ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Συντονισμένη Σχεδίαση Υλικού – Λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ Β. ΝΙΚΑ

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

Αθήνα, Αύγουστος 2003

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

Συντονισμένη Σχεδίαση Υλικού – Λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ Β. ΝΙΚΑ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Αυγούστου 2003.

.....	.....	.....
Κ. Ζ. Πεκμεστζή	Παναγιώτης Τσανάκας	Ηλίας Κουκούτσης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Αύγουστος 2003

.....

ΚΩΝΣΤΑΝΤΙΝΟΣ Β. ΝΙΚΑΣ

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

Πρόλογος

Αντικείμενο της διπλωματικής εργασίας αποτελεί η παρουσίαση της μεθοδολογίας της "**Συντονισμένης Σχεδίασης Υλικού – Λογισμικού"** και των εργαλείων που χρησιμοποιούνται για την κατασκευή ενός απλού συστήματος. Τα εργαλεία αυτά είναι το Seamless CVE ™ και το Platform Express ™ της εταιρίας Mentor Graphics ®. Το σύστημα που σχεδιάστηκε αποτελείται από έναν πολλαπλασιαστή, ο οποίος δέχεται δεδομένα από έναν επεξεργαστή ARM, πραγματοποιεί τους πολλαπλασιασμούς και επιστρέφει τα γινόμενα στον επεξεργαστή.

Πιο συγκεκριμένα, το Κεφάλαιο 1 αναφέρει κάποια εισαγωγικά στοιχεία για την μέθοδο της "Συντονισμένης Σχεδίασης Υλικού – Λογισμικού". Το Κεφάλαιο 2 περιγράφει τη γενική αρχιτεκτονική και τη λειτουργία των εργαλείων που χρησιμοποιούνται κατά τη μεθοδολογία αυτή. Το Κεφάλαιο 3 παρουσιάζει τη λειτουργία των εργαλείων που χρησιμοποιήθηκαν για την εκπόνηση της παρούσας διπλωματικής. Το Κεφάλαιο 4 επιχειρεί μια συνοπτική παρουσίαση του διαδρόμου APB που χρησιμοποιήθηκε στο σύστημα μας. Το Κεφάλαιο 5 παρουσιάζει τη σχεδίαση του υλικού του συστήματός μας, ενώ το Κεφάλαιο 6 περιγράφει την ενσωμάτωση του συστήματος στο πρόγραμμα Platform Express™ . Το Κεφάλαιο 7 παρουσιάζει το λογισμικό του συστήματος και τα αποτελέσματα της προσομοίωσης που πραγματοποιήθηκε με το εργαλείο Seamless CVE ™. Τέλος, στο Κεφάλαιο 8 επιχειρείται μια ανακεφαλαίωση και η διατύπωση ιδεών – προτάσεων για μελλοντικές εργασίες που θα στηριχθούν στην εμπειρία που συσσωρεύτηκε κατά την εκπόνηση της παρούσας διπλωματικής εργασίας.

Για την εκπόνηση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω κατά κύριο λόγο τον καθηγητή κ. Κ. Ζ. Πεκμεστζή και τον κ. Γ. Οικονομάκο, για τις συμβουλές και τις ιδέες τους κατά τη διάρκεια της εργασίας αυτής. Επίσης, θα ήθελα να τονίσω τη συμβολή των μεταπτυχιακών φοιτητών Κ. Ασφή και Κ. Γκότση, και να τους ευχαριστήσω για τη βοήθεια τους στην αντιμετώπιση σημαντικών πρακτικών προβλημάτων που προέκυψαν, καθώς και για την ηθική και ψυχολογική συμπαράσταση που μου προσέφεραν.

Λέξεις – Κλειδιά : Συντονισμένη Σχεδίαση Υλικού – Λογισμικού, Seamless CVE, Platform Express, PxEdit, πολλαπλασιαστής, φίλτρο, διάδρομος AMBA APB.

Abstract

The purpose of the present diploma thesis is the presentation of "Hardware – Software Codesign" approach and the computer programs that are used during the design of an embedded system. In particular, in order to design a multiplier that will receive data from an ARM processor, execute the multiplications and return the results to the processor, we will use Seamless CVE ™ and Platform Express ™ of Mentor Graphics ®.

Chapter 1 introduces "Hardware – Software Codesign" and points out the advantages of this approach. Chapter 2 describes the architecture of the computer programs that are used during this approach. Chapter 3 presents the programs that were used during the design of our system. Chapter 4 attempts a short presentation of the bus APB, which was used to connect our multiplier with the processor. Chapter 5 describes the design of the hardware, while Chapter 6 describes the integration of our system to the environment of Platform Express ™. Chapter 7 presents the software that was used to test our system and the results of the simulation with the program Seamless CVE ™. Finally, in Chapter 8 we express some ideas of future projects, that will exploit the experience and the knowledge that was acquired during this project.

Key Words : Hardware – Software Codesign, Seamless CVE, Platform Express, PxEdit, multiplier, filter, bus AMBA APB.

Περιεχόμενα

Ευρετήριο Σχημάτων

Κεφάλαιο 7

Ευρετήριο Πινάκων

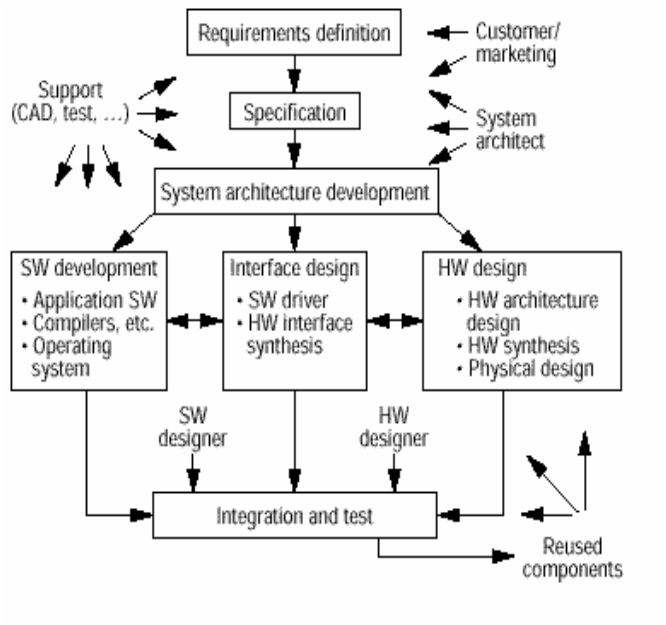
Κεφάλαιο 3

Κεφάλαιο 5

1. HARDWARE-SOFTWARE CODESIGN

1.1 Εισαγωγή

Όπως έχει γίνει πια φανερό, τα embedded συστήματα έχουν κυριαρχήσει στην αγορά, καθώς οι δυνατότητες τους αυξάνονται αξιοποιώντας κατάλληλα τις τεχνολογικές εξελίξεις, διευρύνοντας ταυτόχρονα τις εφαρμογές τους. Είναι λοιπόν λογικό η σχεδίαση τους να αποτελεί μια πρόκληση, μιας και η προσπάθεια για την παραγωγή φθηνών συστημάτων με περισσότερες δυνατότητες είναι συνεχής.
Η διαδικασία σχεδιασμού φαίνεται αναλυτικά στο παρακάτω σχήμα :



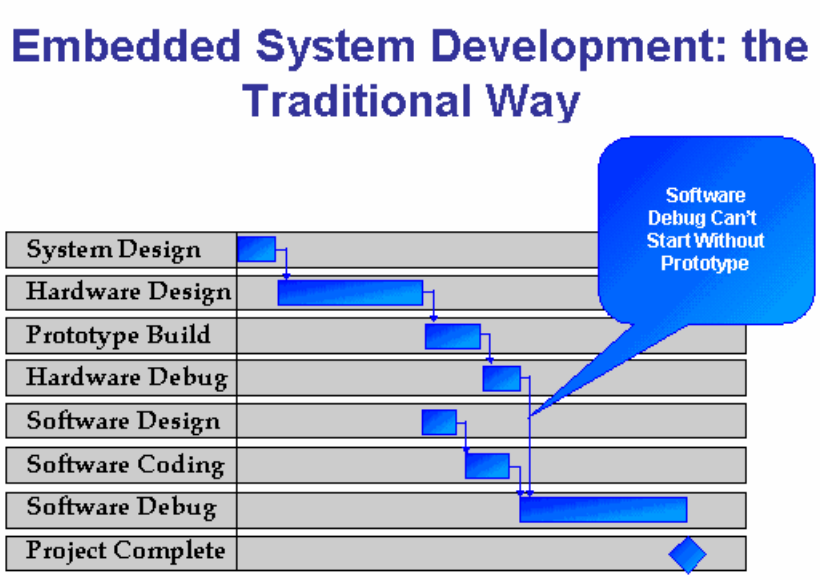
Σχήμα 1.1 Διαδικασία Σχεδιασμού Embedded Συστημάτων

Σύμφωνα λοιπόν 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 ποιά από αυτά θα υλοποιούνται από το λογισμικό και ποιά 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 compilers. Επίσης, κατασκευάζονται οι οδηγοί (drivers) που είναι απαραίτητοι για την σωστή λειτουργία του συστήματος. Μόλις ολοκληρωθεί η

κατασκευή του υλικού ακολουθεί η ενοποίηση υλικού-λογισμικού και η τελική δοκιμή του συστήματος.

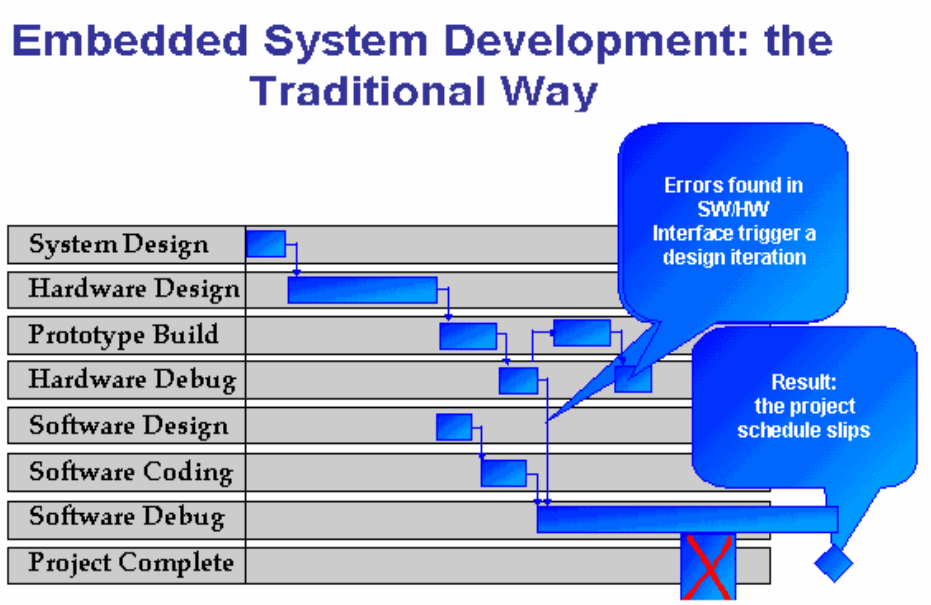
Εύκολα λοιπόν διαπιστώνουμε πως ο διαχωρισμός της σχεδίασης του Υλικού και του Λογισμικού γίνεται αρκετά νωρίς, ενώ στην συνέχεια εξελίσσονται ανεξάρτητα, αλληλεπιδρώντας ελάχιστα μέχρι την τελική ενοποίηση. Συχνά μάλιστα, το υλικό σχεδιάζεται χωρίς προηγουμένως να έχουν συγκεκριμενοποιηθεί πλήρως οι υπολογιστικές απαιτήσεις του λογισμικού. Από την άλλη μεριά, οι υπεύθυνοι του λογισμικού δεν επηρεάζουν τη σχεδίαση του υλικού. Η διαδικασία αυτή περιορίζει όπως είναι φανερό τη δυνατότητα διερεύνησης διαφόρων συνδυασμών υλικού-λογισμικού, προκειμένου να βρεθεί ο πιο αποδοτικός.

Ένα άλλο μειονέκτημα της παραπάνω διαδικασίας αποτελεί ο χρόνος που απαιτείται. Συγκεκριμένα, όπως γίνεται φανερό και από το παρακάτω διάγραμμα, ο έλεγχος και η διόρθωση τυχόν λαθών του λογισμικού μπορεί να πραγματοποιηθεί μόνο μετά την κατασκευή ενός πρωτοτύπου :



Σχήμα 1.2 Χρονοδιάγραμμα Υλοποίησης Συστήματος με τη παραδοσιακή μέθοδο

Είναι όμως πολύ πιθανό να ανακαλυφθούν προβλήματα, η επίλυση των οποίων να απαιτεί αλλαγές είτε στο υλικό είτε στο λογισμικό.



Σχήμα 1.3 Χρονοδιάγραμμα Υλοποίησης Συστήματος με τη παραδοσιακή μέθοδο

Στην περίπτωση αυτή λοιπόν το κόστος του συστήματος αυξάνεται καθώς πρέπει να επαναληφθεί η σχεδίαση κάποιων τμημάτων, παρατηρείται παραβίαση του χρονοδιαγράμματος που έχει τεθεί, ενώ τέλος δεν επιτυγχάνεται η μέγιστη δυνατή απόδοση του συστήματος, γεγονός που οφείλεται στο ότι συνήθως καλείται το λογισμικό να διορθώσει και να αντιμετωπίσει τα προβλήματα που προκαλεί η κατασκευή ανεπαρκούς ή ακατάλληλου υλικού.

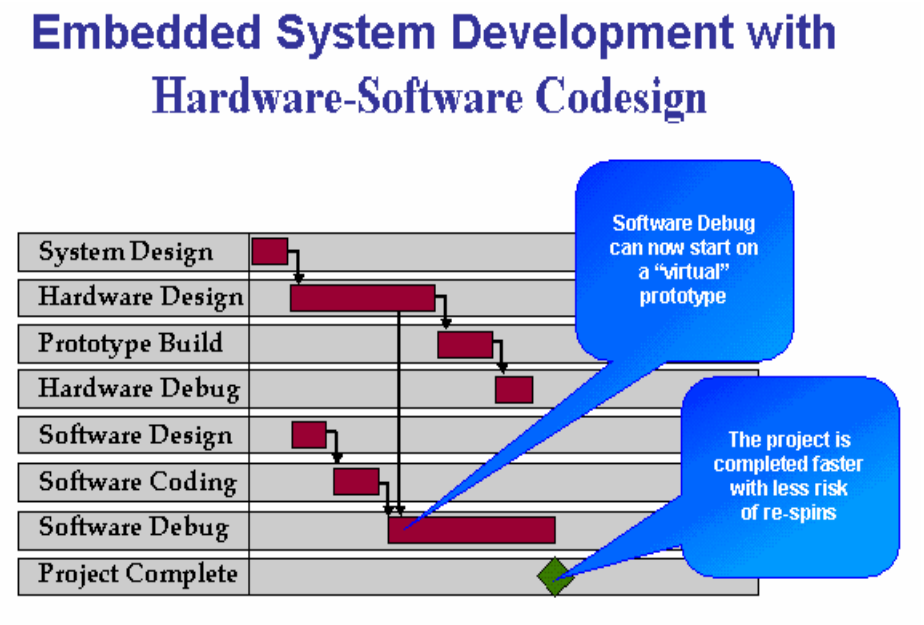
Τα παραπάνω προβλήματα σε συνδυασμό με την ολοένα κι αυξανόμενη πολυπλοκότητα των συστημάτων και τα διαρκώς συρρικνωμένα χρονικά διαστήματα που απαιτούνται από τη βιομηχανία μεταξύ της σχεδίασης του συστήματος και της εισαγωγής του στην αγορά, απαιτούν την εξέλιξη και βελτίωση της παραπάνω διαδικασίας. Συγκεκριμένα, απαιτείται μια πιο ενιαία και ολοκληρωμένη προσέγγιση της σχεδίασης embedded συστημάτων. Η προσέγγιση αυτή ονομάζεται "**Συντονισμένη Σχεδίαση Υλικού-Λογισμικού**" (**Hardware-Software Codesign**) ή πιο απλά "**Συντονισμένη Σχεδίαση**" (**Codesign**).

1.2 Ορισμός – Πλεονεκτήματα

Η **"Συντονισμένη Σχεδίαση Υλικού-Λογισμικού" (Hardware-Software Codesign)** αποτελεί, λοιπόν, μια πιο ενιαία και ολοκληρωμένη προσέγγιση της σχεδίασης, με κύριο σκοπό την ενδελεχή εξερεύνηση και αξιολόγηση των διαφόρων δυνατών συνδυασμών υλικού και λογισμικού. Η δυνατότητα αυτή οδηγεί στην σχεδίαση αποδοτικών υλοποιήσεων με μέγιστη απόδοση, αξιοπιστία και μικρό κόστος κατασκευής.

Πιο συγκεκριμένα, σύμφωνα με τη μέθοδο αυτή, κατασκευάζεται με τη βοήθεια διαφόρων ειδικών προγραμμάτων και γλωσσών περιγραφής υλικού ένα "εικονικό"

πρωτότυπο. Αυτό χρησιμοποιείται για τον έλεγχο του λογισμικού, ο οποίος ξεκινάει νωρίτερα από ότι προηγουμένως. Το γεγονός αυτό εικονίζεται στο επόμενο διάγραμμα :



Σχήμα 1.4 Χρονοδιάγραμμα Υλοποίησης Συστήματος με τη μέθοδο Hardware-Software Codesign

Διαπιστώνουμε λοιπόν ότι πλέον είναι αναγκαία η συνεργασία των σχεδιαστών υλικού και λογισμικού, γεγονός με πολλά πλεονεκτήματα. Καταρχάς ελαχιστοποιείται ο κίνδυνος ύπαρξης σφαλμάτων στην τελική υλοποίηση, κι άρα και η πιθανότητα επανασχεδίασης διαφόρων τμημάτων του συστήματος. Επομένως, ελαχιστοποιείται ο χρόνος που απαιτείται προκειμένου να ολοκληρωθεί η σχεδίαση, καθώς και το κόστος της.

