συνάρτησης με βάση τα σήματα που έχουμε ορίσει εμείς, τότε προκαλείται ερέθισμα. Στον ίδιο πίνακα και στο πεδίο Output Enable επιλέγουμε Pulse (High), που σημαίνει ότι δέκα κύκλους μετά το συμβάν που έχει προκαλέσει το ερέθισμα ένας παλμός ρολογιού με τιμή λογικό '1' θα οδηγηθεί στην έξοδο του στοιχείου του ChipScope. Στον

τελευταίο πίνακα του παραθύρου Trigger Setup επιλέγουμε στο πεδίο Window πόσα γεγονότα που προκαλούν ερέθισμα θέλουμε να αναλύσουμε, εμείς επιλέγουμε ένα, στο πεδίο Depth πόσα δείγματα δεδομένων θέλουμε να πάρουμε, 512, και στο πεδίο Position σε πιο δείγμα θέλουμε να αντιστοιχεί το ερέθισμα. Συγκεντρωτικά οι αλλαγές που κάνουμε στο παράθυρο Trigger Setup φαίνονται στο σχήμα 5.62.



Σχήμα 5.62 – Οι ρυθμίσεις που αφορούν το ερέθισμα και την λήψη των δειγμάτων

Προτού ξεκινήσουμε την ανάλυση, αναμένοντας το ερέθισμα, οργανώνουμε λίγο τα σήματα του παραθύρου Waveform για να μπορούμε να τα παρατηρήσουμε καλύτερα. Κρατώντας πατημένο το Shift επιλέγουμε τα σήματα από OPB_Abus [31] έως OPB_Abus[0]. Κάνουμε δεξί-κλικ και επιλέγουμε Add to Bus → New Bus. Κάνουμε δεξί-κλικ στον BUS_0 που προκύπτει, επιλέγουμε Rename και τον μετονομάζουμε σε OPB_Abus[31:0]. Την ίδια διαδικασία ακολουθούμε για να δημιουργήσουμε και τον διάδρομο δεδομένων OPB_Dbus[31:0]. Επιλέγοντας Trigger Setup → Run, μετά από μια μικρή παύση που αντιστοιχεί στην ανάλυση και την λήψη δειγμάτων από τα σήματα του διαδρόμου δεδομένων OPB, στο παράθυρο Waveform εμφανίζονται η διεύθυνση που κρατάει την τιμή για τα επτά τμήματα του δεξιότερου ψηφίου (80002200) της οθόνης επτά τμημάτων και η τιμή αυτή, η οποία εμφανίζεται τέσσερις φορές, αφού έχουμε μεταφορά δεδομένων οκτώ bit σε έναν διάδρομο δεδομένων των 32 bit οπότε απλώς στα σήματα του διαδρόμου που δεν χρησιμοποιούνται αντιγράφεται η ίδια πληροφορία. Εκτελώντας πολλές φορές Run διαπιστώνουμε ότι, όπως ήταν αναμενόμενο όλες οι τιμές ανήκουν στον πίνακα Digit[] που ορίσαμε στο αρχείο πηγαίου κώδικα 7segled.c. Συνεχίζοντας και επειδή το δεξιότερο ψηφίο αλλάζει πολύ γρήγορα για να μπορούμε να διαπιστώσουμε αν είναι σωστές οι τιμές που παίρνουμε, μπορούμε να αλλάξουμε την διεύθυνση με αυτήν του καταχωρητή που ελέγχει το αριστερότερο ψηφίο, 80002203, και πράγματι παρατηρούμε ότι τα δεδομένα που εμφανίζονται συμφωνούν με το ψηφίο που εμφανίζεται στην οθόνη επτά τμημάτων. Αξιοσημείωτο είναι επίσης το γεγονός ότι το σήμα OPB_BE[3] έχει την τιμή λογικό '1' κατά την μεταφορά των δεδομένων, κάτι που σημαίνει ότι μόνο τα 8 LSBit του διαδρόμου δεδομένων περιέχουν έγκυρα δεδομένα.



Σχήμα 5.63 – Οι κυματομορφές που δημιουργεί ο ChipScope Pro Analyzer

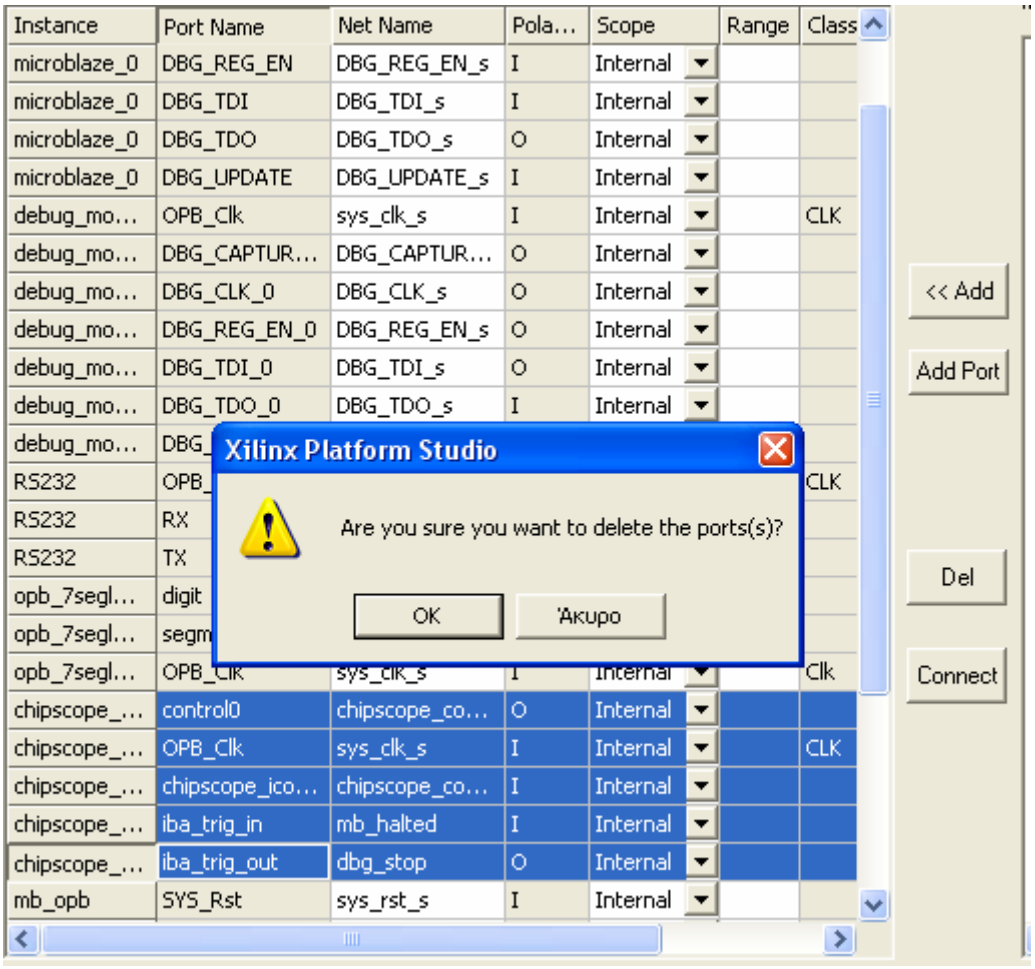
5.3 Εφαρμογή $2^{η}$

5.3.1 Υλοποίηση ψηφιακού φίλτρου έξι σημείων

Στην δεύτερη εφαρμογή θα υλοποιήσουμε ένα ψηφιακό φίλτρο έξι σημείων χρησιμοποιώντας την κάρτα Spartan-3 Starter Kit. Στην παρούσα φάση δεν μας ενδιαφέρει το είδος του φίλτρου, καθώς το μόνο που καθορίζει την λειτουργία του είναι οι τιμές των έξι συντελεστών με τους οποίους θα πολλαπλασιάζει τα δεδομένα εισόδου του. Η συνάρτηση του προς υλοποίηση φίλτρου είναι:

$y_n = a_0 * x_n + a_1 * x_{n-1} + a_2 * x_{n-2} + a_3 * x_{n-3} + a_4 * x_{n-4} + a_5 * x_{n-5}$ $n = 0,1,2,3,4,\ldots$

Η πλατφόρμα υλικού στην οποία θα προστεθεί το φίλτρο μας είναι αυτή που χρησιμοποιήθηκε στο τελευταίο βήμα της πρώτης εφαρμογής, αφού πρώτα αφαιρέσουμε τα στοιχεία του Chipscope. Επομένως επιλέγουμε Project → Add/Edit Cores και στην καρτέλα Ports επιλέγουμε όλες τις θύρες των στοιχείων του Chipscope και πατάμε Del, ενώ το ίδιο κάνουμε και για τις θύρες του MicroBlaze, MB_Halted και DBG_STOP.