Επίσης, η "εικονική" ενοποίηση υλικού-λογισμικού δίνει τη δυνατότητα στους σχεδιαστές να μελετήσουν πολύ νωρίς τη συμπεριφορά του συστήματος. Μπορούν λοιπόν να καταλήξουν σε συμπεράσματα σχετικά με την απόδοση του συστήματος, όπως για παράδειγμα ποιες λειτουργίες πρέπει να πραγματοποιούνται από το λογισμικό και ποιες από το υλικό. Μπορούν δηλαδή να κάνουν τη σωστή επιλογή του συνδυασμού υλικού και λογισμικού, η οποία θα οδηγήσει στην υλοποίηση ενός αποδοτικού συστήματος.

Τέλος, η χρησιμοποίηση των "εικονικών" προτύπων ενισχύει την επαναχρησιμοποίηση έτοιμων τμημάτων υλικού, τα οποία είτε έχουν χρησιμοποιηθεί σε προηγούμενα σχέδια είτε έχουν υλοποιηθεί από άλλες σχεδιαστικές ομάδες. Οι σχεδιαστές μπορούν να μελετήσουν τη συμπεριφορά αυτών των τμημάτων και να επιλέξουν τα κατάλληλα, ελαχιστοποιώντας έτσι το χρόνο σχεδίασης του υλικού, καθώς και τον κίνδυνο ύπαρξης σφαλμάτων στην τελική σχεδίαση του υλικού.

1.3 Προβλήματα – Μελλοντικές Προκλήσεις

Το πιο σημαντικό ίσως πρόβλημα της παραπάνω προσέγγισης αποτελεί το γεγονός πως η προσομοίωση του υλικού σε επίπεδο πυλών είναι πολύ αργή για να επιτρέπει την εκτέλεση του λογισμικού. Είναι απαραίτητο λοιπόν να βρεθούν οι κατάλληλες λύσεις προκειμένου να επιταχυνθεί η προσομοίωση του υλικού.
Ένα άλλο πολύ σημαντικό πρόβλημα αποτελεί ο συγχρονισμός της προσομοίωσης του υλικού και του λογισμικού, καθώς απαιτείται διαρκής έλεγχος της συνάφειας και της συνέπειας της μνήμης (memory coherence and consistency), ο οποίος καταναλώνει αρκετό χρόνο, που μάλιστα αυξάνεται με την πολυπλοκότητα του συστήματος.
Τα προβλήματα αυτά αποτελούν αντικείμενο έρευνας και ήδη κυκλοφορούν στην αγορά πολλά προγράμματα "επαλήθευσης" (**co-verification tools**), τα οποία επιλύουν τα προβλήματα αυτά σε κάποιο βαθμό. Και αυτά όμως είναι σχετικά πρόσφατα και εύλογα συμπεραίνουμε πως υπάρχουν πολλές δυνατότητες εξέλιξης. Παρ' όλα αυτά έχουν γίνει ήδη φανερά τα πλεονεκτήματα της μεθοδολογίας αυτής και στη βιομηχανία.

Βιβλιογραφία

1. Rolf Ernst : "*Codesign of Embedded Systems : Status and Trends"*, IEEE DESIGN & TEST OF COMPUTERS, April-June 1998.
2. Sanjaya Kumar, James H. Aylor, Barry W.Johnson and Wm. A. Wulf : " *A Framework for Hardware/Software Codesign"*, IEEE Computer, Volume 26, Number 12, 1993.

2. ΕΡΓΑΛΕΙΑ ΕΠΑΛΗΘΕΥΣΗΣ (CO-VERIFICATION TOOLS)

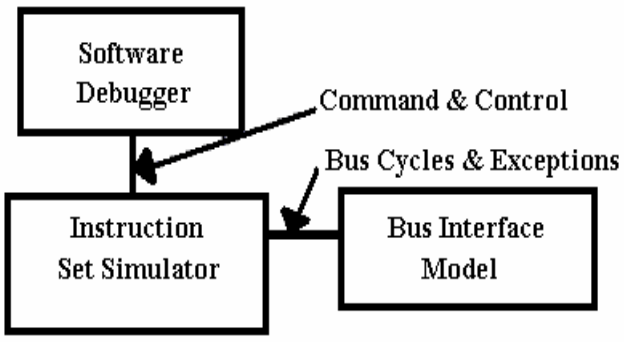
2.1 Εισαγωγή

Για να αντιμετωπιστούν λοιπόν αποτελεσματικά τα προβλήματα που αναφέρθηκαν προηγουμένως, τα διάφορα εργαλεία που έχουν δημιουργηθεί χρησιμοποιούν "αφαιρετικά" μοντέλα των επεξεργαστών σε συνδυασμό με το μοντέλο του υπό σχεδίαση συστήματος. Στη συνέχεια τα μοντέλα αυτά προσομοιώνονται, προκειμένου να ελεγχθεί τόσο το υλικό όσο και το λογισμικό.

Πιο συγκεκριμένα το αφαιρετικό μοντέλο του επεξεργαστή παρέχει τις εξής λειτουργίες :

- Επιτρέπει στο σχεδιαστή του λογισμικού να παρακολουθεί την κατάσταση του επεξεργαστή και να επεμβαίνει κατά τη διάρκεια της εκτέλεσης του προγράμματος. Γι' αυτό το λόγο τα εργαλεία που υπάρχουν στην αγορά περιλαμβάνουν κάποιο γραφικό αποσφαλματωτή (**graphical debugger**).
- Προσομοιώνεται μόνο η λειτουργική συμπεριφορά (**functional behavior**) του επεξεργαστή. Για το σκοπό αυτό χρησιμοποιούνται ειδικοί προσομοιωτές (**Instruction Set Simulators**), οι οποίοι είναι ταχύτεροι από την προσομοίωση του υλικού του επεξεργαστή, καθώς δεν χρειάζεται να υπολογίζουν όλες τις μεταβολές των σημάτων που πραγματοποιούνται στις πύλες και τους καταχωρητές του επεξεργαστή.
- Με τη βοήθεια μοντέλων διαδρόμων (**bus interface models)** επιτυγχάνεται η αλληλεπίδραση του υλικού του υπό σχεδίαση συστήματος με τον επεξεργαστή.

Διαγραμματικά το μοντέλο του επεξεργαστή που χρησιμοποιείται είναι το εξής :

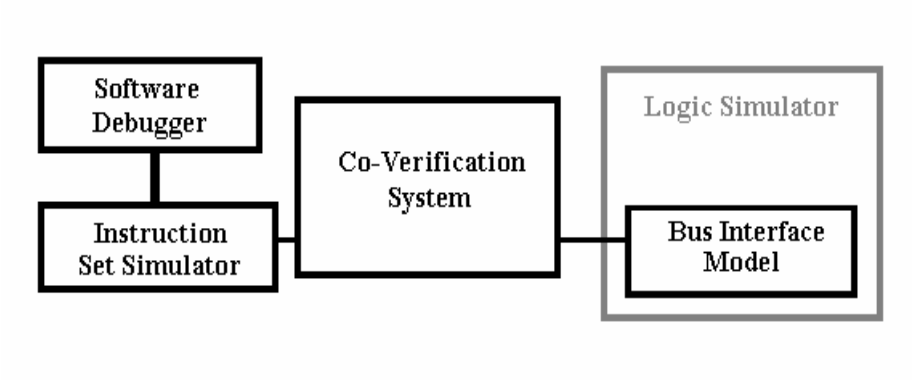


Σχήμα 2.1 Μοντέλο Επεξεργαστή

Τα εργαλεία λοιπόν ενώνουν τα μοντέλα προκειμένου να επιτευχθεί η προσομοίωση του υλικού και του λογισμικού. Έτσι :

- Πραγματοποιούν την λογική προσομοίωση του κυκλώματος με τη βοήθεια ειδικών προσομοιωτών (**Logic Simulators**).
- Εγκαθιστούν την επικοινωνία μεταξύ του υλικού και του λογισμικού του συστήματος.
- Φορτώνουν στο σύστημα εκτελέσιμα στιγμιότυπα του λογισμικού
- Εκτελούν το λογισμικό και δίνουν την δυνατότητα διόρθωσης λαθών τόσο στο υλικό όσο και στο λογισμικό (**Hardware and Software Debuggers**).

Η αρχιτεκτονική των προγραμμάτων συνοψίζεται στο παρακάτω σχήμα :



Σχήμα 2.2 Αρχιτεκτονική Προγράμματος Co-Verification

Ο διορθωτής λογισμικού (**Software Debugger)** παρέχει στο σχεδιαστή τη δυνατότητα παρακολούθησης των εσωτερικών καταχωρητών του επεξεργαστή, καθώς και της κατάστασης της μνήμης. Επίσης, του επιτρέπει να παρακολουθεί τις μεταβλητές που χρησιμοποιούνται και να ανιχνεύει τυχόν λογικά λάθη του λογισμικού.
Ο λογικός προσομοιωτής (**Logic Simulator)** προσομοιώνει το υλικό του υπό σχεδίαση συστήματος και δίνει στο σχεδιαστή τη δυνατότητα παρακολούθησης των αλλαγών των σημάτων. Έτσι γίνεται εύκολη η ανίχνευση τυχόν λαθών που υπάρχουν στη σχεδίαση. Η εκτέλεση όμως λογισμικού στην προσομοίωση αυτή είναι αρκετά αργή, αφού η τυπική ταχύτητα εκτέλεσης είναι περίπου 5 εντολές το δευτερόλεπτο. Έτσι τα εργαλεία καλούνται να βελτιστοποιήσουν την απόδοση, προκειμένου να είναι δυνατή η εκτέλεση μεγάλων τμημάτων λογισμικού κατά την προσομοίωση του υλικού.

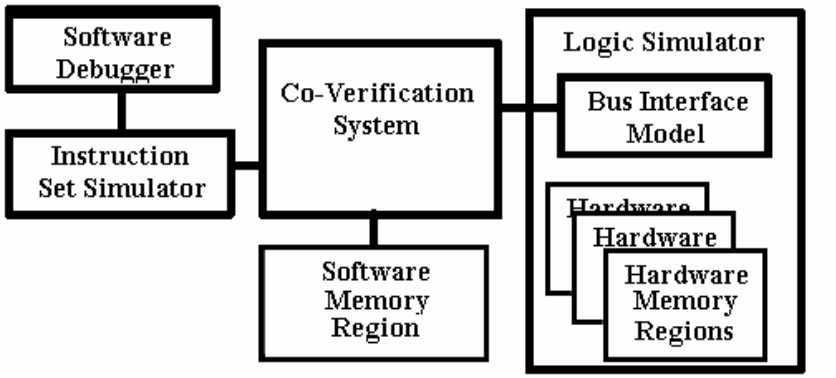
2.2 Βελτιστοποίηση Απόδοσης

Η ζητούμενη βελτιστοποίηση της απόδοσης στηρίζεται στην απόκρυψη κύκλων λειτουργίας από τον επεξεργαστή (**cycle hiding**). Κατά την απόκρυψη αυτή, κάποιες

συναλλαγές και λειτουργίες της μνήμης δεν προσομοιώνονται από τον Logic Simulator (προσομοιωτής υλικού) που όπως είδαμε είναι αργός, αλλά πραγματοποιούνται από κάποιο άλλο κομμάτι του λογισμικού του εργαλείου.
Συγκεκριμένα, ο ISS (Instruction Set Simulator) αναλαμβάνει κάθε φορά να αποκωδικοποιήσει την εντολή και να προσδιορίσει τα δεδομένα που πρέπει να μετακινηθούν ή να επεξεργαστούν, καθώς και τους κύκλους λειτουργίας του διαδρόμου που απαιτούνται.
Ταυτόχρονα παρέχεται στους σχεδιαστές η δυνατότητα κατάτμησης της μνήμης του συστήματος και χαρακτηρισμού των διαφόρων τμημάτων της ως εξής :

- Λογισμικό (Software)
- Υλικό (Hardware)

Στη συνέχεια και με βάση τα αποτελέσματα του ISS, αν η εντολή αναφέρεται σε τμήμα που έχει χαρακτηριστεί ως τμήμα υλικού ενεργοποιείται ο Logic Simulator και προσομοιώνει τους κύκλους λειτουργίας που απαιτούνται. Αν αντίθετα η εντολή αναφέρεται σε τμήμα χαρακτηρισμένο ως τμήμα λογισμικού, οι κύκλοι που απαιτούνται για την εκτέλεση της εντολής αποκρύπτονται από το Logic Simulator, ο οποίος δεν ενεργοποιείται, με αποτέλεσμα να επιταχύνεται η συνολική προσομοίωση. Έχουμε δηλαδή την εξής αρχιτεκτονική :



Σχήμα 2.3 Αρχιτεκτονική Co-Verification Προγράμματος

Όπως όμως είναι γνωστό, κατά την εκτέλεση του λογισμικού η πλειοψηφία των εντολών αποτελείται από εντολές αναφοράς στη μνήμη. Συγκεκριμένα έχουμε τις εξής εντολές :

- Ανάκλησης εντολών (Instruction fetch)
- Ανάκλησης δεδομένων (Data references)
- Εισόδου / Εξόδου Δεδομένων

Κάθε μια από τις εντολές αυτές χρησιμοποιεί για αρκετούς κύκλους το διάδρομο. Επίσης μια γραμμή κώδικα γραμμένη σε C απαιτεί πολλαπλάσιους κύκλους. Εύλογα λοιπόν συμπεραίνουμε πως με κατάλληλη τμηματοποίηση και χαρακτηρισμό της μνήμης η απόδοση του co-verification εργαλείου βελτιώνεται σημαντικά. Το συμπέρασμα αυτό γίνεται πιο κατανοητό με τη βοήθεια του παρακάτω παραδείγματος.
Έστω ότι η μνήμη έχει το εξής περιεχόμενο :

Code Segment

```
00FE   0214 D6A3    LD  SP,  6030
00FF   0000 6030
0100   0214 41A7    LS   R5,  @SP
0101   0214 42B8    LD  R6,  @SP+1
0102   351C 5C7D    JMPS  5C7D
……..
5C7D   5260 E355    OUT 55,  R3
5C7E   537C 2A05    IN @R5,  2A
```

Data Segment

```
6030   0080 0000
6031   1234 5678
6032   55AA AA55
```

Ξεκινώντας την εκτέλεση του προγράμματος από τη διεύθυνση 0100 παρατηρούμε στο διάδρομο τις εξής λειτουργίες :

Fetch	**@ 0100**	**Instruction Fetch**
Read	**@ 6030**	**Data Reference**
Fetch	**@ 0101**	**Instruction Fetch**
Read	**@ 6031**	**Data Reference**
Fetch	**@ 0102**	**Instruction Fetch**
Fetch	**@ 57CD**	**Instruction Fetch**
I/O WRITE	**@ Port 55**	**I/O Cycle**
Fetch	**@ 5C7E**	**Instruction Fetch**
I/O READ	**@ Port 2A**	**I/O Cycle**
Write	**@ 00800000**	**Data Reference**

Στην περίπτωση αυτή λοιπόν ο Logic Simulator θα προσομοιώσει 10 κύκλους λειτουργίας του διαδρόμου. Αν στη συνέχεια χαρακτηρίσουμε το Code Segment ως τμήμα "λογισμικού", στο διάδρομο θα έχουμε την εξής εικόνα :

Fetch	@ 0100	Instruction Fetch
Read	**@ 6030**	**Data Reference**
Fetch	@ 0101	Instruction Fetch
Read	**@ 6031**	**Data Reference**
Fetch	@ 0102	Instruction Fetch
Fetch	@ 57CD	Instruction Fetch
I/O WRITE	**@ Port 55**	**I/O Cycle**
Fetch	@ 5C7E	Instruction Fetch
I/O READ	**@ Port 2A**	**I/O Cycle**
Write	**@ 00800000**	**Data Reference**

Δηλαδή ο Logic Simulator θα προσομοιώσει τώρα μόνο 5 κύκλους λειτουργίας του διαδρόμου. Αν χαρακτηρίσουμε ως τμήμα "λογισμικού" και το Data Segment θα έχουμε την εξής δραστηριότητα :

Fetch	@ 0100	Instruction Fetch
Read	@ 6030	Data Reference
Fetch	@ 0101	Instruction Fetch
Read	@ 6031	Data Reference
Fetch	@ 0102	Instruction Fetch
Fetch	@ 57CD	Instruction Fetch
I/O WRITE	**@ Port 55**	**I/O Cycle**
Fetch	@ 5C7E	Instruction Fetch
I/O READ	**@ Port 2A**	**I/O Cycle**
Write	@ 00800000	Data Reference

Επομένως στην περίπτωση αυτή ο Logic Simulator απαιτείται να προσομοιώσει μόνο 2 κύκλους λειτουργίας.

Όταν λοιπόν οι σχεδιαστές έχουν ελέγξει την ορθή λειτουργία του συστήματος τους, μπορούν στη συνέχεια με κατάλληλο διαχωρισμό της μνήμης να μειώνουν τον αριθμό των λειτουργιών που προσομοιώνει ο Logic Simulator. Μπορούν δηλαδή να προσομοιώνουν μόνο τα τμήματα που τους ενδιαφέρουν, αφού για παράδειγμα δεν προσφέρει κάτι η συνεχής προσομοίωση της ανάκλησης των εντολών. Έτσι, αυξάνεται η ταχύτητα της συνολικής προσομοίωσης και δίνεται στους σχεδιαστές η δυνατότητα να πραγματοποιήσουν περισσότερες προσομοιώσεις σε μικρότερο χρονικό διάστημα, με αποτέλεσμα να ελαττώνεται ο απαιτούμενος χρόνος σχεδίασης του συστήματος.

Βιβλιογραφία

1. Rolf Ernst : "*Codesign of Embedded Systems : Status and Trends"*, IEEE DESIGN & TEST OF COMPUTERS, April-June 1998.
2. Sanjaya Kumar, James H. Aylor, Barry W.Johnson and Wm. A. Wulf : " *A Framework for Hardware/Software Codesign*", IEEE Computer, Volume 26, Number 12, 1993.
3. Mentor Graphics® : "*Seamless CVE User's and Reference Manual*"

3. Παρουσίαση Εργαλείων

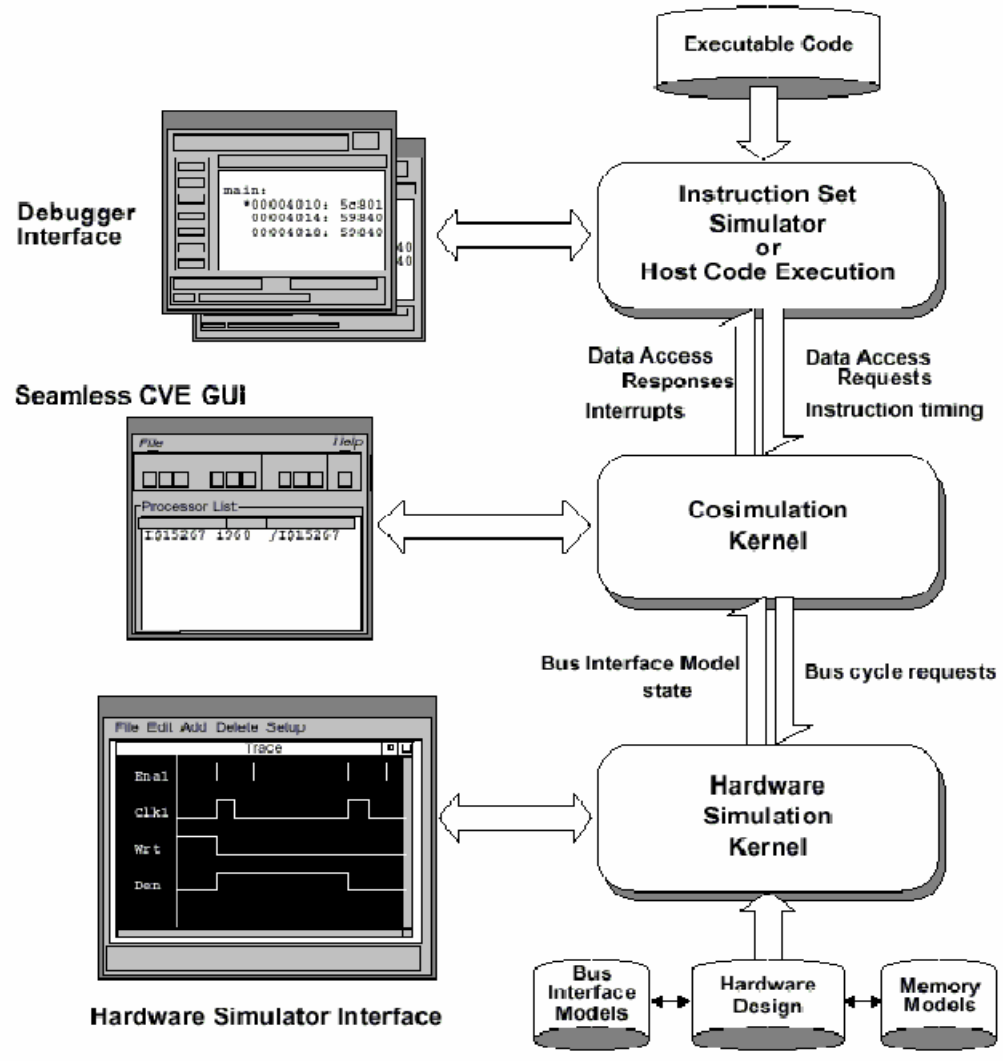
Για την πραγματοποίηση της διπλωματικής εργασίας χρησιμοποιήθηκαν διάφορα εργαλεία περιγραφής υλικού και επαλήθευσης, τα οποία παρουσιάζονται στη συνέχεια.

3.1 Seamless CVE ™ Co-verification Tool

Το βασικό εργαλείο που χρησιμοποιήθηκε προκειμένου να ακολουθηθεί η μέθοδος της Συντονισμένης Σχεδίασης είναι το "**Seamless CVE ™ Co-verification Tool"** της εταιρίας **Mentor Graphics ®**, το οποίο μπορεί να αναλυθεί στα εξής τμήματα :

- Τον πυρήνα της ταυτόχρονης προσομοίωσης (**Co-Simulation Kernel**)
- Τον ISS (**Instruction Set Simulator)**
- Τη διαπροσωπεία και τον πυρήνα του προσομοιωτή υλικού (**Hardware Simulator Interface and Kernel**)

Διαγραμματικά έχουμε :



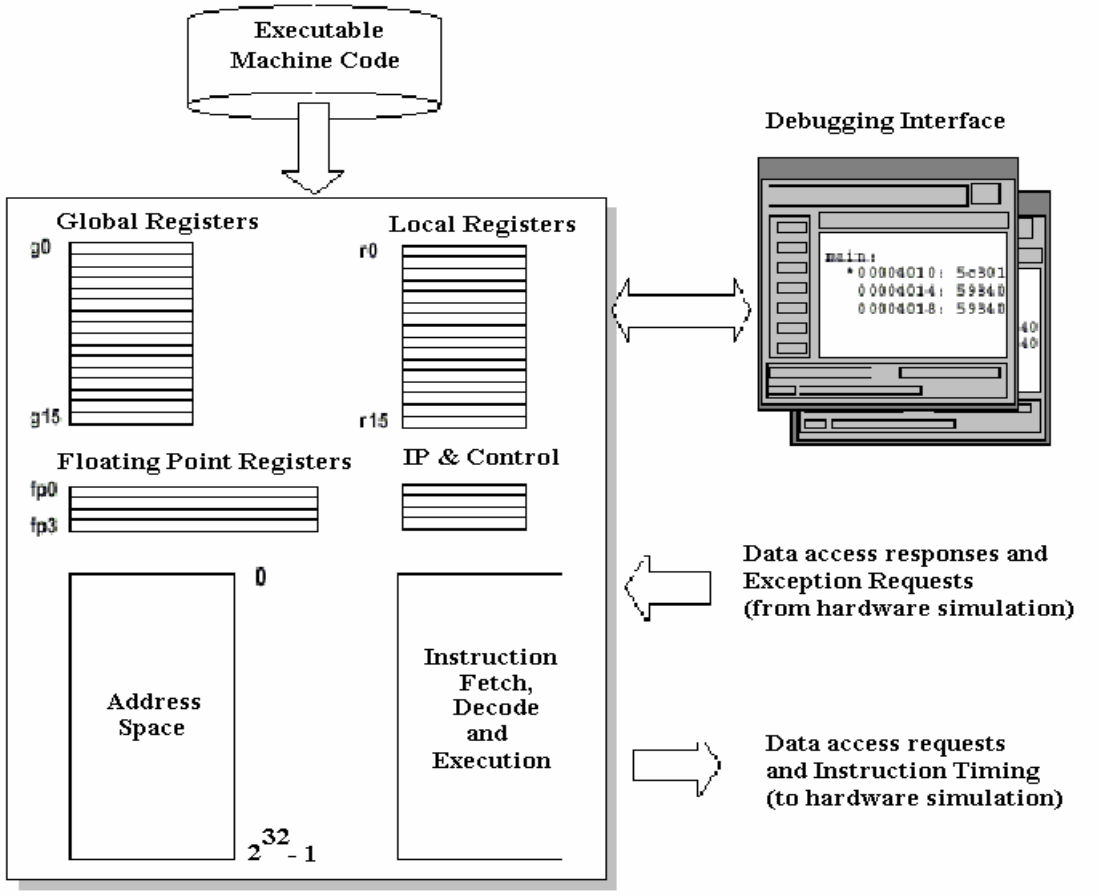
Σχήμα 3.1 Αρχιτεκτονική του Seamless CVE ™

3.1.1 Hardware Simulator Interface and Kernel

Ο προσομοιωτής αυτός αναλαμβάνει την προσομοίωση του υλικού του υπό σχεδίαση συστήματος. Ο πυρήνας εκτελεί τους επιθυμητούς συνδυασμούς υλικού, στους οποίους όμως έχουν συμπεριληφθεί ένα ή περισσότερα μοντέλα διαδρόμων (**bus interface models)**, ενώ οι μνήμες που περιλαμβάνονται στο σχέδιο έχουν αντικατασταθεί από τα ειδικά μοντέλα μνημών που χρησιμοποιεί το Seamless CVE ™, ώστε να είναι δυνατή η βελτιστοποίηση της απόδοσης. Το υλικό που χρησιμοποιείται έχει περιγραφεί σε κάποια γλώσσα περιγραφής υλικού (**VHDL** ή **Verilog**). Τέλος, η διαπροσωπεία προσφέρει στο σχεδιαστή τη δυνατότητα παρακολούθησης των διαφόρων σημάτων.

3.1.2 Instruction Set Simulator

Ο Instruction Set Simulator αποτελεί μια λογισμική εφαρμογή, η οποία μοντελοποιεί τη λειτουργική συμπεριφορά του επεξεργαστή και τρέχει πολύ πιο γρήγορα από την αντίστοιχη προσομοίωση του υλικού του επεξεργαστή, καθώς δεν υπολογίζει τις μεταβολές των σημάτων στις πύλες και τους καταχωρητές στο εσωτερικό του επεξεργαστή. Χρησιμοποιείται για την εκτέλεση του κώδικα μηχανής που παράγεται από το λογισμικό του υπό σχεδίαση συστήματος
Η αρχιτεκτονική του παρουσιάζεται στο επόμενο σχήμα :



Σχήμα 3.2 Αρχιτεκτονική του ISS

Ο debugger που παρέχεται ελέγχει τη λειτουργία του ISS και προσφέρει στο σχεδιαστή λειτουργίες όπως βηματική εκτέλεση εντολών, παρουσίαση των τιμών των καταχωρητών και της μνήμης, καθώς και άλλες τυπικές λειτουργίες ενός debugger.

3.1.3 Co-Simulation Kernel

Ο πυρήνας αυτός ελέγχει την επικοινωνία μεταξύ της προσομοίωσης του υλικού και αυτής του λογισμικού. Επίσης, συγχρονίζει την πρόσβαση των προσομοιωτών στη μνήμη του συστήματος έτσι, ώστε να διασφαλίζεται η συνάφεια και η συνέπεια της μνήμης (memory coherency and consistency), ενώ η διαπροσωπεία του παρέχει στο σχεδιαστή τη δυνατότητα ρύθμισης διαφόρων παραμέτρων της προσομοίωσης.

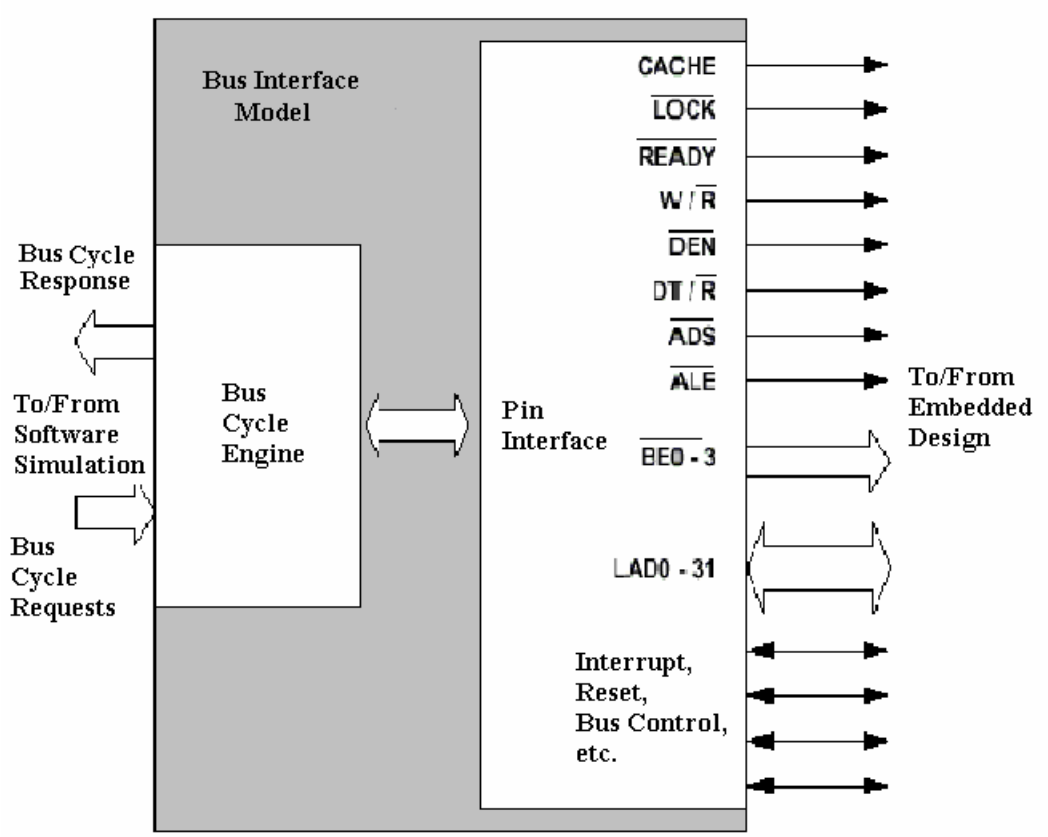
3.1.4 Bus Interface Models

Όσον αφορά την προσομοίωση του υλικού ο επεξεργαστής ουσιαστικά προσδιορίζεται από τον τρόπο με τον οποίο αλληλεπιδρούν τα "ποδαράκια" (**pins**) εισόδου–εξόδου με τα υπόλοιπα τμήματα του υλικού του υπό σχεδίαση τμήματος. Στο Seamless CVE ™ η αλληλεπίδραση αυτή μοντελοποιείται από τα "μοντέλα διαδρόμου" (**Bus Interface Models**), τα οποία δεν μοντελοποιούν την εσωτερική λογική του επεξεργαστή, με αποτέλεσμα να είναι πολύ ταχύτερα από την προσομοίωση ενός πλήρες μοντέλου του επεξεργαστή.

Τα μοντέλα αυτά είναι έτσι σχεδιασμένα ώστε να παρέχουν όλες τις λειτουργίες του αντίστοιχου επεξεργαστή. Έτσι μπορούν να :

- Διαβάσουν και να γράψουν τη μνήμη
- Διαβάσουν και να γράψουν τις θύρες Εισόδου – Εξόδου
- Πραγματοποιήσουν επανεκκίνηση (**reset**) και στάση (**halt**)
- Χειριστούν εξαιρέσεις (**exceptions)**, διακοπές (**interrupts)** και σφάλματα (**faults)**
- Ζητήσουν και να πάρουν τον έλεγχο του διαδρόμου (**Bus request and grant**)

Η αρχιτεκτονική τους περιγράφεται στο επόμενο σχήμα :



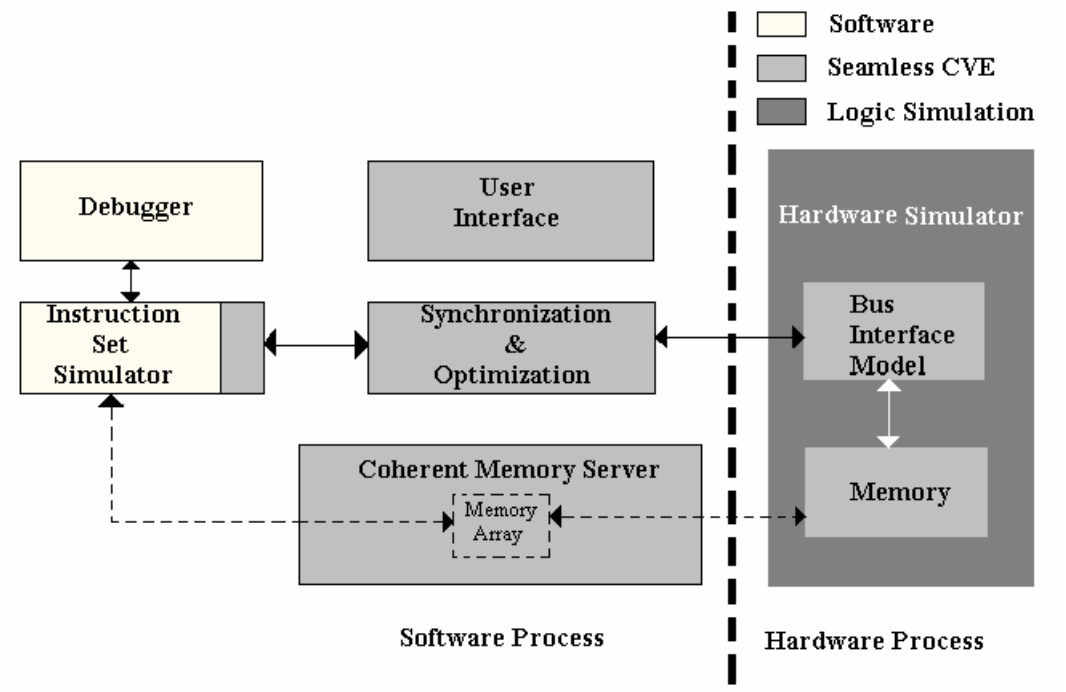
Σχήμα 3.3 Μοντέλο Διαδρόμου

Κατά τη διάρκεια της προσομοίωσης παράγονται από τον ISS αιτήσεις πρόσβασης σε δεδομένα σύμφωνα με το λογισμικό του συστήματος, οι οποίες αποτελούν αναγνώσεις ή εγγραφές είτε στη μνήμη είτε στις θύρες εισόδου—εξόδου. Όταν το μοντέλο διαδρόμου λάβει μια τέτοια αίτηση, εκτελεί τους απαιτούμενους κύκλους, επιβάλλοντας τα απαιτούμενα σήματα στα ανάλογα "ποδαράκια" εισόδου—εξόδου, όπως αυτά περιγράφονται από τον κατασκευαστή του επεξεργαστή.
Ανάλογη είναι η συμπεριφορά του μοντέλου και σε άλλα γεγονότα, όπως για παράδειγμα σε κάποια διακοπή. Το μοντέλο προσομοιώνει τη συμπεριφορά που έχει προβλέψει ο κατασκευαστής του επεξεργαστή και στέλνει τα απαιτούμενα δεδομένα στην προσομοίωση του λογισμικού, προκειμένου να εξυπηρετηθεί η αντίστοιχη ρουτίνα διακοπής.