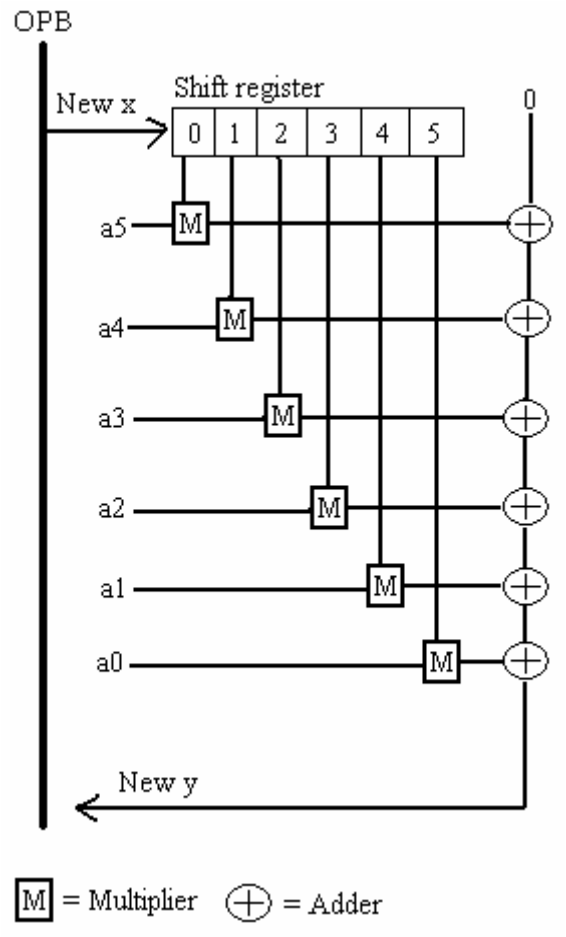


Σχήμα 5.64 – Διαγραφή των θυρών του Chipscope, ενώ έχει προηγηθεί αυτή των δύο θυρών του MicroBlaze

Στην συνέχεια στην καρτέλα Bus Connections αποσυνδέουμε το στοιχείο chipscope_opb_iba_0 mon_opb από τον διάδρομο δεδομένων OPB και τέλος στην καρτέλα Peripherals αφαιρούμε τα δύο στοιχεία του Chipscope από το σύστημα. Συνεπώς έχοντας, πλέον, έτοιμο το βασικό

σύστημα θα μας απασχολήσει κυρίως η περιγραφή του φίλτρου και η προσθήκη του στην ήδη υπάρχουσα πλατφόρμα υλικού.

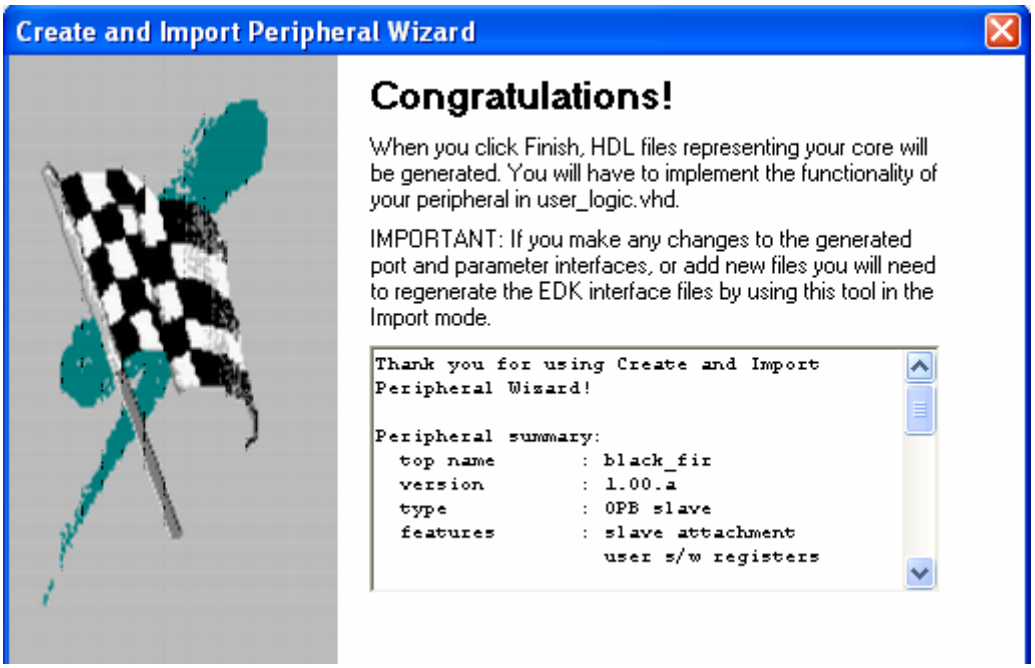
Για την ταχύτερη λειτουργία του φίλτρου προτιμήθηκε να χρησιμοποιηθούν οι έξι από τους δεκαοκτώ hardware πολλαπλασιαστές της κάρτας Spartan-3 Starter Kit. Η αρχιτεκτονική του φίλτρου φαίνεται στο σχήμα 5.65.



Σχήμα 5.65 – Η αρχιτεκτονική του φίλτρου

Όπως γίνεται κατανοητό για την αποθήκευση των εισόδων του φίλτρου χρησιμοποιείται ένα τμήμα ολίσθησης καταχωρητών, έξι θέσεων, όπου όταν έρχεται ένα νέο δεδομένο τα υπόλοιπα ολισθαίνουν κατά μία θέση, με το παλαιότερο δεδομένο να απορρίπτεται. Με την είσοδο του νέου δεδομένου τα μερικά γινόμενα από τους έξι πολλαπλασιαστές αλλάζουν και το ίδιο συμβαίνει και με την έξοδο του φίλτρου.

Το φίλτρο αυτό θα συνδεθεί με τον διάδρομο δεδομένων του MicroBlaze σαν ένα νέο περιφερειακό. Το περιφερειακό δεν θα έχει εξωτερικές θύρες, παρά θα συνδέεται μόνον εσωτερικά με το σύστημα, και θα χρησιμοποιεί δύο καταχωρητές των 32 bit, τον έναν για την αποστολή των νέων δεδομένων και τον άλλον για την επιστροφή της νέας εξόδου του φίλτρου. Για τον σκοπό αυτό, όπως έχουμε δείξει και στην πρώτη εφαρμογή, εκκινούμε τον βοηθό εισαγωγής περιφερειακού (Tools → Import Peripheral Wizard), ονομάζουμε το περιφερειακό black_fir, ορίζουμε δύο καταχωρητές μεγέθους 32 bit ελεγχόμενους από το λογισμικό ως υπηρεσία της διεπαφής IPIF και ολοκληρώνουμε τον βοηθό.



Σχήμα 5.66 – Περίληψη του περιφερειακού που θα ελέγχει το φίλτρο

Εφόσον το περιφερειακό δεν χρησιμοποιεί εξωτερικές θύρες, δεν χρειάζεται να πραγματοποιήσουμε εγγραφές στα αρχεία black_fir.vhd και black_fir_v2_1_0.mpd, ενώ στο αρχείο user_logic.vhd θα αντικαταστήσουμε μόνο το τμήμα architecture. Επειδή η λειτουργία του φίλτρου που πρέπει να περιγράψουμε είναι πιο πολύπλοκη σε σχέση με ό,τι έχουμε κάνει έως τώρα, η περιγραφή δεν θα γίνει μέσα στο user_logic.vhd αλλά θα χρησιμοποιηθούν τέσσερα άλλα τμήματα κώδικα (components). Το user_logic.vhd θα κάνει χρήση του top-level αυτών. Αναλυτικότερα τα components είναι:

- reg.vhd. Το όνομα του component είναι reg. Είναι ένας καταχωρητής μεγέθους 16 bit που σε κάθε παλμό ρολογιού, και εάν το σήμα επίτρεψης που δέχεται έχει τιμή λογικό ‘1’, βγάζει στην έξοδο τα σήματα που εφαρμόζονται στην είσοδό του.
- shift_reg.vhd. Όνομα component reg_port. Χρησιμοποιεί έξι components reg προκειμένου να ολισθαίνει τις θέσεις των καταχωρητών με κάθε παλμό ρολογιού, και εφόσον το σήμα επίτρεψης που υποδεικνύει ότι υπάρχει νέο δεδομένο έχει τιμή λογικό ‘1’. Εκτός από την νέα είσοδο και το σήμα επίτρεψης το component reg_port έχει έξι εξόδους των 16 bit, όσοι και οι καταχωρητές που παίρνουν μέρος στην διαδικασία της ολίσθησης.
- mac.vhd. Όνομα component mac. Δεν χρησιμοποιεί άλλα components. Δέχεται σαν είσοδο δύο αριθμούς, έναν των 16 bit που αντιστοιχεί στα δεδομένα της εισόδου του φίλτρου και έναν των 12 bit που αντιστοιχεί στους συντελεστές του φίλτρου, και ένα σήμα επίτρεψης για την εκκίνηση του mac. Οι δύο αριθμοί πολλαπλασιάζονται και το αποτέλεσμα προστίθεται με μία είσοδο των 32 bit, η οποία παρακάτω θα αντιστοιχηθεί με το μέχρι εκείνη την στιγμή άθροισμα των μερικών γινομένων του φίλτρου. Η έξοδος του mac είναι το αποτέλεσμα της πρόσθεσης μεταξύ του μερικού γινομένου και της εισόδου των 32 bit. Η πράξη του πολλαπλασιασμού περιγράφεται με τον τελεστή ‘*’, και όχι με κάποια συγκεκριμένη αρχιτεκτονική, προκειμένου το εργαλείο σύνθεσης του XPS να το αντιστοιχήσει με έναν hardware πολλαπλασιαστή του FPGA της κάρτας Spartan-3 Starter Kit.
- fir.vhd. Το όνομα του component είναι fir. Είναι το top-level αρχείο του φίλτρου, όχι του περιφερειακού που είναι το black_fir.vhd, και χρησιμοποιεί ένα component reg_port και έξι mac. Λαμβάνει σαν είσοδο το νέο δεδομένο του φίλτρου και το σήμα επίτρεψης για την ολίσθηση του reg_port. Αντιστοιχίζει τους ολισθημένους καταχωρητές σε έξι mac και με την ολοκλήρωση των πράξεων επιστρέφει την νέα έξοδο του φίλτρου.

Όλα τα παραπάνω components περιλαμβάνουν επιπροσθέτως μία είσοδο ρολογιού και μία είσοδο καθαρισμού που μηδενίζει όλα τα σήματα. Τα τέσσερα παραπάνω αρχεία VHDL βρίσκονται στο παράρτημα στο τέλος του παρόντος συγγράμματος.

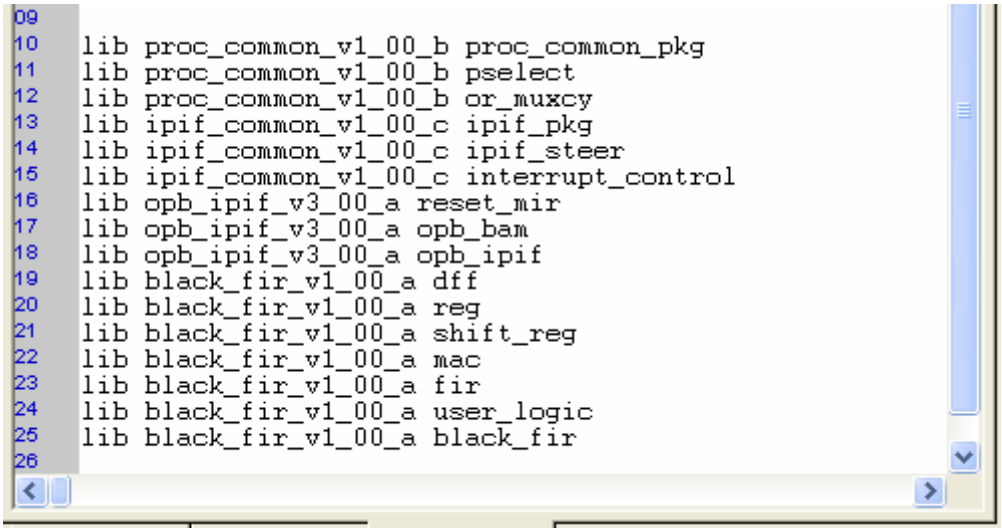
Το user_logic.vhd χρησιμοποιεί το component fir. Όπως έχουμε εξηγήσει και στην πρώτη εφαρμογή όταν το περιφερειακό λαμβάνει ένα νέο δεδομένο από την εφαρμογή λογισμικού του MicroBlaze, τότε ένα σήμα παίρνει τιμή λογικό '1' για να προαναγγείλει την συγκεκριμένη άφιξη. Το σήμα αυτό το περνάμε ως παράμετρο στο component fir για να ενεργοποιήσει την ολίσθηση, αφού έχει έρθει νέο δεδομένο, αλλά και να εκκινήσει τα mac. Για λόγους σταθεροποίησης της τιμής του νέου δεδομένου προσθέτουμε μία καθυστέρηση ενός παλμού ρολογιού στο παραπάνω σήμα προτού το περάσουμε στο fir. Πιο αναλυτικά ο κώδικας για το τμήμα architecture του user_logic.vhd ακολουθεί:

```
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

------------------------------------------------------------
-- Declaration of the flip-flop and fir components
------------------------------------------------------------
component dff port (
                        d       :       in std_logic;
                        clear   :       in std_logic;
                        clk     :       in std_logic;
                        q       :       out std_logic
                          );
end component;

component fir port (
                        xnew:    in std_logic_vector(0 to 15);
                        clk      : in std_logic;
                        clr      : in std_logic;
                        wr       : in std_logic;
                        ynew     : out std_logic_vector (0 to 31)
                          );
end component;



    ---------------------------------------------------------------
    -- Signals and s/w accessible registers for user logic
    ---------------------------------------------------------------
    signal slv_reg0                 : std_logic_vector(0 to C_DWIDTH-1);  --new x
    signal slv_reg1                 : std_logic_vector(0 to C_DWIDTH-1);  --new y
    signal enable                   : std_logic; -- the enable signal for the shift through the dff
    signal xnew                     : std_logic_vector(0 to 15); -- the signal that drives the new x
    signal ynew                     : std_logic_vector(0 to 31); -- the signal that drives the new y
    signal slv_reg_write_select     : std_logic_vector(0 to 1); -- the signal for the write action
    signal slv_reg_read_select      : std_logic_vector(0 to 1); -- the signal for the read action
    signal slv_ip2bus_data          : std_logic_vector(0 to C_DWIDTH-1);
    signal slv_read_ack             : std_logic; -- the overall read acknowledge
    signal slv_write_ack            : std_logic; -- the overall write acknowledge

begin

    ---------------------------------------------------------------
    -- Map user logic S/W register read/write select signal
    ---------------------------------------------------------------
    slv_reg_write_select <= Bus2IP_WrCE(0 to 1);
```

```
    slv_reg_read_select  <= Bus2IP_RdCE(0 to 1);
    slv_write_ack        <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1);
    slv_read_ack         <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1);

    ---------------------------------------------------------------------------
    -- User logic S/W accessible registers write action for the new x
    ---------------------------------------------------------------------------
    SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
    begin

      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
        if Bus2IP_Reset = '1' then
          slv_reg0 <= (others => '0');
        else
          case slv_reg_write_select is
            when "10" =>
              for byte_index in 0 to (C_DWIDTH/8)-1 loop
                if ( Bus2IP_BE(byte_index) = '1' ) then
                  slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
                end if;
              end loop;
                    when others => null;
            end case;
          end if;
      end if;

    end process SLAVE_REG_WRITE_PROC;

    ---------------------------------------------------------------------------
    -- User logic S/W accessible registers read action for the new y
    ---------------------------------------------------------------------------
    SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg1) is
    begin

        case slv_reg_read_select is
                when "01" => slv_ip2bus_data <= slv_reg1;
                when others => slv_ip2bus_data <= (others => '0');
        end case;

    end process SLAVE_REG_READ_PROC;


-------------------------------------------------------------------------------
-- One clock delay for the enable signal and new data for the filter
-------------------------------------------------------------------------------
dffen:    dff port map (d => Bus2IP_WrCE(0), clear => Bus2IP_Reset, clk => Bus2IP_Clk, q => enable);
firmac:   fir port map (xnew => xnew, clk => Bus2IP_Clk, clr => Bus2IP_Reset, wr => enable, ynew => ynew);

    ----------------------------------------
    -- Code to drive IP to Bus signals
    ----------------------------------------
    IP2Bus_Data                 <= slv_ip2bus_data;

    xnew                        <= slv_reg0(16 το 31); -- 16 LSBits
    slv_reg1                    <= ynew;

    IP2Bus_Ack          <= slv_write_ack or slv_read_ack;
    IP2Bus_Error        <= '0';
    IP2Bus_Retry        <= '0';
    IP2Bus_ToutSup      <= '0';

end IMP;
```

Σημειώνουμε εδώ ότι, όπως φαίνεται και από τον κώδικα, επειδή έχουμε επιλέξει καταχωρητές των 32 bit, η νέα είσοδος του φίλτρου χρησιμοποιεί μόνο τα 16 LSBit του καταχωρητή που μεταφέρει τα νέα δεδομένα. Σώζουμε και κλείνουμε το αρχείο.