3.1.5 Μοντέλα Μνημών (Seamless CVE™ Memory Models)

Όπως αναφέρθηκε και προηγουμένως το Seamless CVE ™ χρησιμοποιεί ειδικά μοντέλα προσομοίωσης για τις μνήμες που χρησιμοποιούνται σε ένα σύστημα. Η κύρια διαφορά τους από τα συμβατικά μοντέλα έγκειται στο ότι τα περιεχόμενα της μνήμης αποθηκεύονται σε ένα πίνακα, τον οποίο διαχειρίζεται μια ειδική διαδικασία (process) που ονομάζεται "**Coherent Memory Server**". Έτσι, τόσο η προσομοίωση του λογισμικού, που εκτελείται από τη διαδικασία που ονομάζεται "**Software Process**", όσο

και η προσομοίωση του υλικού, που εκτελείται από τη διαδικασία που ονομάζεται "**Hardware Process**", έχουν ανεξάρτητη πρόσβαση στη μνήμη. Αντίθετα, τα συμβατικά μοντέλα μνήμης διαχειρίζονται από την προσομοίωση του υλικού και δεν είναι δυνατή η απευθείας προσπέλαση τους από το λογισμικό. Το γεγονός αυτό έχει σαν αποτέλεσμα να καθυστερεί η προσομοίωση του λογισμικού από την πολύ πιο αργή προσομοίωση του υλικού.



Σχήμα 3.4 Αρχιτεκτονική Μοντέλων Μνήμης του Seamless CVE ™

Η ανεξάρτητη πρόσβαση του υλικού και του λογισμικού σε συνδυασμό με τον κατάλληλο χαρακτηρισμό των τμημάτων της μνήμης επιτρέπει την αφαίρεση αχρείαστων κύκλων λειτουργίας του υλικού, επιτυγχάνοντας με αυτό τον τρόπο την βελτιστοποίηση της απόδοσης, όπως αναφέρθηκε σε προηγούμενο κεφάλαιο. Στο περιβάλλον του Seamless CVE™ είναι δυνατό να χαρακτηριστούν τα τμήματα της μνήμης ως εξής :

- **Λογισμικό (Software)** : Χρησιμοποιείται για το χαρακτηρισμό εκείνων των τμημάτων της μνήμης που προσπελαύνει το λογισμικό και δεν περιλαμβάνονται στο υπό προσομοίωση υλικό. Τα τμήματα αυτά χρησιμοποιούνται αποκλειστικά από τον ISS και δεν προκαλούν κύκλους λειτουργίας στην προσομοίωση του διάδρομου. Αν για κάποιο λόγο ο προσομοιωτής υλικού προσπελάσει κάποια διεύθυνση που ανήκει σε ένα τέτοιο τμήμα, τότε αποφασίζει μόνος του για το περιεχόμενο της μνήμης. Δηλαδή το περιεχόμενο δεν θα είναι ίδιο με αυτό που βλέπει το λογισμικό.

- **Υλικό (Hardware)** : Χρησιμοποιείται για το χαρακτηρισμό εκείνων των τμημάτων της μνήμης που αντιστοιχούν σε "συσκευές με απεικόνιση μνήμης" (memory mapped devices) και σε συμβατικά μοντέλα μνήμης, τα οποία δεν μπορούν να βελτιστοποιηθούν. Η προσπέλαση αυτών των τμημάτων αναγκάζει κάθε φορά τον προσομοιωτή υλικού να προσομοιώνει τους αντίστοιχους κύκλους λειτουργίας. Δηλαδή δεν μπορούν να χρησιμοποιηθούν για την βελτιστοποίηση της ταχύτητας της προσομοίωσης.
- **Βελτιστοποιούμενα (Optimizable)** : Χρησιμοποιείται για τον χαρακτηρισμό εκείνων των τμημάτων της μνήμης που μπορούν να χρησιμοποιηθούν για την βελτιστοποίηση της ταχύτητας της προσομοίωσης.
- **Απαγορευμένα (Illegal)** : Χρησιμοποιείται για τον χαρακτηρισμό εκείνων των τμημάτων της μνήμης όπου δεν επιτρέπεται η πρόσβαση στο λογισμικό. Αν ο προσομοιωτής λογισμικού προσπαθήσει να προσπελάσει κάποια διεύθυνση που ανήκει σε ένα τέτοιο τμήμα παράγονται τα κατάλληλα μηνύματα λάθους. Αντίθετα, ο προσομοιωτής του υλικού μπορεί να την προσπελάσει και να καθορίσει το περιεχόμενό της.

Τέλος, το Seamless CVE™ παρέχει τα εξής μοντέλα μνήμης :

- Δυναμική (dynamic) RAM
- Στατική (static) RAM
- Δίθυρη (dual-port) RAM
- Ουρά FIFO
- Καταχωρητές

Τα μοντέλα αυτά είναι παραμετροποιήσιμα, επιτρέποντας στο σχεδιαστή να ορίσει το "μέγεθος" (width) της διεύθυνσης και του διαδρόμου δεδομένων.

3.1.6 Βελτιστοποιήσεις (Optimizations)

Χρησιμοποιώντας την τεχνική της απόκρυψης κύκλων λειτουργίας από την προσομοίωση του υλικού που εξηγήθηκε σε προηγούμενο κεφάλαιο, το Seamless CVE™ παρέχει στο σχεδιαστή τις εξής τρεις επιλογές :

- **Βελτιστοποίηση Πρόσβασης Δεδομένων (Data Access Optimization)**: Όταν απαιτείται η πρόσβαση σε διευθύνσεις της μνήμης οι οποίες έχουν χαρακτηριστεί ως "βελτιστοποιούμενες" (optimizable), τότε ο ISS αποκτά απευθείας πρόσβαση στο συγκεκριμένο κομμάτι της μνήμης μέσω του Coherent Memory Server και οι αντίστοιχοι κύκλοι λειτουργίας αποκρύπτονται από τον προσομοιωτή του υλικού. Επομένως επιταχύνεται η προσομοίωση.

- **Βελτιστοποίηση Ανάκλησης Εντολής (Instruction Fetch Optimization)**: Η ενεργοποίηση της επιλογής αυτής έχει ως αποτέλεσμα η ανάκληση εντολών που είναι αποθηκευμένες σε τμήμα της μνήμης χαρακτηρισμένο ως "βελτιστοποιούμενο" (optimizable) να γίνεται απευθείας χωρίς να προσομοιώνονται οι αντίστοιχοι κύκλοι λειτουργίας του διαδρόμου από τον προσομοιωτή του υλικού. Αποτέλεσμα και πάλι η επιτάχυνση της προσομοίωσης.
- **Χρονική Βελτιστοποίηση (Time Optimization)**: Και στις δύο προηγούμενες επιλογές ο προσομοιωτής υλικού δεν προσομοιώνει τους κύκλους λειτουργίας του διαδρόμου, το ρολόι όμως που χρησιμοποιείται συνεχίζει να προχωρεί κανονικά. Κατά την ενεργοποίηση αυτής της τρίτης επιλογής δεν διατηρείται πλέον ο χρονικός συγχρονισμός των δύο προσομοιώσεων (υλικού και λογισμικού). Αντίθετα, ο προσομοιωτής υλικού ενεργοποιείται πλέον μόνο όταν απαιτείται πρόσβαση σε περιοχές της μνήμης που δεν έχουν χαρακτηριστεί ως βελτιστοποιούμενες (optimizable). Έτσι, η προσομοίωση του λογισμικού δεν καθυστερεί από την πιο αργή προσομοίωση του υλικού και επιτυγχάνεται η μεγαλύτερη επιτάχυνση της συνολικής προσομοίωσης.

Στη συνέχεια παρατίθεται ένας πίνακας, ο οποίος συνοψίζει τις αλλαγές που παρατηρούνται στη προσομοίωση του υλικού με βάση την επιλογή των κατάλληλων βελτιστοποιήσεων και το χαρακτηρισμό των αντίστοιχων τμημάτων της μνήμης.

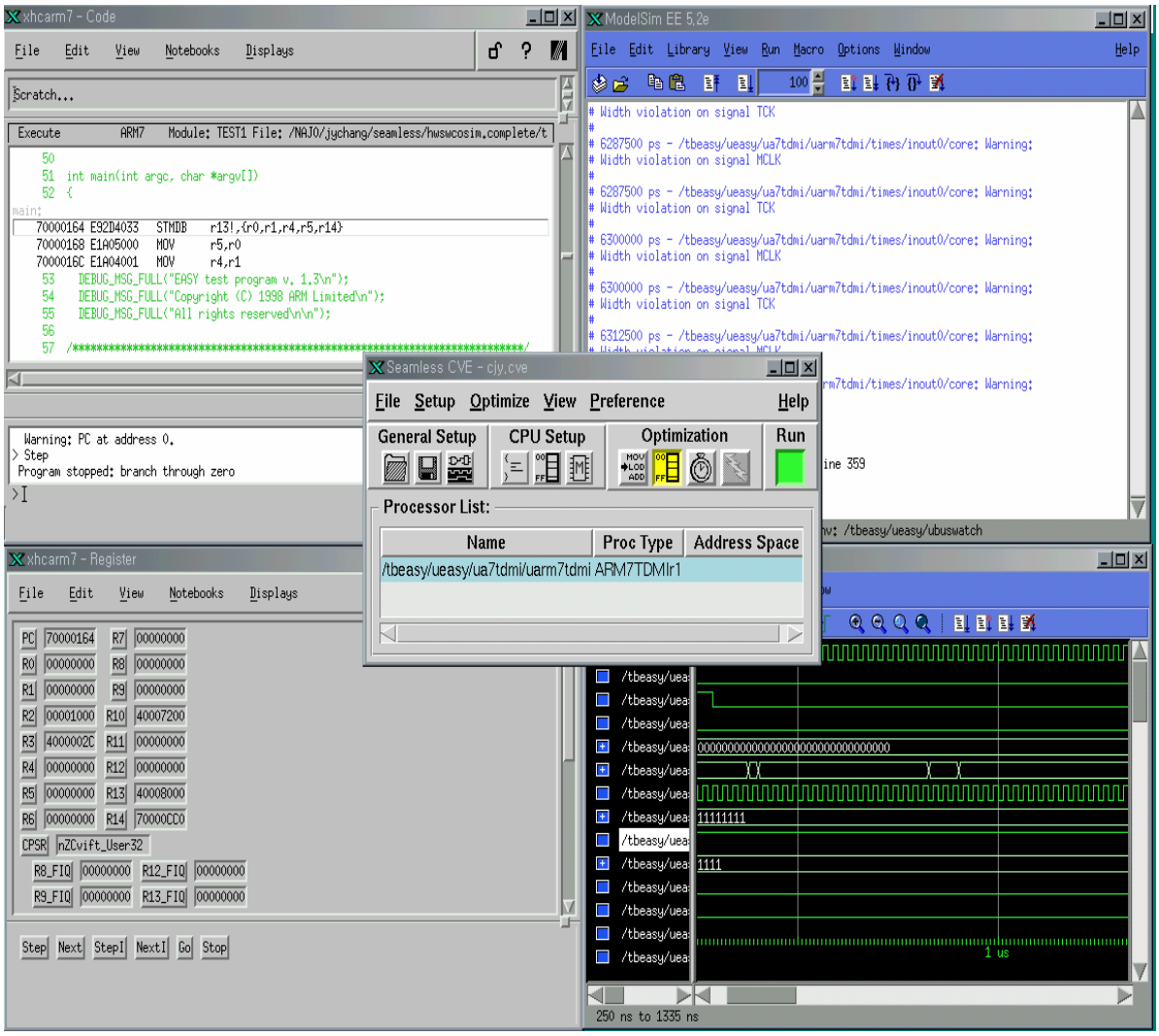
Είδος Τμήματος Μνήμης	**Είδος Βελτιστοποίησης**	**Προσομοίωση Υλικού**	**Περιγραφή**
Hardware	Οποιαδήποτε	Πλήρης Κύκλος Λειτουργίας Διαδρόμου	Προσομοιώνεται η λειτουργία του διαδρόμου και οι αντίστοιχοι κύκλοι του ρολογιού
Illegal	Οποιαδήποτε	Μήνυμα Λάθους	Παράγεται το κατάλληλο μήνυμα λάθους
Software	Time Optimization	Όχι	Η προσομοίωση του υλικού δεν προχωρά
	Οποιαδήποτε άλλη	Μόνο Κύκλοι Ρολογιού	Το ρολόι της προσομοίωσης του υλικού προχωρά αλλά δεν προσομοιώνεται καμιά λειτουργία

Optimizable	Data Access Optimization	Μόνο Κύκλοι Ρολογιού	Το ρολόι της προσομοίωσης του υλικού προχωρά αλλά δεν προσομοιώνεται καμιά λειτουργία
	Data Access + Time Optimization	Όχι	Η προσομοίωση του υλικού δεν προχωρά
	Instruction Fetch Optimization	Μόνο Κύκλοι Ρολογιού (εφόσον η πρόσβαση στη μνήμη γίνεται για ανάκληση εντολής)	Το ρολόι της προσομοίωσης του υλικού προχωρά αλλά δεν προσομοιώνεται καμιά λειτουργία
	Instruction Fetch + Time Optimization	Όχι (εφόσον η πρόσβαση στη μνήμη γίνεται για ανάκληση εντολής)	Η προσομοίωση του υλικού δεν προχωρά
	Οποιαδήποτε άλλη	Πλήρης Κύκλος Λειτουργίας Διαδρόμου	Προσομοιώνεται η λειτουργία του διαδρόμου και οι αντίστοιχοι κύκλοι του ρολογιού

Πίνακας 3.1 Βελτιστοποιήσεις Απόδοσης του Seamless CVE ™

Γίνεται λοιπόν φανερό πως ο σχεδιαστής έχει τη δυνατότητα να ρυθμίσει την ταχύτητα της προσομοίωσης κάνοντας τις κατάλληλες επιλογές.

Τέλος, παρατίθεται ένα στιγμιότυπο του περιβάλλοντος του Seamless CVE™, όπου διακρίνονται ο προσομοιωτής υλικού, ένα παράθυρο όπου μπορούμε να παρακολουθήσουμε τις μεταβολές των σημάτων, ο debugger του λογισμικού και τα περιεχόμενα των καταχωρητών του επεξεργαστή και τέλος το παράθυρου του Co-Simulation Kernel, ο οποίος όπως είδαμε ελέγχει την προσομοίωση και δίνει στο σχεδιαστή τη δυνατότητα να ρυθμίσει τις διάφορες παραμέτρους που υπάρχουν.

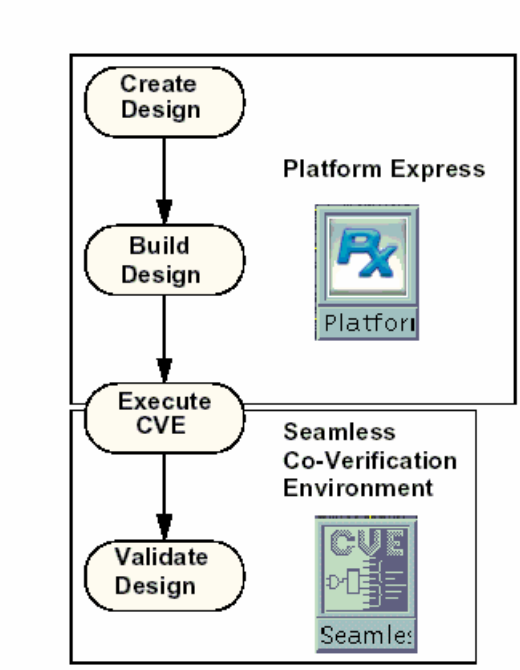


Σχήμα 3.5 Περιβάλλον εργασίας του Seamless CVE ™

3.2 Platform Express™

Το δεύτερο βασικό εργαλείο που χρησιμοποιήθηκε στην διπλωματική αυτή είναι το **Platform Express ™** και αυτό της εταιρείας Mentor Graphics ® . Το εργαλείο αυτό βοηθά το σχεδιαστή να κατασκευάσει εύκολα και αρκετά γρήγορα ένα σύστημα και στη συνέχεια να επαληθεύσει ότι λειτουργεί σωστά. Για την επαλήθευση (verification) χρησιμοποιεί το Seamless CVE ™ που παρουσιάστηκε προηγουμένως.

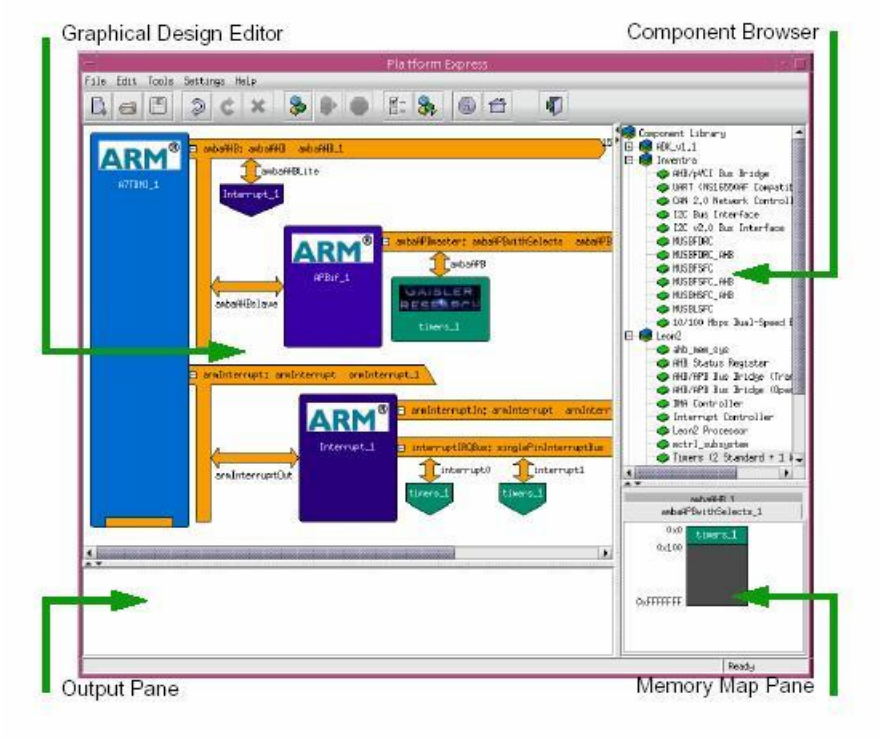
Η διαδικασία σχεδιασμού και επαλήθευσης εικονίζεται γραφικά στο επόμενο σχήμα :



Σχήμα 3.6 Διαδικασία Σχεδιασμού και Επαλήθευσης

Το Platform Express ™ παρέχει στο σχεδιαστή μια σειρά από έτοιμα τμήματα (components) υλικού, όπως επεξεργαστές, μνήμες, Ethernet controllers, UARTs και άλλα. Επίσης, του δίνει τη δυνατότητα να κατασκευάσει ένα δικό του κύκλωμα και να το ενσωματώσει στις βιβλιοθήκες του. Εκτός από το υλικό, το εργαλείο αυτό παρέχει και έτοιμο λογισμικό για κάθε κομμάτι υλικού, το οποίο χρησιμοποιείται για να ελεγχθεί η σωστή λειτουργία του αντίστοιχου τμήματος.

Το περιβάλλον εργασίας του Platform Express ™ φαίνεται στο παρακάτω σχήμα :



Σχήμα 3.7 Περιβάλλον Εργασίας του Platform Express ™

Ο χειρισμός του προγράμματος είναι αρκετά απλός. Αρχικά, ο σχεδιαστής επιλέγει τον επεξεργαστή που θα χρησιμοποιήσει. Στη συνέχεια, από το παράθυρο του Component Browser (Σχήμα 3.7) επιλέγει το υλικό που επιθυμεί να συμπεριλάβει στο σύστημα του. Αν το κομμάτι που επιλέχθηκε δεν είναι συμβατό με το διάδρομο του επεξεργαστή, το Platform Express είναι έτσι κατασκευασμένο, ώστε με βάση τις ιδιότητες του επιλεγμένου υλικού να χρησιμοποιήσει αυτόματα τις απαιτούμενες "γέφυρες" (**Bus Bridges**), προκειμένου να είναι δυνατή η επικοινωνία του επεξεργαστή με το συγκεκριμένο κομμάτι υλικού. Επίσης, αν το επιλεγμένο υλικό χρησιμοποιεί διακοπές, τότε το Platform Express χρησιμοποιεί και πάλι αυτόματα τα απαραίτητα στοιχεία έτσι, ώστε το σύστημα να λειτουργεί σωστά.Ο σχεδιαστής καλείται απλώς κάθε φορά να αντιστοιχίσει κάθε τμήμα του υλικού που χρησιμοποιείται στη διεύθυνση μνήμης που επιθυμεί. Τη δομή της μνήμης μπορεί ανά πάσα στιγμή να την παρακολουθεί στο παράθυρο Memory Map Pane (Σχήμα 3.7).

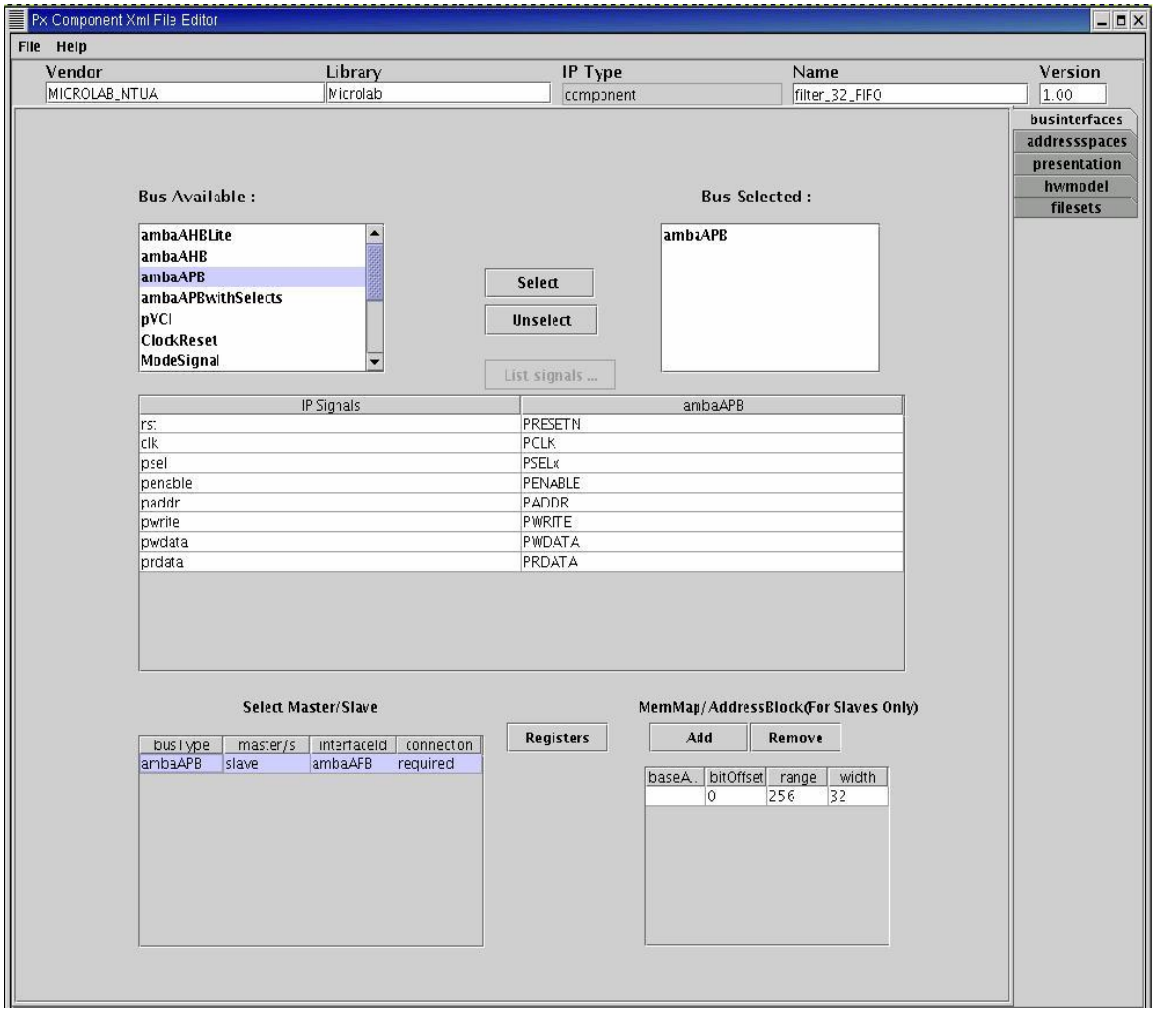
Στη συνέχεια ο σχεδιαστής εκτελεί την εντολή **Build** και το πρόγραμμα ελέγχει την ορθότητα του σχεδίου. Μετά την επιτυχή ολοκλήρωση της διαδικασίας αυτής, το σύστημα πρέπει να "επαληθευτεί". Το Platform Express ™ καλεί το Seamless CVE ™, το οποίο και ρυθμίζει αυτόματα να προσομοιώσει το σύστημα που σχεδιάστηκε χρησιμοποιώντας το λογισμικό που πρέπει.

Στο σημείο αυτό πρέπει να σημειωθεί ότι κατά την προσομοίωση του υλικού είναι δυνατό να αποκρύπτεται από τον σχεδιαστή ο κώδικας που περιγράφει και υλοποιεί κάποιο κομμάτι του υλικού. Έστω ότι κάποιος σχεδιαστής ενσωματώνει κάποιο κομμάτι στη βιβλιοθήκη και ότι κάποιος άλλος επιθυμεί να το χρησιμοποιήσει στο σχέδιο του.

Είναι ελεύθερος να το προσομοιώσει για να αποφασίσει αν το χρειάζεται, όμως αν επιθυμεί να προχωρήσει στην κατασκευή τότε δεν μπορεί, αφού δεν έχει στη διάθεση του τον κώδικα περιγραφής του συγκεκριμένου κομματιού. Με αυτό τον τρόπο προστατεύονται τα δικαιώματα του δημιουργού, καθώς για την υλοποίηση του συστήματος ο σχεδιαστής είναι υποχρεωμένος να επικοινωνήσει με το δημιουργό του αντίστοιχου κομματιού, προκειμένου να λάβει τη σχετική άδεια και να αποκτήσει τον απαιτούμενο κώδικα.

3.3 PxEdit ™

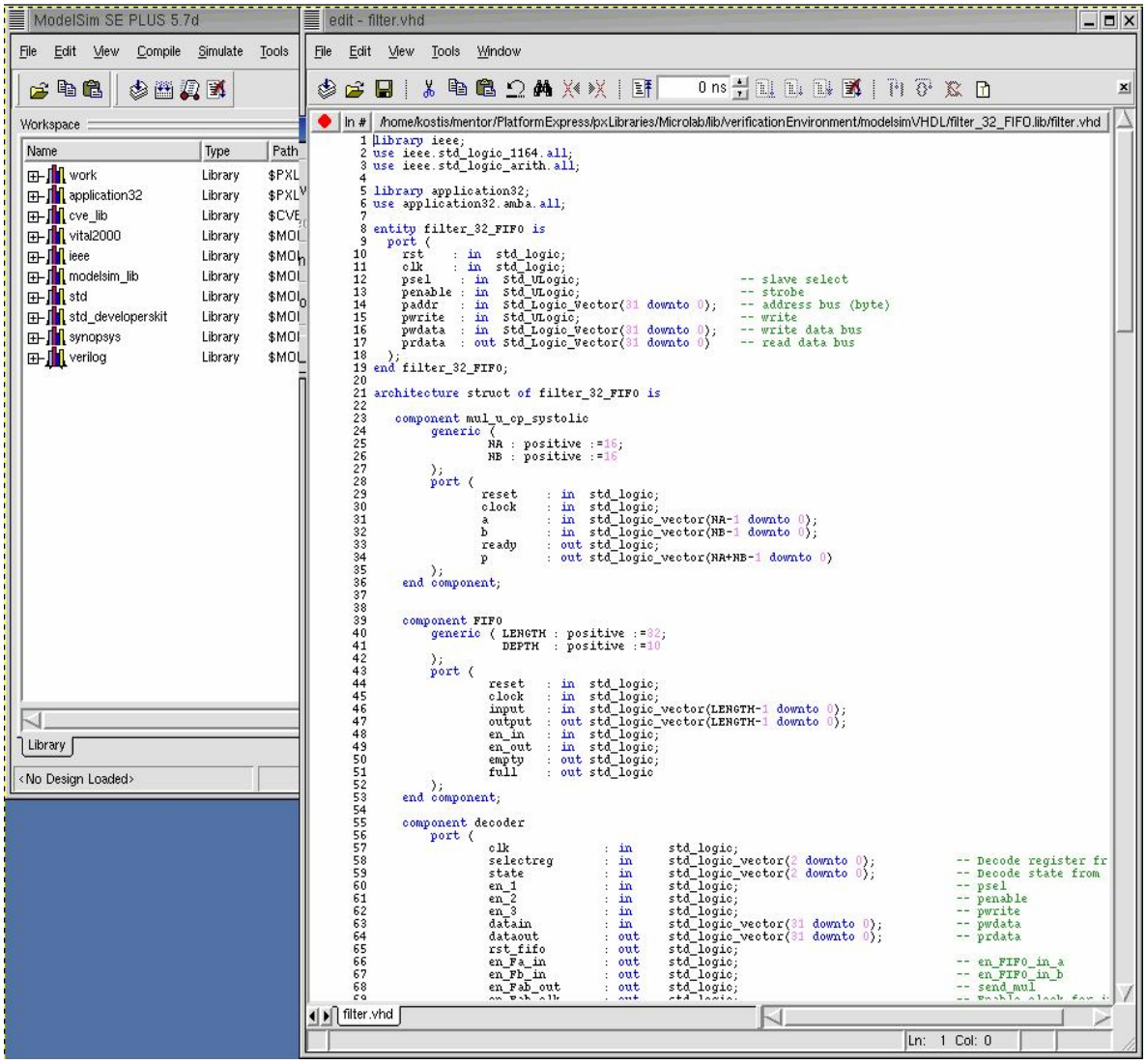
Το πρόγραμμα PxEdit ™ χρησιμοποιεί ο σχεδιαστής προκειμένου να ενσωματώσει τα κομμάτια υλικού, που έχει φτιάξει ο ίδιος, στο Platform Express ™. Συγκεκριμένα, το PxEdit διαβάζει τον κώδικα που είναι γραμμένος σε κάποια γλώσσα περιγραφής υλικού και δημιουργεί ένα XML αρχείο, το οποίο περιέχει τη λίστα με τα σήματα του υλικό, καθώς και μια σειρά από άλλες ρυθμίσεις και παραμέτρους, οι οποίες είναι απαραίτητες, προκειμένου να ενσωματωθεί το κομμάτι αυτό στη βιβλιοθήκη του Platform Express ™. Παρακάτω φαίνεται το περιβάλλον εργασίας του προγράμματος :



Σχήμα 3.8 Περιβάλλον Εργασίας του PxEdit ™

3.4 Modelsim

Για τη δημιουργία του υλικού χρησιμοποιήθηκε το πρόγραμμα **Modelsim ™** της εταιρίας Mentor Graphics ® (έκδοση 5.7d), ενώ η περιγραφή έγινε χρησιμοποιώντας τη γλώσσα **VHDL**. Το περιβάλλον εργασίας δίνεται στο παρακάτω σχήμα :



Σχήμα 3.9 Περιβάλλον Εργασίας Modelsim

3.5 Λειτουργικό Σύστημα – Γλώσσα Προγραμματισμού

Για την πραγματοποίηση της διπλωματικής όλα τα παραπάνω προγράμματα εγκαταστάθηκαν και εκτελέστηκαν στο λειτουργικό σύστημα **Red Hat Linux 7.3**. Τέλος, για την υλοποίηση του απαιτούμενου λογισμικού χρησιμοποιήθηκε η γλώσσα προγραμματισμού **C**, ενώ για τη δημιουργία των εκτελέσιμων αρχείων για επεξεργαστή ARM πραγματοποιήθηκαν οι απαιτούμενες ρυθμίσεις ώστε το Platform Express™ να χρησιμοποιήσει τους compilers του λογισμικού **ADS 1.2 for Linux**.

Βιβλιογραφία

1. Mentor Graphics® : "*Seamless CVE User's and Reference Manual*"
2. Mentor Graphics® : "*Platform Express User's Guide*"

4. Παρουσίαση Διαδρόμου AMBA

4.1 Εισαγωγή

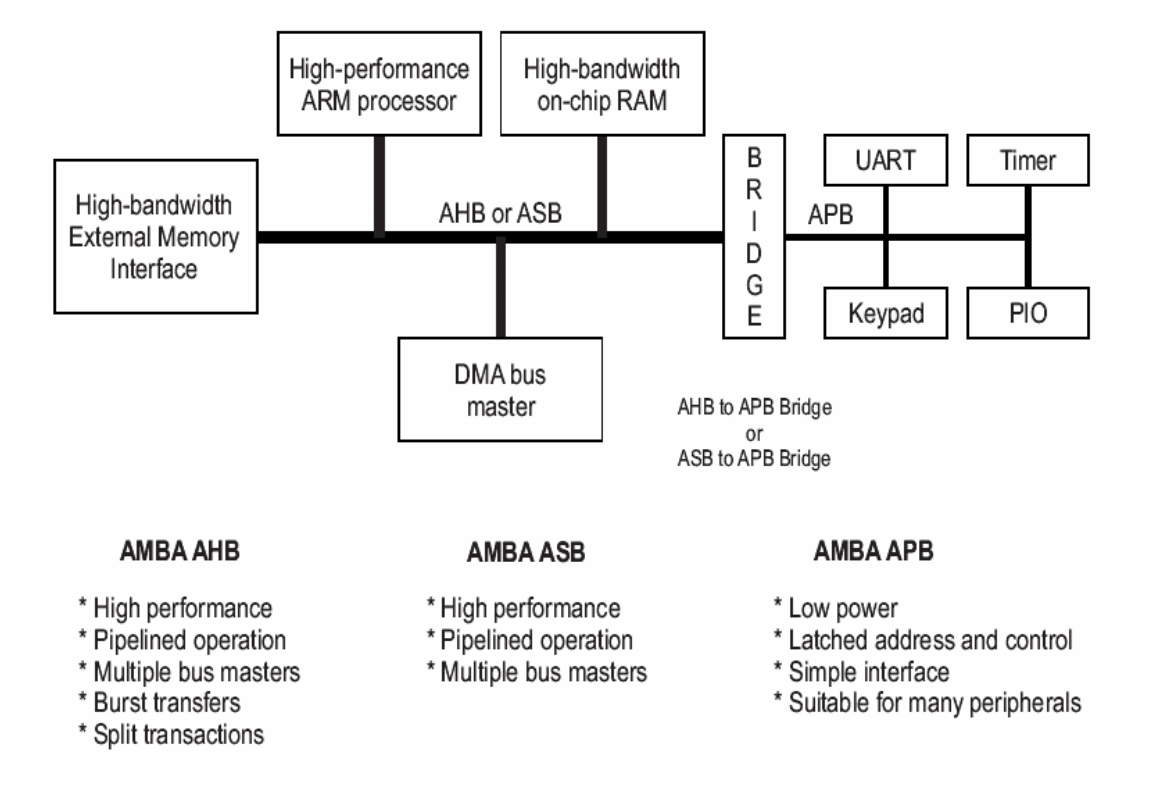
Για την επικοινωνία του επεξεργαστή με το σύστημα που σχεδιάστηκε στην παρούσα διπλωματική χρησιμοποιήθηκε ο διάδρομος της εταιρείας ARM, ο οποίος χρησιμοποιεί το πρωτόκολλο ***Advanced Microcontroller Bus Architecture (AMBA)*** και θεωρείται κατάλληλος για τη σχεδίαση embedded συστημάτων υψηλής απόδοσης. Το πρωτόκολλο αυτό (AMBA) έχει σχεδιαστεί έτσι, ώστε :

- Να είναι δυνατή η γρήγορη και σωστή διασύνδεση των τμημάτων του embedded συστήματος.
- Να είναι όσο το δυνατόν ανεξάρτητο από την τεχνολογία που χρησιμοποιείται (technology-independent). Με αυτό τον τρόπο διασφαλίζεται ότι για την υλοποίηση του συστήματος μπορούν να διασυνδεθούν περιφερειακά σχεδιασμένα με διαφορετικές τεχνολογίες.
- Να ελαχιστοποιείται το υλικό που απαιτείται για την επίτευξη της επικοινωνίας κατά την κατασκευή του κυκλώματος.