Όπως είδαμε εμείς έχουμε γράψει σε διαφορετικά αρχεία τα απαραίτητα components, όμως το XPS δεν το γνωρίζει εκτός εάν εμείς του το δηλώσουμε. Για τον σκοπό αυτό ανοίγουμε το αρχείο black_fir_v2_1_0.pao που βρίσκεται στον ίδιο φάκελο με το black_fir_v2_1_0.mpd. Το αρχείο PAO (Peripheral Analysis Order) απαριθμεί τα αρχεία και τις βιβλιοθήκες που πρέπει να συνδεθούν με την περιγραφή του περιφερειακού. Όπως βλέπουμε στο αρχείο αυτό έχουν αυτομάτως συμπεριληφθεί τα user_logic και black_fir, επομένως εμείς δεν έχουμε παρά να προσθέσουμε τα υπόλοιπα αρχεία, με σειρά χρήσης προχωρώντας προς το top-level, ώστε να προκύψει το αρχείο του σχήματος 5.67, και να σώσουμε το αρχείο.



Σχήμα 5.67 – Το αρχείο PAO

Αυτό που απομένει για να ολοκληρωθεί η προετοιμασία της πλατφόρμας υλικού είναι να συνδέσουμε το περιφερειακό με το υπόλοιπο σύστημα χρησιμοποιώντας το παράθυρο διαλόγου Add/Edit Hardware Platform Specifications. Επιλέγουμε λοιπόν Project → Add/Edit Cores και στην καρτέλα Peripherals προσθέτουμε το στοιχείο black_fir και του δίνουμε διεύθυνση βάσης την 0x80002300 και high address την 0x800023ff. Στην καρτέλα Bus Connections συνδέουμε το περιφερειακό μας στον διάδρομο δεδομένων OPB και στην καρτέλα Ports, αφού δεν διαθέτει εξωτερικές θύρες, εισάγουμε μόνον την θύρα OPB_Clk την οποία ονομάζουμε sys_clk_s και την μετατρέπουμε σε εσωτερική, Internal. Πατώντας OK για να αποδεχτούμε τις αλλαγές έχουμε ολοκληρώσει την επεξεργασία της πλατφόρμας υλικού.

Το υλικό συνεπώς που υλοποιεί το φίλτρο έχει περιγραφεί λεπτομερώς και απομένει μία εφαρμογή λογισμικού η οποία θα περνάει στο υλικό το νέο δεδομένο και θα διαβάζει από αυτό το αποτέλεσμα του φίλτρου. Για τον σκοπό αυτό μεταβαίνουμε στην καρτέλα Applications του αριστερού παραθύρου του XPS και πιο συγκεκριμένα στην ήδη υπάρχουσα εφαρμογή MyProj. Εκεί αφαιρούμε από τα αρχεία πηγαίου κώδικα της εφαρμογής (Sources) το system_delay.c, το οποίο δεν μας χρειάζεται πια, και ανοίγουμε ένα νέο έγγραφο προς επεξεργασία. Στον κώδικα που θα γράψουμε σε αυτό το έγγραφο θα καθορίσουμε, εκτός από την μέθοδο ελέγχου του φίλτρου, και τον τρόπο με τον οποίο θα γίνεται η επικοινωνία του συστήματος με έναν προσωπικό υπολογιστή μέσω της σειριακής θύρας. Η επικοινωνία αυτή είναι απαραίτητη για την παρουσίαση της λειτουργίας του φίλτρου που θα ακολουθήσει. Μέσω της σειριακής θύρας θα μεταφέρονται δεδομένα με την μορφή ενός byte και εμείς θα καλούμαστε να τα λαμβάνουμε και να ανακτούμε την πραγματική τους σημασία. Έτσι για το νέο δεδομένο του φίλτρου, που είναι μεγέθους 16 bit, λαμβάνουμε δύο byte με πρώτα τα οκτώ MSBits, από αυτά συνθέτουμε τον αριθμό, τον αποστέλλουμε στο hardware του φίλτρου, λαμβάνουμε την έξοδο του φίλτρου,

την χωρίζουμε σε τέσσερα bytes και τα αποστέλλουμε μέσω της σειριακής θύρας, με πρώτα τα οκτώ MSBits του αριθμού. Ο κώδικας σε C που περιγράφει την παραπάνω διαδικασία είναι αυτός που ακολουθεί:

```
#include <xbasic_types.h>
#include <xparameters.h>
#include <xuartlite_l.h>

int main()
{
Xuint16  xnew; /* new data */
Xuint16  xnewmon;  /* the 8 LSBits of the new data read from the serial port */
Xuint16  xnewdek;  /* the 8 MSBits of the new data read from the serial port */
Xuint32  ynew=0;  /* the new output of the filter */
Xuint8   y11=0;     /* the 8 MSBits (0 to 7) of ynew */
Xuint8   y12=0;    /* bits 8 to 15 of ynew */
Xuint8   y21=0;   /* bits 16 to 23 of ynew */
Xuint8   y22=0;   /* the 8 LSBits (24 to 31) of ynew */
Xuint32  *firc;  /* pointer to the base address of the filter */

    firc = (Xuint32 *)XPAR_BLACK_FIR_0_BASEADDR; /* the pointer's value */
    dispLED(ynew); /* clean the display */

    while(1)  /* infinite loop */
    {
        xnewdek = XUartLite_RecvByte(XPAR_RS232_BASEADDR);  /* read byte */
        xnewdek = (xnewdek << 8);  /* the 8 MSBits of xnew */
        xnewmon = XUartLite_RecvByte(XPAR_RS232_BASEADDR);  /* read byte */
        xnew = xnewdek + xnewmon;  /* the creation of the new data */

        *firc = xnew;  /* send the data to the filter */
        dispLED(xnew); /* show the new data on the display, we don't care about the DP*/
        ynew = *(firc + 1); /* read the output of the filter */

        y11 = (Xuint8)(ynew >> 24);  /* isolate the 8 MSBits of ynew */
        y12 = (Xuint8)(ynew >> 16);  /* 8 to 15 */
        y21 = (Xuint8)(ynew >> 8);   /* 16 to 23 */
        y22 = (Xuint8)(ynew);        /* isolate the 8 LSBits of ynew */

        XUartLite_SendByte(XPAR_RS232_BASEADDR, y11);  /* send byte */
        XUartLite_SendByte(XPAR_RS232_BASEADDR, y12);
        XUartLite_SendByte(XPAR_RS232_BASEADDR, y21);
        XUartLite_SendByte(XPAR_RS232_BASEADDR, y22);
    }
}
```

Ο παραπάνω κώδικας γράφτηκε με τέτοιο τρόπο ώστε να επικοινωνεί αρμονικά με το LabVIEW της National Instruments, το οποίο χρησιμοποιήθηκε στην παρουσίαση της λειτουργίας του φίλτρου που θα ακολουθήσει. Σώζουμε των κώδικα ως system.c και επισυνάπτουμε το αρχείο (Add File) στην εφαρμογή MyProj, όπου ήδη υπάρχει το 7segled.c.

Έχοντας ολοκληρώσει την περιγραφή τόσο του υλικού όσο και του λογισμικού, αν επιλέξουμε Tools → Download το σύστημα θα υλοποιηθεί από το XPS και θα κατέβει στο FPGA έτοιμο για χρήση.

5.3.2 Παρουσίαση της λειτουργίας του φίλτρου

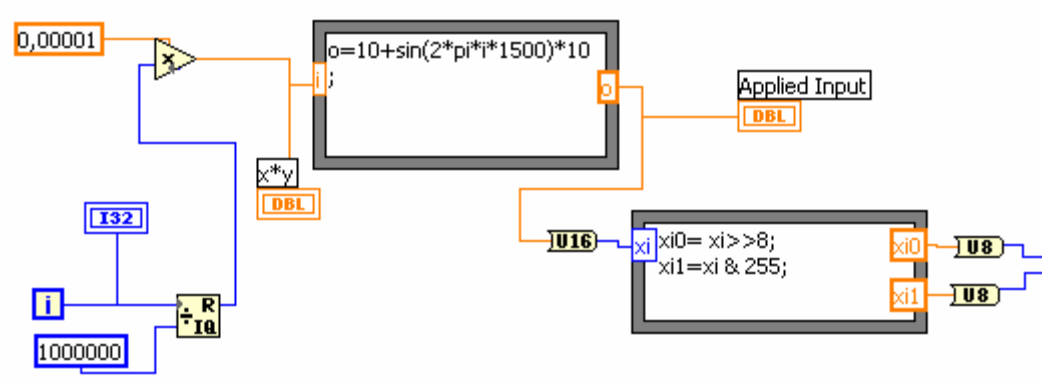
Η παρουσίαση, και στην ουσία η επιβεβαίωση, της λειτουργίας του φίλτρου έγινε με την βοήθεια του MATLAB της MathWorks και του LabVIEW της National Instruments. Η περιγραφή των προγραμμάτων αυτών ξεφεύγει από τα πλαίσια του παρόντος συγγράμματος και για τον λόγο αυτό θα αναφέρουμε μόνο όσα είναι απολύτως απαραίτητα. Θα χρησιμοποιήσουμε δύο διαφορετικά φίλτρα. Οι συντελεστές για τα δύο αυτά φίλτρα θα προέλθουν από το MATLAB.

Με την χρήση του πρώτου φίλτρου θα γίνει μία περισσότερο ποιοτική ανάλυση και για τον λόγο αυτό δεν θα χρησιμοποιήσουμε καμία άλλη πληροφορία για αυτό παρά μονάχα τους έξι συντελεστές του:

$a_0 = 0.0615$
$a_1 = 0.2832$
$a_2 = 0.4446$
$a_3 = 0.4446$
$a_4 = 0.2832$
$a_5 = 0.0615$

Λόγω της συμμετρίας των συντελεστών του φίλτρου, συμπεραίνουμε ότι πρόκειται για ένα βαθυπερατό φίλτρο, επομένως θα πρέπει να αναμένουμε ανάλογα αποτελέσματα. Οι τιμές των συντελεστών εισάγονται στο αρχείο fir.vhd, το οποίο βρίσκεται στο παράρτημα, στις αναθέσεις των σημάτων b στα components mac. Επειδή δεν χειριζόμαστε αριθμούς με υποδιαστολή, πολλαπλασιάζουμε τους συντελεστές με τον αριθμό 10.000, ώστε να γίνουν ακέραιοι, και έπειτα διαιρούμε με τον ίδιο αριθμό το αποτέλεσμα του φίλτρου ώστε να επανέλθει σε κανονικά επίπεδα. Αφού έχουμε εισάγει τους συντελεστές στο fir.vhd, επιλέγουμε από το μενού του XPS Tools → Clean → All, καθώς μπορεί να κάναμε μία αλλαγή σε ένα VHDL αρχείο αλλά το XPS δεν ανιχνεύει κάποια δομική αλλαγή και συνεπώς δεν θα συνθέσει πάλι το σύστημα. Επομένως εμείς καθαρίζουμε το σύστημα και επιλέγοντας Tools → Download το διορθωμένο σύστημα θα υλοποιηθεί και θα κατέβει στο FPGA.

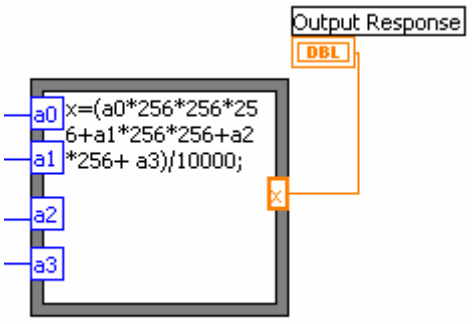
Το φίλτρο μας είναι έτοιμο να δεχθεί νέα δεδομένα, τα οποία θα οδηγηθούν μέσω της σειριακής θύρας και θα προέρχονται από ένα κύκλωμα του LabVIEW. Όπως είπαμε το πρώτο φίλτρο θα χρησιμοποιηθεί χωρίς να είναι γνωστά τα λεπτομερή του στοιχεία, για αυτό έχουμε δημιουργήσει ένα κύκλωμα που θα τροφοδοτεί το φίλτρο με τιμές ημιτόνου περιόδου συγκεκριμένων δειγμάτων. Επειδή δεν χρησιμοποιούμε πραγματικό χρόνο αλλά στέλνουμε εμείς τα δείγματα του ημιτόνου, κρατάμε σταθερό τον ρυθμό αποστολής δειγμάτων και μειώνουμε την περίοδο των ημιτόνων μετρούμενη σε δείγματα, δηλαδή αυξάνουμε την συχνότητά τους. Η διαδικασία αποστολής φαίνεται στο σχήμα 5.68.



Σχήμα 5.68 – Ο τρόπος παραγωγής των δειγμάτων

Στο σχήμα 5.68 μπορούμε να παρατηρήσουμε δύο πράγματα. Το πρώτο είναι ότι παίρνουμε δείγματα από ένα ημίτονο πλάτους δέκα μονάδων, οι οποίες δεν έχουν φυσική υπόσταση, και η περίοδος του ημιτόνου σε δείγματα καθορίζεται από έναν συντελεστή i ο οποίος αυξάνει κατά 0,00001. Επομένως η περίοδος του ημιτόνου καθορίζεται από τον αριθμό των αυξήσεων του συντελεστή i που πρέπει να γίνουν ώστε το όρισμα της συνάρτησης sin() να ξεπεράσει την τιμή 2π. Το δεύτερο είναι ότι επειδή χρησιμοποιούμε μη προσημασμένους αριθμούς, και δεδομένου ότι έχουμε να κάνουμε με ένα βαθυπερατό φίλτρο, έχουμε προσθέσει στο ημίτονο μία dc τιμή 10 μονάδων ώστε να μην χρειαστεί να χειριστούμε αρνητικούς αριθμούς. Τα δείγματα του ημιτόνου, όπως φαίνεται κάτω δεξιά στο σχήμα 5.68, μεταδίδονται σαν ακέραιοι, δηλαδή έχουμε στρογγυλοποίηση, και με την μορφή δύο byte όπως ακριβώς τα αναμένει το φίλτρο μας.

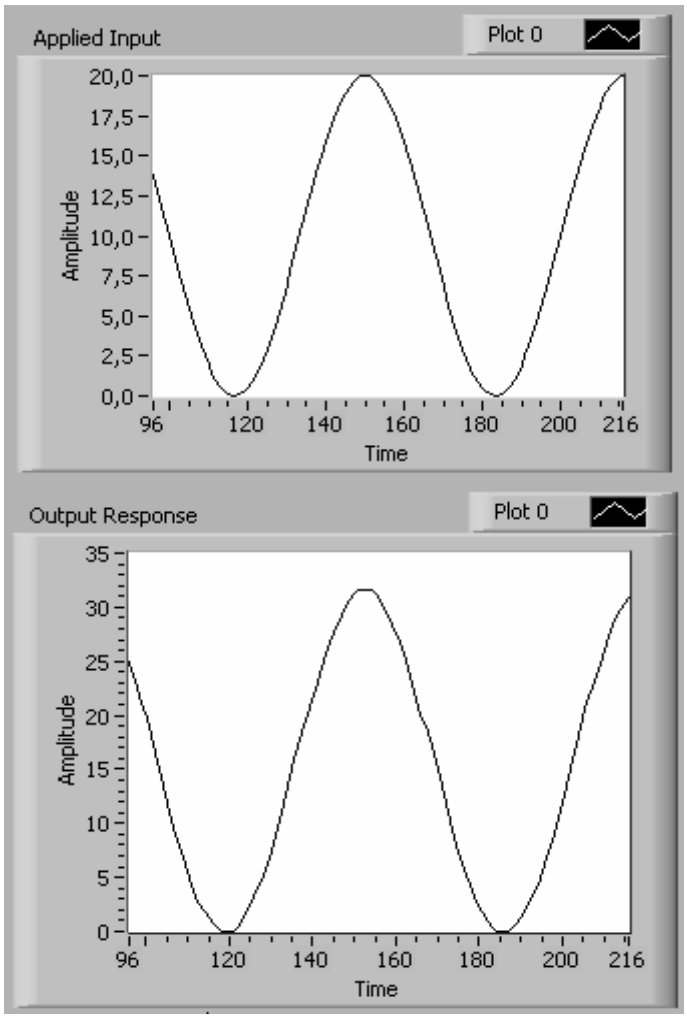
Το αποτέλεσμα του φίλτρου επιστρέφει με την μορφή τεσσάρων byte στο LabVIEW, όπου ανασυνθέτεται ο αριθμός με την διαδικασία που φαίνεται στο σχήμα 5.69.