Με βάση τον ορισμό του AMBA διακρίνονται τρία διαφορετικά είδη διαδρόμων :

- **Advanced High-performance Bus (AHB)** : Ο διάδρομος αυτός θεωρείται κατάλληλος για τη διασύνδεση συστημάτων υψηλής απόδοσης που χρησιμοποιούν ρολόι υψηλής συχνότητας. Συνήθως χρησιμοποιείται ως ο κύριος διάδρομος του συστήματος που συνδέει τον επεξεργαστή με τις μνήμες (on-chip και off-chip) αλλά και τα υπόλοιπα περιφερειακά που χρησιμοποιούνται.
- **Advanced System Bus (ASB)** : Ο διάδρομος αυτός αποτελεί μια εναλλακτική λύση όταν το υπό σχεδίαση σύστημα δεν απαιτεί την υψηλή απόδοση που προσφέρει ο προηγούμενος διάδρομος. Και αυτός χρησιμοποιείται για τη σύνδεση του επεξεργαστή με τις με τις μνήμες (on-chip και off-chip) και τα υπόλοιπα περιφερειακά που χρησιμοποιούνται.
- **Advanced Peripheral Bus (APB)** : Ο διάδρομος αυτός χρησιμοποιείται μόνο για τη διασύνδεση των περιφερειακών κυκλωμάτων. Είναι κατάλληλα κατασκευασμένος, ώστε να ελαχιστοποιείται τόσο η κατανάλωση ενέργειας όσο και η πολυπλοκότητα της σύνδεσης σε αυτόν των διαφόρων κυκλωμάτων, και χρησιμοποιείται σε συνδυασμό με έναν από τους δύο προηγούμενους διαδρόμους.

Ένα τυπικό σύστημα που χρησιμοποιεί διάδρομο AMBA φαίνεται στο επόμενο σχήμα :



Σχήμα 4.1 Αρχιτεκτονική Συστήματος με AMBA διάδρομο

Ο AMBA ASB αποτέλεσε την πρώτη γενιά διαδρόμων συστήματος της εταιρίας ARM, ο οποίος έχει αντικατασταθεί πλέον σε μεγάλο βαθμό από τον AHB. Ο AMBA AHB είναι ένας διάδρομος κατάλληλα σχεδιασμένος για την ικανοποίηση των απαιτήσεων των συστημάτων υψηλής απόδοσης, τα οποία χρησιμοποιούν ρολόι υψηλής συχνότητας. Σε ένα σύστημα που χρησιμοποιεί τους διαδρόμους αυτούς έχουμε τα εξής :

- **AHB/ASB master** : Αποτελεί τον "κύριο" του διαδρόμου και μπορεί να ξεκινήσει τη διαδικασία ανάγνωσης ή εγγραφής παρέχοντας την επιθυμητή διεύθυνση και κάποια πληροφορία ελέγχου. Κάθε φορά μόνο ένας "κύριος" (master) μπορεί να χρησιμοποιεί το διάδρομο.
- **AHB/ASB slave** : Αποτελεί τον "σκλάβο" του διαδρόμου που ανταποκρίνεται στις διαδικασίες ανάγνωσης ή εγγραφής που απευθύνονται σε αυτόν. Κάθε φορά ειδοποιεί τον ενεργό "κύριο" (master) του διαδρόμου για την επιτυχή ή μη ολοκλήρωση της αντίστοιχης διαδικασίας.
- **AHB/ASB arbiter** : Αποτελεί το "διαιτητή" του διαδρόμου, ο οποίος διασφαλίζει ότι μόνο ένας "κύριος" (master) ξεκινάει κάποια διαδικασία μεταφοράς δεδομένων. Ανάλογα με τις απαιτήσεις της εφαρμογής μπορεί να υλοποιηθεί το κατάλληλο πρωτόκολλο εξυπηρέτησης των αιτήσεων χρησιμοποίησης του διαδρόμου.

- **AHB/ASB decoder** : Χρησιμοποιείται για την αποκωδικοποίηση των διευθύνσεων και την επιλογή του επιθυμητού "σκλάβου" (slave) για την επιτυχή ολοκλήρωση της διαδικασίας ανάγνωσης ή εγγραφής.

Η ύπαρξη του διαιτητή επιτρέπει τη χρησιμοποίηση πολλών επεξεργαστών ως "κυρίων" (masters) του διαδρόμου. Εκτός από τους επεξεργαστές, μπορούν να χρησιμοποιηθούν και άλλα κυκλώματα, όπως για παράδειγμα Direct Memory Access (DMA) συσκευές ή Digital Signal Processor (DSP). Ως "σκλάβοι" (slave) χαρακτηρίζονται συνήθως η μνήμη και η γέφυρα διασύνδεσης με το διάδρομο APB. Μπορούν να χρησιμοποιηθούν και άλλα περιφερειακά, είναι προτιμότερο όμως να τοποθετούνται στο διάδρομο APB.
Από το προηγούμενο σχήμα γίνεται φανερό ότι ο APB χρησιμοποιείται ως δευτερεύον διάδρομος υλοποιώντας την επικοινωνία των περιφερειακών με το διάδρομο υψηλού εύρους ζώνης του κυρίου συστήματος. Τα περιφερειακά αυτά έχουν τα εξής χαρακτηριστικά :

- Δεν απαιτούν μεγάλο εύρος ζώνης
- Η διαπροσωπεία τους υλοποιείται με τη βοήθεια memory-mapped καταχωρητών
- Η πρόσβαση σε αυτά είναι δυνατή με κατάλληλο προγραμματισμό.

4.2 Επιλογή κατάλληλου μοντέλου

Για την υλοποίηση του κύριου διαδρόμου του συστήματος μπορούν να χρησιμοποιηθούν όπως αναφέρθηκε προηγουμένως, τόσο ο AHB όσο και ο ASB. Προτείνεται όμως η χρησιμοποίηση του AHB, καθώς οδηγεί σε καλύτερη χρησιμοποίηση του εύρους ζώνης. Όσον αφορά τα περιφερειακά, μπορούν να τοποθετηθούν στον κύριο διάδρομο. Η λύση αυτή όμως δεν είναι η προτιμότερη, καθώς σε ένα σύστημα με μεγάλο αριθμό περιφερειακών συσκευών παρατηρείται συχνά αύξηση της κατανάλωσης ενέργειας και μείωση της απόδοσης.

Γενικά, ο AHB ή ο ASB χρησιμοποιείται για τη διασύνδεση :

- Κυκλωμάτων που λειτουργούν ως "κύριοι" του διαδρόμου (bus masters).
- Μνήμης (on-chip memory blocks και external memory interfaces).
- Περιφερειακών υψηλού εύρους ζώνης που χρησιμοποιούν ουρές FIFO.
- DMA slave περιφερειακά.

Αντίστοιχα, ο APB συνίσταται για τη διασύνδεση :

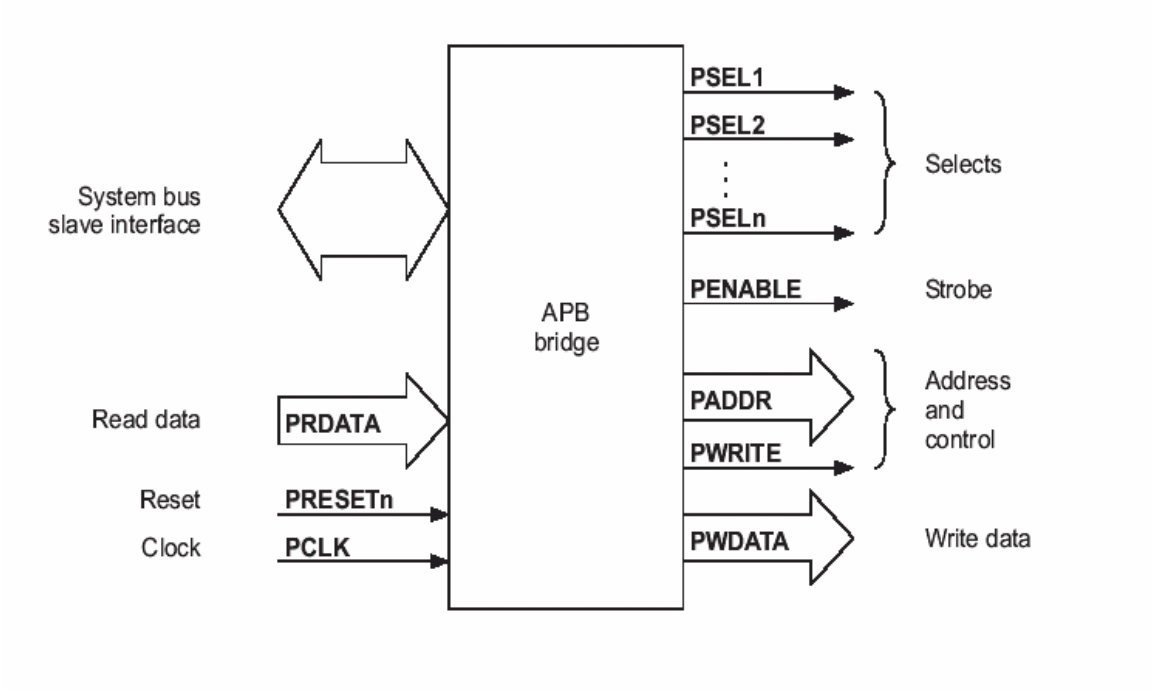
- Memory-mapped συστημάτων που λειτουργούν ως σκλάβοι του διαδρόμου (bus slaves).

- Συστημάτων με μικρές απαιτήσεις ενέργειας.
- Πολλών περιφερειακών, προκειμένου να αποφευχθεί η υπερφόρτωση του κύριου διαδρόμου.

Το σύστημα το οποίο υλοποιήθηκε στην παρούσα διπλωματική αποτελεί ένα απλό περιφερειακό. Για το λόγο αυτό σχεδιάστηκε έτσι, ώστε να είναι δυνατή η τοποθέτηση του σε διάδρομο **AMBA APB**.

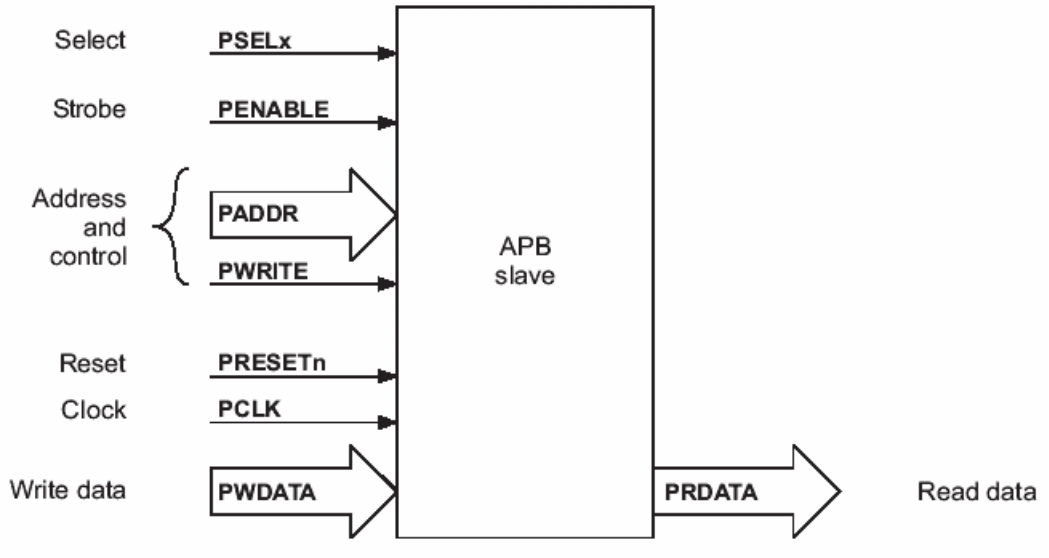
4.3 Παρουσίαση του AMBA APB

Όπως αναφέρθηκε και προηγουμένως ο APB χρησιμοποιεί μια γέφυρα για να συνδεθεί με τον κύριο διάδρομο του συστήματος, ο οποίος είναι AHB ή ASB. Η γέφυρα αυτή αποτελεί το μοναδικό "κύριο" του APB και εικονίζεται στο επόμενο σχήμα :



Σχήμα 4.2 Γέφυρα διασύνδεσης διαδρόμου APB

Αντίστοιχα, οι "σκλάβοι" του APB έχουν την παρακάτω εικονιζόμενη αρχιτεκτονική :



Σχήμα 4.3 Αρχιτεκτονική "σκλάβου" διαδρόμου APB

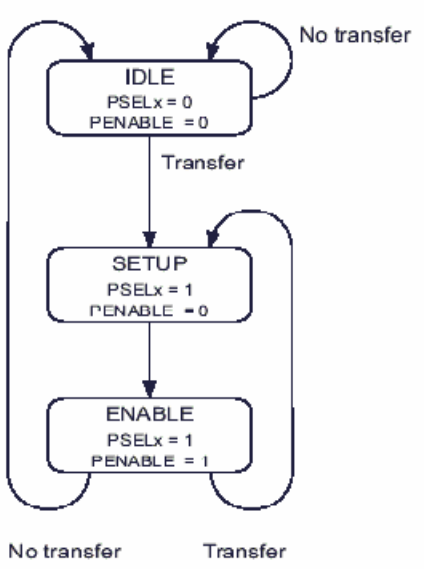
Τα σήματα που χρησιμοποιούνται στο διάδρομο APB και εικονίζονται στα παραπάνω σχήματα είναι τα εξής :

- **PCLK** : Αποτελεί το ρολόι του διαδρόμου. Για την πραγματοποίηση των μεταφορών δεδομένων στο διάδρομο χρησιμοποιείται το θετικό μέτωπο του παλμού.
- **PRESETn** : Αποτελεί το σήμα αρχικοποίησης (reset) του διαδρόμου, η οποία πραγματοποιείται όταν το σήμα αυτό γίνει **μηδέν**.
- **PADDR[31:0]**: Το σήμα αυτό περιέχει κάθε φορά τη διεύθυνση του διασυνδεδεμένου συστήματος το οποίο πρέπει να υλοποιήσει κάποια διαδικασία ανάγνωσης ή εγγραφής. Η διεύθυνση έχει μήκος 32 bits και παρέχεται κάθε φορά από τη γέφυρα διασύνδεσης.
- **PSELx** : Για κάθε "σκλάβο" του διαδρόμου υπάρχει και ένα αντίστοιχο σήμα PSELx. Η τιμή του σήματος αυτού προσδιορίζεται κάθε φορά από τον αποκωδικοποιητή που υπάρχει ενσωματωμένος στη γέφυρα διασύνδεσης του διαδρόμου APB με τον AHB ή τον ASB. Συγκεκριμένα, γίνεται 1 το σήμα που αναφέρεται στο "σκλάβο" που πρέπει να επικοινωνήσει με τον "κύριο" του διαδρόμου, ενώ των υπολοίπων παραμένει στο μηδέν.
- **PENABLE** : Το σήμα αυτό χρησιμοποιείται για να σηματοδοτεί το δεύτερο κύκλο της μεταφοράς δεδομένων, η οποία εκτελείται σε δύο κύκλους.
- **PWRITE** : Το σήμα αυτό προσδιορίζει αν πραγματοποιείται εγγραφή ή ανάγνωση. Όταν είναι **μηδέν** εκτελείται ανάγνωση, ενώ όταν είναι ίσο με **ένα** εγγραφή.
- **PRDATA[31:0]** : Το σήμα αυτό παίρνει τιμή από τον επιλεγμένο σκλάβο κατά την διάρκεια της ανάγνωσης, δηλαδή όταν το PWRITE είναι μηδέν. Έχει μήκος

32 bits και περιέχει τα δεδομένα που θα μεταφερθούν στον "κύριο" του διαδρόμου που τα ζήτησε.

- **PWDATA[31:0]** : Το σήμα αυτό παίρνει τιμή από τη γέφυρα διασύνδεσης του διαδρόμου APB με τον AHB ή τον ASB κατά τη διάρκεια της εγγραφής, όταν δηλαδή το PWRITE είναι ίσο με ένα. Έχει μήκος 32 bits και περιέχει τα δεδομένα που μεταφέρονται στον επιλεγμένο "σκλάβο" από τον ενεργό "κύριο" του διαδρόμου.

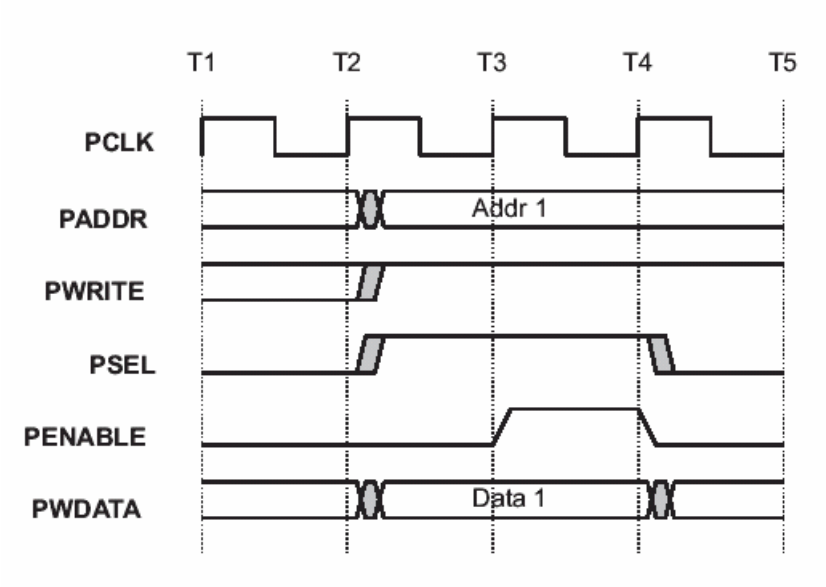
Η λειτουργία του διαδρόμου APB περιγράφεται από το παρακάτω διάγραμμα καταστάσεων :



Σχήμα 4.4 Λειτουργία Διαδρόμου APB

Όσο ο διάδρομος παραμένει αδρανής βρισκόμαστε στην κατάσταση **IDLE**. Όταν απαιτείται η μεταφορά δεδομένων, ο διάδρομος μετακινείται στην κατάσταση **SETUP**, όπου γίνεται ίσο με ένα το κατάλληλο σήμα PSELx, επιλέγεται δηλαδή ο απαιτούμενος "σκλάβος". Στην κατάσταση αυτή ο διάδρομος παραμένει για μια περίοδο του ρολογιού. Έτσι, στο επόμενο θετικό μέτωπο του ρολογιού μεταφέρεται στην κατάσταση **ENABLE**. Στο σημείο αυτό γίνεται ίσο με ένα το σήμα PENABLE, ενώ τα σήματα PWRITE, PSELx και PADDR διατηρούν την τιμή που είχαν στην προηγούμενη κατάσταση. Στην περίοδο αυτή του ρολογιού πραγματοποιείται η μεταφορά των δεδομένων και στη συνέχεια ο διάδρομος επιστρέφει στην αρχική του κατάσταση.