Σχήμα 5.69 – Η ανασύνθεση του αριθμού στο LabVIEW

Είναι φανερό ότι ο αριθμός ανασυνθέτεται από τα τέσσερα bytes που στέλνει η κάρτα Spartan-3 Starter Kit ενώ φαίνεται, και αυτό που έχουμε ήδη αναφέρει, ότι διαιρούμε το αποτέλεσμα του φίλτρου με τον αριθμό 10.000 για να αντισταθμίσουμε τον πολλαπλασιασμό με τον ίδιο αριθμό των συντελεστών του φίλτρου.

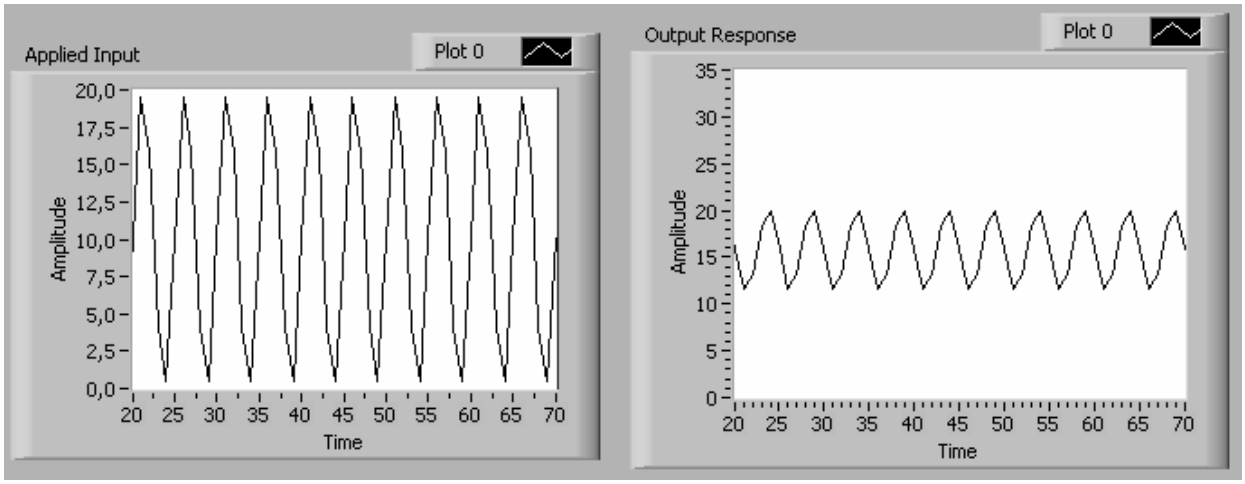
Στέλνοντας δείγματα ημιτόνου περιόδου 67 δειγμάτων έχουμε τα παρακάτω αποτελέσματα, με το πάνω διάγραμμα να αντιστοιχεί στην είσοδο και το κάτω στην έξοδο του φίλτρου:



Σχήμα 5.70 – Είσοδος και έξοδος ενός ημιτόνου 67 δειγμάτων

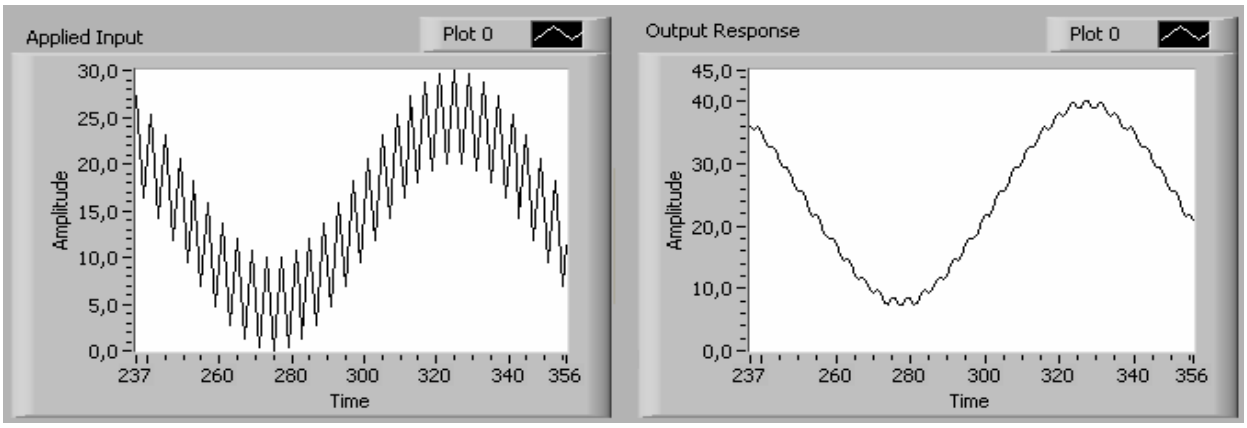
Αν συνυπολογίσουμε το γεγονός ότι για είσοδο 10 dc το φίλτρο έχει έξοδο 15,79 μονάδες, για αυτό και η αύξηση του πλάτους, και το ότι εισάγονται μόνο ακέραιοι αριθμοί στο φίλτρο, για αυτό και κάποιες ανωμαλίες στην κυματομορφή, τότε το ημίτονο περνάει από το φίλτρο.

Το γεγονός ότι έχουμε να κάνουμε με ένα βαθυπερατό φίλτρο φαίνεται από το ότι μειώνοντας την περίοδο, σε δείγματα, των ημιτόνων τότε μειώνεται το πλάτος τους στην έξοδο του φίλτρου. Ενδεικτικά στο σχήμα 5.71 φαίνεται η είσοδος και η έξοδος από το φίλτρο ενός ημιτόνου 6 δειγμάτων.



Σχήμα 5.71 – Όσο μειώνεται η περίοδος των ημιτόνων μειώνεται και το πλάτος της εξόδου τους

Ιδιαίτερα διαφωτιστικό είναι το επόμενο σχήμα, όπου φαίνεται η έξοδος του φίλτρου όταν εισάγουμε δύο ημίτονα, που το ένα προστίθεται στο άλλο, διαφορετικής περιόδου, με αποτέλεσμα την ενίσχυση αυτού με την μεγαλύτερη περίοδο και την καταπίεση αυτού με την μικρότερη περίοδο.

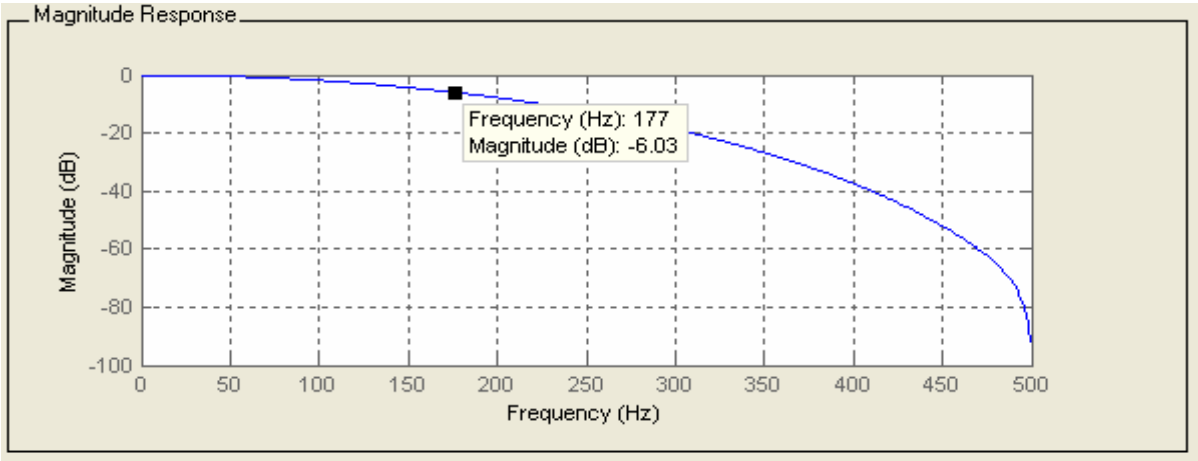


Σχήμα 5.72 – Η έξοδος του φίλτρου με είσοδο το άθροισμα δύο ημιτόνων διαφορετικής περιόδου