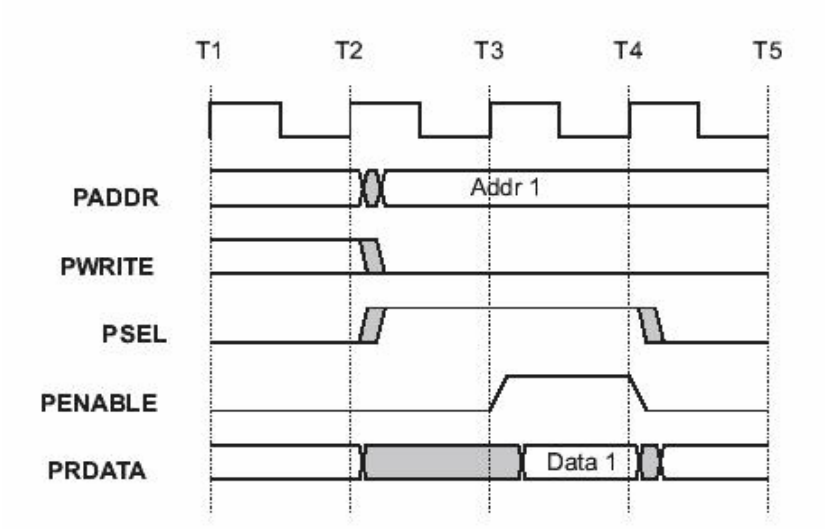
Συγκεκριμένα για την εκτέλεση εγγραφής έχουμε το εξής διάγραμμα χρονισμού :



Σχήμα 4.5 Διάγραμμα Χρονισμού Εγγραφής

Στο παραπάνω σχήμα, στον παλμό T_2 ο διάδρομος βρίσκεται στην κατάσταση SETUP, όπου και αποκτούν τις κατάλληλες τιμές τα σήματα PADDR, PWRITE, PSEL και PWDATA. Στον παλμό T_3 γίνεται ίσο με ένα το σήμα PENABLE, δηλαδή βρισκόμαστε στην κατάσταση ENABLE. Στο τέλος του παλμού αυτού ολοκληρώνεται η μεταφορά δεδομένων στον επιλεγμένο "σκλάβο" και το σήμα PENABLE γίνεται ξανά ίσο με μηδέν. Αντίθετα, τα σήματα PADDR και PWRITE δεν αλλάζουν μέχρι την επόμενη μεταφορά δεδομένων, προκειμένου να επιτευχθεί η μείωση της κατανάλωσης ενέργειας.

Αντίστοιχα, για την εκτέλεση της ανάγνωσης έχουμε το εξής σχήμα :



Σχήμα 4.6 Διάγραμμα Χρονισμού Ανάγνωσης Διαδρόμου APB

Όπως και στην περίπτωση της εγγραφής, όταν ο διάδρομος βρίσκεται στην κατάσταση SETUP, δηλαδή κατά τον παλμό T_2 σύμφωνα με το προηγούμενο σχήμα, τα σήματα

PADDR, PWRITE και PSEL αποκτούν τις κατάλληλες τιμές. Κατά τη διάρκεια του επόμενου παλμού του ρολογιού, δηλαδή όταν ο διάδρομος βρίσκεται στην κατάσταση ENABLE, ο επιλεγμένος "σκλάβος" δίνει στο σήμα PRDATA τα δεδομένα που πρέπει να μεταφερθούν. Τα δεδομένα αυτά διαβάζονται και μεταφέρονται στον ενεργό "κύριο" του διαδρόμου στο θετικό μέτωπο του επόμενου παλμού, δηλαδή στο τέλος της κατάστασης ENABLE.

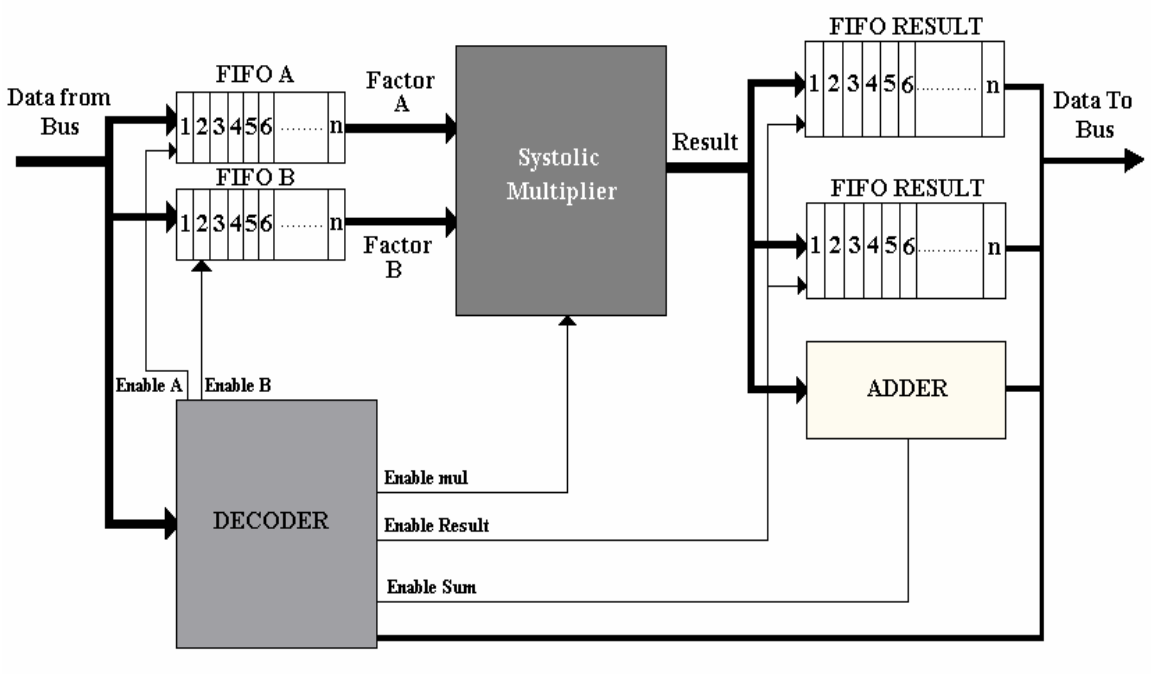
Βιβλιογραφία

1. "*AMBA ™ Specification (Rev 2.0)*", ARM

5. Υλοποίηση

5.1 Εισαγωγή

Σκοπό της παρούσας διπλωματικής αποτελεί ο σχεδιασμός ενός πολλαπλασιαστή, ο οποίος θα επικοινωνεί με έναν ARM επεξεργαστή. Πιο συγκεκριμένα, θα κατασκευαστεί το σύστημα που φαίνεται στο επόμενο σχήμα :



Σχήμα 5.1 Αρχιτεκτονική Συστήματος

Όπως γίνεται λοιπόν κατανοητό και από το παραπάνω σχήμα, ο επεξεργαστής στέλνει μέσω του διαδρόμου στο σύστημα μας τα ζευγάρια των πολλαπλασιαστών που επιθυμεί. Επίσης, στέλνει και κατάλληλες "λέξεις", οι οποίες αποκωδικοποιούνται από τον **αποκωδικοποιητή (decoder)** και μετατρέπονται στις κατάλληλες εντολές. Τα ζευγάρια των αριθμών αποθηκεύονται στις αντίστοιχες **ουρές αποθήκευσης (FIFO)**. Όταν ξεκινήσει ο πολλαπλασιασμός, οι αριθμοί αυτοί μεταφέρονται στον **πολλαπλασιαστή**. Το κάθε γινόμενο αποθηκεύεται χωρισμένο σε δύο κομμάτια, το καθένα σε μια **ουρά αποθήκευσης**, προκειμένου να μπορεί να το διαβάσει ο επεξεργαστής, ενώ ταυτόχρονα αθροίζονται μεταξύ τους χρησιμοποιώντας τον **αθροιστή** που φαίνεται στο σχήμα. Δηλαδή, ο επεξεργαστής μπορεί να διαβάσει όχι μόνο τα επιμέρους γινόμενα, αλλά και το άθροισμα τους.

Από τα παραπάνω γίνεται κατανοητό ότι το σύστημα μας μπορεί να λειτουργήσει ως ένας απλός πολλαπλασιαστής. Παράλληλα όμως μπορεί να χρησιμοποιηθεί για την υλοποίηση της συνάρτησης:

$$y = a_0x_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n$$

Δηλαδή το σύστημά μας μπορεί με τις κατάλληλες ρυθμίσεις να λειτουργήσει ως ένα φίλτρο.
Στη συνέχεια παρουσιάζονται αναλυτικά τα δομικά στοιχεία του συστήματος μας.

5.2 Συστολικός Πολλαπλασιαστής (Systolic Multiplier)

Το πιο βασικό στοιχείο του κυκλώματος μας είναι ο πολλαπλασιαστής. Η οντότητα του περιγράφεται από τον εξής κώδικα σε γλώσσα VHDL :

```
ENTITY mul_u_cp_systolic IS
   GENERIC(
      NA : positive := 32;
      NB : positive := 32
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      a     : IN     std_logic_vector (NA-1 DOWNTO 0);
      b     : IN     std_logic_vector (NB-1 DOWNTO 0);
      ready : OUT    std_logic;
      p     : OUT    std_logic_vector (NA+NB-1 DOWNTO 0)
   );

END mul_u_cp_systolic ;
```

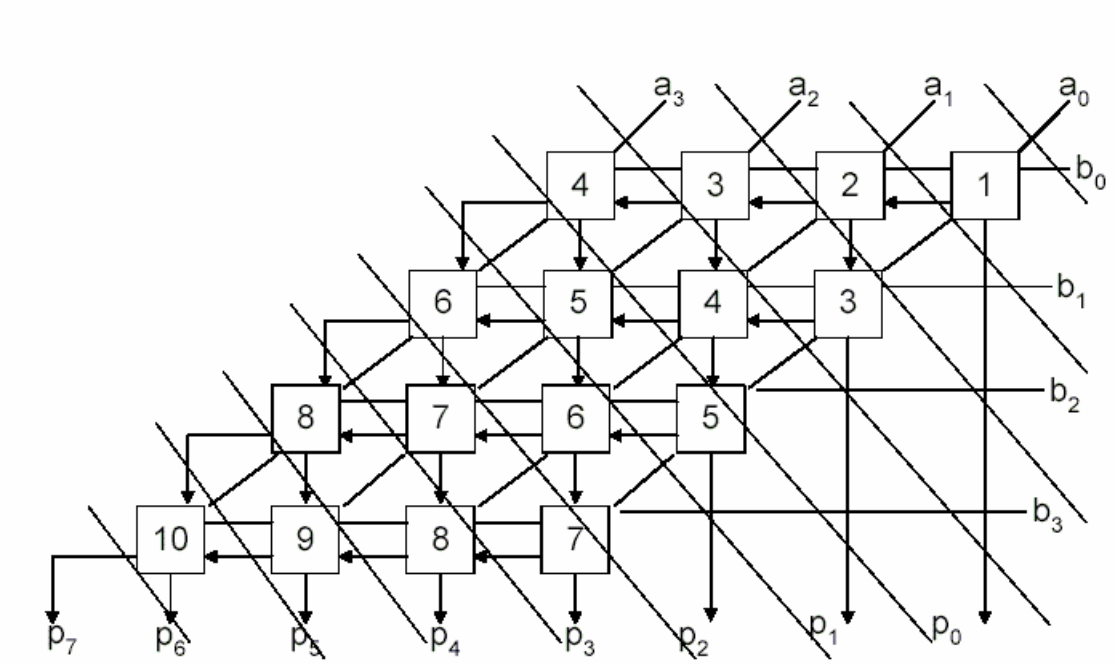
Ο πολλαπλασιαστής λοιπόν δέχεται ως είσοδο :

- Δύο δυαδικούς αριθμούς (**a,b**) μήκους 32 bits.
- Το ρολόι του συστήματος (**clock**).
- Ένα σήμα το οποίο μηδενίζει τον πολλαπλασιαστή (**reset**).

Η έξοδος του είναι :

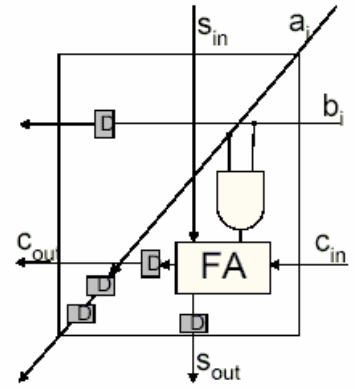
- Το γινόμενο (**p**) μήκους 64 bits.
- Ένα σήμα **(ready)**, το οποίο γίνεται 1 όταν προκύπτει το αποτέλεσμα του πολλαπλασιασμού.

Πρέπει να τονιστεί το γεγονός ότι ο πολλαπλασιαστής που σχεδιάστηκε είναι **συστολικός**. Επομένως σε κάθε περίοδο του ρολογιού διαβάζει τις καινούριες εισόδους. Όταν περάσει ένας συγκεκριμένος αριθμός παλμών, ο οποίος εξαρτάται από τον αριθμό των bits των πολλαπλασιαστέων, τότε σε κάθε παλμό προκύπτει και το αντίστοιχο αποτέλεσμα.
Η δομή του γίνεται κατανοητή στο επόμενο σχήμα :



Σχήμα 5.2 Αρχιτεκτονική Συστολικού Πολλαπλασιαστή

Όπως φαίνεται από την προηγούμενη εικόνα αλλά και από τον κώδικα (Παράρτημα Α) για την υλοποίηση του πολλαπλασιαστή έχει χρησιμοποιηθεί δομική αρχιτεκτονική (structural architecture). Συγκεκριμένα, ο πολλαπλασιαστής αποτελείται από "**κύτταρα**", τα οποία εκτελούν τις πράξεις μεταξύ των bits, και κατάλληλα τοποθετημένους καταχωρητές ολίσθησης, οι οποίοι δημιουργούν τις απαιτούμενες καθυστερήσεις.
Το κάθε κύτταρο έχει την εξής δομή :



Σχήμα 5.3 Δομή Κυττάρου Πολλαπλασιαστή

και η οντότητα του περιγράφεται ως εξής :

```
ENTITY mul_cell_systolic IS
   GENERIC(
      delay_a : positive := 1;
      delay_b : positive := 1;
      delay_s : positive := 1;
      delay_c : positive := 1
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      ai    : IN     std_logic;
      bi    : IN     std_logic;
      si    : IN     std_logic;
      ci    : IN     std_logic;
      ao    : OUT    std_logic;
      bo    : OUT    std_logic;
      so    : OUT    std_logic;
      co    : OUT    std_logic
   );

END mul_cell_systolic ;
```

Οι καταχωρητές που χρησιμοποιούνται αποτελούν ουσιαστικά μια σειρά από D Flip-flops, ώστε να παρέχουν τις απαιτούμενες καθυστερήσεις στην είσοδο τους. Η οντότητα τους είναι η παρακάτω :

```
ENTITY shift_register IS
   GENERIC(
      depth : natural := 1
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      datai : IN     std_logic;
      datao : OUT    std_logic
   );
END shift_register ;
```

Η καθυστέρηση που προσθέτουν εξαρτάται από το βάθος (**depth**). Σε κάθε σημείο λοιπόν του πολλαπλασιαστή τοποθετούνται καταχωρητές κατάλληλου βάθους, προκειμένου να επιτευχθεί η επιθυμητή καθυστέρηση. Η πλήρης περιγραφή του καταχωρητή και του κυττάρου του πολλαπλασιαστή παρατίθεται στο παράδειγμα.

5.3 Ουρά Αποθήκευσης (FIFO)

Όπως αναφέρθηκε παραπάνω, στο σύστημα μας χρησιμοποιούνται 4 ουρές FIFO. Οι δύο πρώτες για να αποθηκεύονται οι δύο πολλαπλασιαστέοι και οι άλλες δύο για να αποθηκεύεται το αποτέλεσμα. Ο τρόπος αυτός αποθήκευσης επιλέχθηκε γιατί στο διάδρομο μπορούν να μεταφερθούν λέξεις των 32 bits. Το αποτέλεσμα όμως έχει μήκος 64 bits, αφού ο κάθε πολλαπλασιαστέος έχει μήκος 32 bits. Αποθηκεύοντας λοιπόν το αποτέλεσμα σε 2 κομμάτια είναι πιο εύκολο να μεταφερθεί στον επεξεργαστή.
Η οντότητα της ουράς περιγράφεται από τον παρακάτω κώδικα :

```
ENTITY FIFO IS
   GENERIC(
      LENGTH : positive := 32;
      DEPTH  : positive := 10
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      input : IN     std_logic_vector (LENGTH-1 DOWNTO 0);
      output: OUT    std_logic_vector (LENGTH-1 DOWNTO 0);
      en_in : IN     std_logic;
      en_out: IN     std_logic;
      empty : OUT    std_logic;
      full  : OUT    std_logic
   );
END FIFO ;
```

Η ουρά λοιπόν προσδιορίζεται καταρχάς από το μήκος (**LENGTH)** και το βάθος (**DEPTH)** της. Το μήκος αναφέρεται στο μήκος των αριθμών που αποθηκεύει, ενώ το βάθος αναφέρεται στην ποσότητα των αριθμών που μπορεί να αποθηκεύσει. Στην περίπτωση μας το μήκος είναι σταθερό και ίσο με **32** bits, ενώ το βάθος μπορεί να ρυθμιστεί ανάλογα με την περίπτωση, δηλαδή πόσους πολλαπλασιασμούς επιθυμούμε να εκτελέσουμε.