Μέχρις στιγμής είδαμε την λειτουργία ενός βαθυπερατού φίλτρου χωρίς να χρησιμοποιήσουμε ημίτονα συγκεκριμένης συχνότητας και χωρίς να ελέγξουμε την συχνότητα αποκοπής του. Για να το κάνουμε αυτό ζητάμε από το MATLAB, μέσω των κατάλληλων εργαλείων σύνθεσης φίλτρων που διαθέτει, να μας δώσει τους συντελεστές ενός βαθυπερατού φίλτρου, έξι συντελεστών, με συχνότητα 6 dB τα 100Hz και για συχνότητα δειγματοληψίας 1000 Hz. Οι συντελεστές που προέκυψαν για το φίλτρο αυτό, με στρογγυλοποίηση στα τέσσερα δεκαδικά ψηφία, είναι:

$a_0 = 0.0212$
$a_1 = 0.1452$
$a_2 = 0.3335$
$a_3 = 0.3335$
$a_4 = 0.1452$
$a_5 = 0.0212$

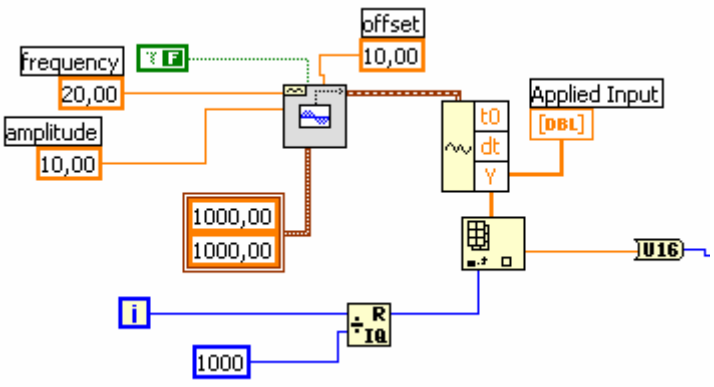
Επίσης η απόκριση πλάτους για το φίλτρο, που δίνει το MATLAB, φαίνεται στο σχήμα 5.73 όπου γίνεται κατανοητό ότι επειδή οι συντελεστές είναι λίγοι στον αριθμό η ζητούμενη συχνότητα 6 dB δεν είναι δυνατόν να επιτευχθεί και έχει μετατοπιστεί στα 177 Hz περίπου.



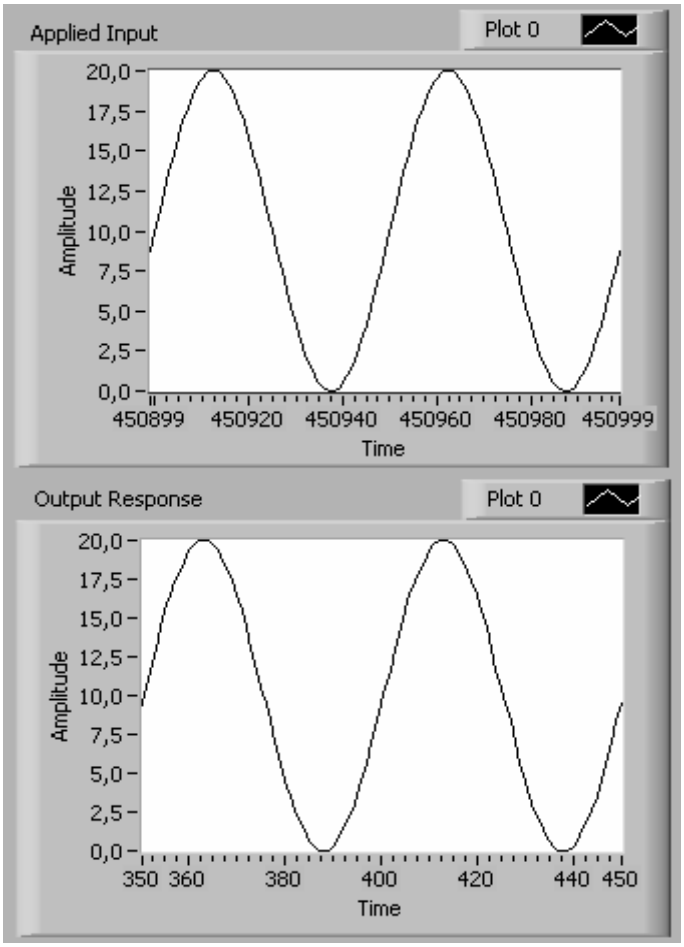
Σχήμα 5.73 – Η απόκριση πλάτους του φίλτρου

Για να αλλάξουμε τους συντελεστές του φίλτρου, όπως αναφέραμε και παραπάνω, μεταβαίνουμε στο αρχείο fir.vhd και αλλάζουμε τις τιμές των b στα components mac. Ασφαλώς έχουμε πολλαπλασιάσει τους συντελεστές του φίλτρου με 10.000 για να πάρουμε ακέραιους συντελεστές. Στην συνέχεια επιλέγουμε Tools → Clean → All και έπειτα Tools → Download και περιμένουμε να κατέβει το σύστημα στο FPGA.

Από πλευράς LabVIEW η επικοινωνία παραμένει η ίδια και το μόνο που αλλάζει είναι η δημιουργία των εισόδων για το φίλτρο. Έχουμε αντικαταστήσει τα δείγματα, μέσω μιας συνάρτησης ημιτόνου, με μία γεννήτρια που στο σήμα της πραγματοποιείται δειγματοληψία με συχνότητα 1000Hz. Με τον τρόπο αυτό έχουμε ημίτονα πραγματικών συχνοτήτων και χρησιμοποιούμε έναν πίνακα 1000 τιμών που προέρχονται από δειγματοληψία για να τροφοδοτήσουμε το φίλτρο.

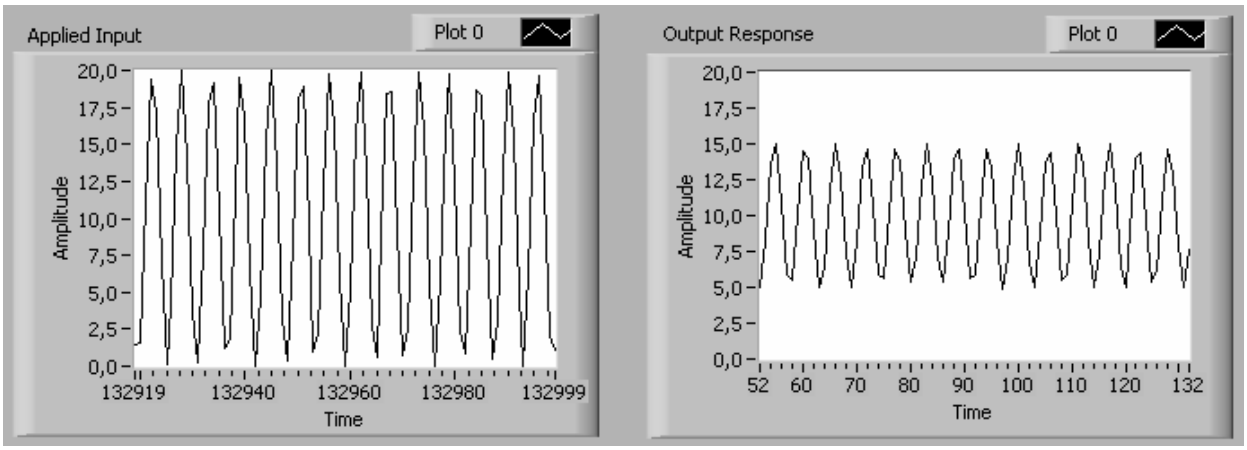


Σχήμα 5.74 – Τροφοδοσία του φίλτρου με τιμές από γεννήτρια



Σχήμα 5.75 – Αποτέλεσμα λειτουργίας του φίλτρου για ημίτονο συχνότητας 20 Hz

Στο σχήμα 5.75 φαίνεται η έξοδος από το φίλτρο όταν η είσοδος του είναι ένα ημίτονο συχνότητας 20 Hz και όπως γίνεται κατανοητό το ημίτονο περνάει σχεδόν άθικτο. Αντιθέτως δεν συμβαίνει το ίδιο με ένα ημίτονο με συχνότητα 177Hz, που όπως είδαμε στο MATLAB είναι η συχνότητα 6 dB, όπου το πλάτος του ημιτόνου μειώνεται στο μισό.



Σχήμα 5.76 – Αποτέλεσμα λειτουργίας του φίλτρου για ημίτονο συχνότητας 177 Hz

Παράρτημα : Κώδικας περιγραφής ψηφιακού φίλτρου

1. Φίλτρο (fir.vhd)

```
--
--  File: fir.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;

entity fir is
        port (
                xnew: in std_logic_vector (0 to 15);
                clk: in std_logic;
                clr: in std_logic;
                wr: in std_logic;
                ynew: out std_logic_vector (0 to 31)
        );
end fir;


architecture behavioural of fir is

-- It uses 6 mac components

component mac port(
                                clk: in std_logic;
                                a: in std_logic_vector (0 to 15);
                                b: in std_logic_vector (0 to 12);
                                nd: in std_logic;
                                clr: in std_logic;
                                qnew: out std_logic_vector (0 to 31);
                                qold: in std_logic_vector (0 to 31)
                        );
end component;

-- It uses one shift register for the insertion of the new data

component reg_port port(
                                        inp: in std_logic_vector (0 to 15);
                                        out0: out std_logic_vector(0 to 15);
                                        out1: out std_logic_vector (0 to 15);
                                        out2: out std_logic_vector (0 to 15);
                                        out3: out std_logic_vector (0 to 15);
                                        out4: out std_logic_vector (0 to 15);
                                        out5: out std_logic_vector (0 to 15);
                                        clk: in std_logic;
                                        clr: in std_logic;
                                        wr:  in std_logic
                                );
end component;

signal nd       : std_logic;
signal temp0    : std_logic_vector(0 to 15);
signal temp1    : std_logic_vector(0 to 15);
signal temp2    : std_logic_vector(0 to 15);
signal temp3    : std_logic_vector(0 to 15);
signal temp4    : std_logic_vector(0 to 15);
signal temp5    : std_logic_vector(0 to 15);
```

```
signal q0                   : std_logic_vector(0 to 31);
signal q1                   : std_logic_vector(0 to 31);
signal q2                   : std_logic_vector(0 to 31);
signal q3                   : std_logic_vector(0 to 31);
signal q4                   : std_logic_vector(0 to 31);

begin

        process (clk)
                begin
                        if clk'event and clk='1' then
                                if wr='1' then
                                        nd <= '1';
                                else nd <= '0';
                                end if;
                        end if;
                end process;

shift:  reg_port port map (inp => xnew, out0 => temp0, out1 => temp1, out2 => temp2, out3 => temp3, out4 =>
temp4, out5 => temp5, clk => clk, clr => clr, wr => wr);
mac0:   mac port map (clk => clk, a => temp0, b => "0001001100111", nd => nd, clr => clr, qnew => q0, qold =>
(others => '0'));
mac1:   mac port map (clk => clk, a => temp1, b => "0101100010000", nd => nd, clr => clr, qnew => q1, qold =>
q0);
mac2:   mac port map (clk => clk, a => temp2, b => "1000101011110", nd => nd, clr => clr, qnew => q2, qold =>
q1);
mac3:   mac port map (clk => clk, a => temp3, b => "1000101011110", nd => nd, clr => clr, qnew => q3, qold =>
q2);
mac4:   mac port map (clk => clk, a => temp4, b => "0101100010000", nd => nd, clr => clr, qnew => q4, qold =>
q3);
mac5:   mac port map (clk => clk, a => temp5, b => "0001001100111", nd => nd, clr => clr, qnew => ynew, qold
=> q4);

end behavioural;
```

2. Πολλαπλασιαστής – Αθροιστής (mac.vhd)

```
--
--  File: mac.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity mac is
        port (
                clk: in std_logic;
                a: in std_logic_vector (0 to 15);
                b: in std_logic_vector (0 to 12);
                nd: in std_logic;
                clr: in std_logic;
                qnew: out std_logic_vector (0 to 31);
                qold: in std_logic_vector (0 to 31)
        );
end mac;


architecture behavioural of mac is

signal mult_int    : std_logic_vector(0 to 28);
signal Q_int       : std_logic_vector(0 to 31);
signal rdy_int     : std_logic;
signal a_reg       : std_logic_vector(0 to 15);  -- The data of the filter
signal b_reg       : std_logic_vector(0 to 12);  -- The coefficients of the filter
signal nd_reg      : std_logic;                  -- The enable signal

begin
        input_regs : process(clk)
        begin
                if clk'event and clk='1' then
                        if clr='1' then
                                a_reg <= (others => '0');
                                b_reg <= (others => '0');
                                nd_reg <= '0';
                        else
                                a_reg <= a;
                                b_reg <= b;
                                nd_reg <= nd;
                        end if;
                end if;
        end process input_regs;

        multi_proc : process(clk)
        begin
                if clk'event and clk='1' then
                        if nd_reg='1' then
                                mult_int <= a*b;
                                rdy_int <= '1';
                        end if;
                end if;
        end process multi_proc;

        accum_proc : process(clk, mult_int, clr)
        begin
                if clr='1' then
```

```
                Q_int <= (others => '0');
            elsif clk'event and clk='1' then
                if rdy_int='1' then
                    Q_int <= mult_int + qold;
                else
                    Q_int <= qold;
                end if;
            end if;
        end process accum_proc;

        qnew <= Q_int;

end behavioural;
```

3. Τμήμα ολίσθησης καταχωρητών (shift_reg.vhd)

```
--
--  File: shift_reg.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;

entity reg_port is
        port (
                inp: in std_logic_vector (0 to 15);
                out0: out std_logic_vector (0 to 15);
                out1: out std_logic_vector (0 to 15);
                out2: out std_logic_vector (0 to 15);
                out3: out std_logic_vector (0 to 15);
                out4: out std_logic_vector (0 to 15);
                out5: out std_logic_vector (0 to 15);
                clk: in std_logic;
                clr: in std_logic;
                wr:  in std_logic
        );
end reg_port;

architecture struct of reg_port is

        -- It uses six registers that refresh their content with a clock and an enable signal

        component reg port(
                                        regin: in std_logic_vector (0 to 15);
                                        regout: out std_logic_vector (0 to 15);
                                        clk: in std_logic;
                                        clr: in std_logic;
                                        wr:  in std_logic
                                );
        end component;

signal regout0      : std_logic_vector (0 to 15);
signal regout1      : std_logic_vector (0 to 15);
signal regout2      : std_logic_vector (0 to 15);
signal regout3      : std_logic_vector (0 to 15);
signal regout4      : std_logic_vector (0 to 15);
signal regout5      : std_logic_vector (0 to 15);

begin

reg0:   reg port map (regin => inp, regout => regout0, clk => clk, clr => clr, wr => wr);
reg1:   reg port map (regin => regout0, regout => regout1, clk => clk, clr => clr, wr => wr);
reg2:   reg port map (regin => regout1, regout => regout2, clk => clk, clr => clr, wr => wr);
reg3:   reg port map (regin => regout2, regout => regout3, clk => clk, clr => clr, wr => wr);
reg4:   reg port map (regin => regout3, regout => regout4, clk => clk, clr => clr, wr => wr);
reg5:   reg port map (regin => regout4, regout => regout5, clk => clk, clr => clr, wr => wr);

        out0 <= regout0;
        out1 <= regout1;
        out2 <= regout2;
        out3 <= regout3;
        out4 <= regout4;
        out5 <= regout5;

end struct;
```

4. Καταχωρητής με παράλληλη φόρτωση (reg.vhd)

```
--
--  File: reg.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;

entity reg is
        port (
                regin: in std_logic_vector (0 to 15);
                regout: out std_logic_vector (0 to 15);
                clk: in std_logic;
                clr: in std_logic;
                wr:  in std_logic
        );
end reg;

architecture behavioural of reg is
begin
        process (clk,clr)
        begin
                if clk'event and clk='1' then
                        if clr='1' then
                                regout <= (others =>'0');
                        elsif wr='1' then
                                regout <= regin;
                        end if;
                end if;
  end process;
end behavioural;
```

5. Κύτταρο D flip-flop (dff.vhd)

```
--
--  File: dff.vhd
--

library IEEE;
use IEEE.std_logic_1164.all;

entity dff is
        port (
                d               :       in std_logic;
                clear   :       in std_logic;
                clk             :       in std_logic;
                q               :       out std_logic);
end dff;

architecture bhv_dff of dff is
begin
        process(clk,clear)
        begin
                if clear = '1' then
                        q <= '0';
                elsif clk'event and clk='1' then
                        q <= d;
                end if;
        end process;
end bhv_dff;
```