Ως εισόδους δέχεται τα εξής σήματα :

- Ένα ρολόι (**clock)**.
- Ένα σήμα ασύγχρονου μηδενισμού **(reset)**, το οποίο μηδενίζει τα περιεχόμενα της ουράς και αρχικοποιεί τις απαραίτητες μεταβλητές. Προκειμένου να αποθηκευτούν δεδομένα στην ουρά είναι απαραίτητη πρώτα η αρχικοποίηση της.
- Τον αριθμό που πρόκειται να αποθηκευτεί (**input**), μήκους LENGTH bits.
- Ένα σήμα επίτρεψης (**en_in)**, το οποίο επιτρέπει στην ουρά στο θετικό μέτωπο του παλμού του ρολογιού να αποθηκεύσει τον αριθμό που βρίσκεται στην είσοδο της, εφόσον υπάρχουν κενές θέσεις.
- Ένα σήμα επίτρεψης (**en_out)**, το οποίο επιτρέπει στην ουρά στο θετικό παλμό του ρολογιού να βγάλει στην έξοδο τον αριθμό που βρίσκεται αποθηκευμένος, εφόσον υπάρχουν ακόμη αποθηκευμένοι αριθμοί.

Οι έξοδοι του κυκλώματος της ουράς είναι οι εξής :

- Ο αριθμός που "αποχωρεί" από την ουρά (**output)**, μήκους LENGTH bits.
- Ένα σήμα (**empty)**, το οποίο είναι 1 όταν δεν στην ουρά δεν υπάρχει κανένας αποθηκευμένος αριθμός.
- Ένα σήμα (**full)**, το οποίο είναι 1 όταν η ουρά έχει γεμίσει και δεν χωρά να αποθηκευτεί άλλος αριθμός.

Η πλήρης περιγραφή της ουράς παρατίθεται στο παράρτημα.

5.4 Αθροιστής (ADDER)

Για να είναι δυνατή η χρησιμοποίηση του πολλαπλασιαστή ως φίλτρο απαιτείται ένας αθροιστής, ο οποίος να αθροίζει τα επιμέρους γινόμενα. Έτσι σχεδιάστηκε η παρακάτω οντότητα :

```
entity ADDER is
    generic (
        WIDTH : positive :=64;
        DEPTH : positive :=10

    );
    port (
        reset   : in  std_logic;
        input   : in  std_logic_vector(WIDTH-1 downto 0);
        output  : out std_logic_vector(WIDTH-1 downto 0);
        overf   : out std_logic;
        enable  : in  std_logic
    );
end ADDER;
```

Ο αθροιστής έχει δύο παραμέτρους, το εύρος (**WIDTH)** και το βάθος (**DEPTH)**. Το εύρος αναφέρεται στο μήκος των αριθμών που αθροίζονται και στην περίπτωση μας είναι ίσο με **64** bits, αφού όπως έχει αναφερθεί πραγματοποιούμε πολλαπλασιασμό 32 bit αριθμών. Το βάθος προσδιορίζει πόσοι αριθμοί θα προστεθούν. Όπως γίνεται εύκολα κατανοητό το βάθος του αθροιστή ταυτίζεται με το βάθος των ουρών του συστήματος.
Οι είσοδοι του είναι οι εξής :

- Ένα σήμα ασύγχρονου μηδενισμού και αρχικοποίησης (**reset**), το οποίο μηδενίζει τον αθροιστή και αρχικοποιεί κατάλληλα τις μεταβλητές που έχουν χρησιμοποιηθεί στη σχεδίαση.
- Τον αριθμό που θα προστεθεί (**input)**, μήκους WIDTH bits.
- Ένα σήμα επίτρεψης (**enable)**, το οποίο επιτρέπει ή όχι την εκτέλεση της πρόσθεσης.

Ο αθροιστής έχει σχεδιαστεί έτσι ώστε κάθε φορά η είσοδος να προστίθεται στο προηγούμενο άθροισμα. Για αυτό το λόγο, απαραίτητη προϋπόθεση της σωστής λειτουργίας του αθροιστή αποτελεί η αρχικοποίηση του με τη βοήθεια του σήματος reset. Επίσης, όπως φαίνεται και από τον κώδικα που παρατίθεται στο παράρτημα, στο θετικό μέτωπο του σήματος επίτρεψης ο αθροιστής διαβάζει τον αριθμό που βρίσκεται στην είσοδο του και θέτει κατάλληλα τα διάφορα βοηθητικά σήματα και πραγματοποιεί την πρόσθεση στο αρνητικό μέτωπο του σήματος επίτρεψης. Οι λόγοι αυτής της επιλογής θα διασαφηνιστούν παρακάτω με τη βοήθεια των προσομοιώσεων.
Τέλος, οι έξοδοι του αθροιστή είναι οι εξής :

- Το τελικό άθροισμα **(output)**, μήκους WIDTH bits.
- Ένα σήμα **(overf**), το οποίο γίνεται 1 όταν κατά τη διάρκεια των προσθέσεων δημιουργηθεί "υπερχείλιση" (**overflow**), όταν δηλαδή το άθροισμα απαιτεί περισσότερα από WIDTH bits για να παρασταθεί. Στην περίπτωση αυτή το σήμα

αυτό ειδοποιεί ότι το άθροισμα που υπάρχει στην έξοδο του αθροιστή δεν είναι σωστό.

5.5 Αποκωδικοποιητής (DECODER)

Προκειμένου να επιτευχθεί ο έλεγχος του συστήματος από τον επεξεργαστή κρίθηκε απαραίτητη η υλοποίηση ενός αποκωδικοποιητή, ο οποίος λαμβάνει δεδομένα από τον επεξεργαστή μέσω του διαδρόμου, τα επεξεργάζεται και ρυθμίζει κατάλληλα τη λειτουργία του συστήματος. Η οντότητα του αποκωδικοποιητή είναι η παρακάτω :

```
entity decoder is
    port (
        clk          : in     std_logic;
        selectreg    : in     std_logic_vector(2 downto 0);
        state        : in     std_logic_vector(2 downto 0);
        en_1         : in     std_logic;
        en_2         : in     std_logic;
        en_3         : in     std_logic;
        datain       : in     std_logic_vector(31 downto 0);
        dataout      : out    std_logic_vector(31 downto 0);
        rst_fifo     : out    std_logic;
        en_Fa_in     : out    std_logic;
        en_Fb_in     : out    std_logic;
        en_Fab_out   : out    std_logic;
        en_Fab_clk   : out    std_logic;
        en_mul       : out    std_logic;
        Fres_rd_in   : in     std_logic;
        en_Fres_in   : out    std_logic;
        en_Fresh_out: out    std_logic;
        en_Fresl_out: out    std_logic;
        en_Fres_clk : out    std_logic;
        reg_a        : inout  std_logic_vector(31 downto 0);
        reg_b        : inout  std_logic_vector(31 downto 0);
        control      : inout  std_logic_vector(31 downto 0);
        reg_resulth : in     std_logic_vector(31 downto 0);
        reg_resultl : in     std_logic_vector(31 downto 0);
        sig_final    : in     std_logic;
        sumh         : in     std_logic_vector(31 downto 0);
        suml         : in     std_logic_vector(31 downto 0);
        overflow     : in     std_logic
    );
end decoder;
```

Αναλυτικά οι είσοδοι του αποκωδικοποιητή είναι οι παρακάτω :

- Το ρολόι του συστήματος (**clock).**
- Δύο σήματα (**selectreg** και **state**), μήκους 3 bits, τα οποία μεταφέρουν πληροφορία στον αποκωδικοποιητή και εξηγούνται παρακάτω.
- Τρία σήματα επίτρεψης (**en_1**, **en_2** και **en_3).**
- Τα δεδομένα που διαβάζονται από το διάδρομο (**datain**) μήκους 32 bits.
- Το σήμα **Fres_rd_in** που γίνεται 1 όταν στον πολλαπλασιαστή έχει παραχθεί το γινόμενο. Ειδοποιεί δηλαδή τον αποκωδικοποιητή για την ολοκλήρωση του πολλαπλασιασμού.

- Τα δύο τμήματα του αποτελέσματος (**reg_resulth** και **reg_resultl)**, που περιέχουν τα 32 MSB και LSB του γινομένου αντίστοιχα.
- Το σήμα **sig_final**, που όταν γίνει 1 ειδοποιεί τον αποκωδικοποιητή ότι έχει ολοκληρωθεί η εισαγωγή των γινομένων στις ουρές αποθήκευσης και η παραγωγή του τελικού αθροίσματος. Αυτό συμβαίνει όταν γεμίσουν οι ουρές εξόδου.
- Τα δύο τμήματα του τελικού αθροίσματος **(sumh** και **suml**), που περιέχουν τα 32 MSB και LSB του τελικού αθροίσματος αντίστοιχα.
- Το σήμα **overflow** που όταν είναι 1 σηματοδοτεί την ύπαρξη υπερχείλισης στο τελικό άθροισμα.

Αντίστοιχα, οι έξοδοι είναι οι εξής :

- Τα δεδομένα που γράφονται στο διάδρομο (**dataout)** για να μεταφερθούν στον επεξεργαστή, μήκους 32 bits.
- Το σήμα **rst_fifo**, το οποίο αρχικοποιεί τις ουρές και τα υπόλοιπα κυκλώματα του συστήματος.
- Τα σήματα **en_Fa_in**, **en_Fb_in**, **en_Fres_in**, τα οποία αποτελούν τα σήματα επίτρεψης εισόδου των ουρών αποθήκευσης των 2 πολλαπλασιαστών και του αποτελέσματος αντίστοιχα.
- Τα σήματα **en_Fab_out**, **en_Fresh_out**, **en_Fresl_out**, τα οποία αποτελούν τα σήματα επίτρεψης εξόδου των ουρών αποθήκευσης των 2 πολλαπλασιαστών και του αποτελέσματος αντίστοιχα.
- Τα σήματα **en_Fab_clk**, και **en_Fres_clk**, τα οποία συνδυάζονται με το ρολόι του συστήματος, προκειμένου να δημιουργηθεί το κατάλληλο ρολόι για την είσοδο στις ουρές αποθήκευσης των πολλαπλασιαστέων και την έξοδο από τις ουρές του αποτελέσματος αντίστοιχα.
- Το σήμα **en_mul**, το οποίο χρησιμοποιείται για την εκκίνηση του πολλαπλασιαστή.

Επίσης, υπάρχουν και σήματα στα οποία έχει αποδοθεί ρόλος εισόδου-εξόδου, προκειμένου να παίρνουν τιμή από το διάδρομο αλλά ταυτόχρονα να μπορούν και να μεταφέρουν το περιεχόμενο τους στο διάδρομο. Αυτά είναι :

- Τα **reg_a** και **reg_b**, μήκους 32 bits, που αντιστοιχούν στους πολλαπλασιαστέους α και β.
- Το **control** μήκους 32 bits, που μεταφέρει πληροφορία από και προς τον επεξεργαστή.

5.5.1 Σήματα Επίτρεψης

Για τη σύνδεση του συστήματος με τον επεξεργαστή ARM χρησιμοποιήθηκε ο διάδρομος **AMBA** (*Advanced Microcontroller Bus Architecture*) και συγκεκριμένα ο **APB** (*Advanced Peripheral Bus*) όπως είδαμε προηγουμένως. Στο πρωτόκολλο αυτό

υπάρχουν όπως έχουμε δει τρία σήματα που προσδιορίζουν την επικοινωνία του επεξεργαστή με το σύστημά μας. Καταρχάς, το σήμα **PSEL**, το οποίο γίνεται 1 μόνο όταν ο επεξεργαστής επικοινωνεί με το δικό μας σύστημα. Όταν το PSEL γίνει 1 διακρίνουμε 2 φάσεις. Μία κατά την οποία το σήμα **PENABLE** παραμένει στο 0 (για μια περίοδο του ρολογιού) και μία όταν το **PENABLE** γίνεται 1, οπότε και εκτελείται η διαδικασία ανάγνωσης ή εγγραφής.
Εμάς λοιπόν μας ενδιαφέρει να διαβάζουμε ή να μεταφέρουμε δεδομένα στο διάδρομο μόνο όταν προορίζονται για το σύστημα μας **(PSEL = 1)** και επιτρέπεται η ανάγνωση ή η εγγραφή (**PENABLE = 1)**. Έχουμε λοιπόν δύο σήματα επίτρεψης, στα οποία αντιστοιχούνται τα σήματα **en_1** και **en_2**.
Τέλος, όπως γίνεται εύκολα κατανοητό, ένα άλλο είδος επίτρεψης αποτελεί το αν εκτελείται ανάγνωση ή εγγραφή, πληροφορία η οποία σύμφωνα με τις προδιαγραφές του διαδρόμου περιέχεται στο σήμα **PWRITE**. Το σήμα αυτό αντιστοιχίζεται στο **en_3**.

5.5.2 Έλεγχος Λειτουργίας Συστήματος

Η κατάσταση λειτουργίας του συστήματος προσδιορίζεται κάθε στιγμή από το σήμα **state**. Συγκεκριμένα, έχουμε τον παρακάτω πίνακα :

State	***Κατάσταση Λειτουργίας***
000	Αρχικοποίηση
001	Λήψη των τιμών του πολλαπλασιαστέου A
010	Λήψη των τιμών του πολλαπλασιαστέου B
011	Πραγματοποίηση των πολλαπλασιασμών, αποθήκευση των αποτελεσμάτων στις ουρές και πραγματοποίηση των προσθέσεων
100	Διάβασμα των αποτελεσμάτων

Πίνακας 5.1 Τιμές Σήματος state

Αναλυτικότερα, κατά τη φάση αρχικοποίησης το σήμα **rst_fifo** τίθεται στο 0, προκειμένου να αρχικοποιηθούν τα υπόλοιπα κυκλώματα του συστήματος. Στις επόμενες δύο καταστάσεις, γίνονται 1 τα σήματα **en_Fa_in** και **en_Fb_in** αντίστοιχα, ώστε να αποθηκευτούν οι αριθμοί στις αντίστοιχες ουρές.
Κατά την τρίτη φάση λειτουργίας, γίνεται αρχικά 1 το σήμα **en_Fab_out**, προκειμένου να επιτραπεί η έξοδος των αριθμών από τις ουρές αποθήκευσης. Στον επόμενο παλμό του ρολογιού γίνεται 1 και το σήμα **en_mul**, ώστε να ξεκινήσει ο πολλαπλασιασμός. Η καθυστέρηση αυτή είναι απαραίτητη προκειμένου να προλάβουν οι αριθμοί να μεταφερθούν από τις εξόδους των ουρών αποθήκευσης στις εισόδους του πολλαπλασιαστή. Όταν στη συνέχεια αρχίσουν να παράγονται τα αποτελέσματα

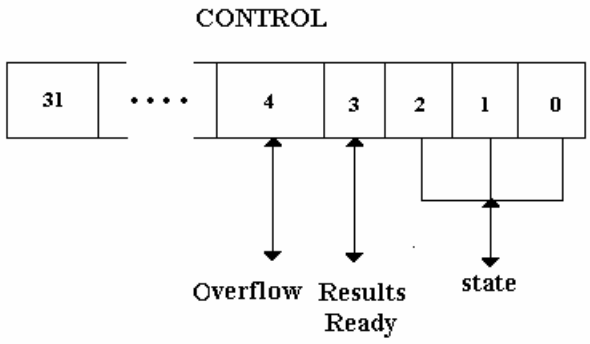
(**Fres_rd_in = 1**), τότε στον επόμενο παλμό γίνεται 1 το σήμα **en_Fres_in** για να επιτραπεί η αποθήκευση του γινομένου στις ουρές. Ταυτόχρονα, ξεκινάει και η πρόσθεση των αποτελεσμάτων. Και στην περίπτωση αυτή, η καθυστέρηση του ενός παλμού είναι απαραίτητη, προκειμένου να προλάβει το γινόμενο να μεταφερθεί από την έξοδο του πολλαπλασιαστή στην είσοδο των ουρών αποθήκευσης και στην είσοδο του αθροιστή.

Στην τέταρτη φάση λειτουργίας γίνονται 1 τα σήματα **en_Fresh_out** και **en_Fresl_out**, ώστε να επιτραπεί η έξοδος των αποθηκευμένων αποτελεσμάτων από τις ουρές και το διάβασμα τους από τον επεξεργαστή. Βέβαια τα σήματα αυτά δεν γίνονται ταυτόχρονα 1, αφού στον διάδρομο μπορούν να τοποθετηθούν μόνο 32 bits κάθε φορά. Το πότε ενεργοποιείται η κάθε ουρά εξαρτάται από τα σήματα επίτρεψης en_1, en_2 και από το ποιο κομμάτι του αποτελέσματος επιθυμεί να διαβάσει ο επεξεργαστής.

Όπως έχει αναφερθεί, η πληροφορία ελέγχου μεταφέρεται από και προς τον επεξεργαστή με το σήμα **control**. Η βασική πληροφορία λοιπόν σχετικά την κατάσταση λειτουργίας του κυκλώματος, δηλαδή η τιμή του σήματος state, ενθυλακώνεται στο σήμα **control** και συγκεκριμένα στα 3 LSB. Όμως πληροφορία ενθυλακώνεται και στα επόμενα δύο bits του σήματος control.

Συγκεκριμένα, όταν γεμίσουν οι ουρές εξόδου και τα αποτελέσματα είναι έτοιμα να διαβαστούν τότε το **3^{o}** bit του control γίνεται 1. Αντίστοιχα, το **4^{o}** bit γίνεται 1 αν υπάρξει υπερχείλιση στο άθροισμα των γινομένων.

Έχουμε δηλαδή την εξής εικόνα :



Σχήμα 5.4 Δομή Σήματος Control

5.5.3 Εγγραφές – Αναγνώσεις

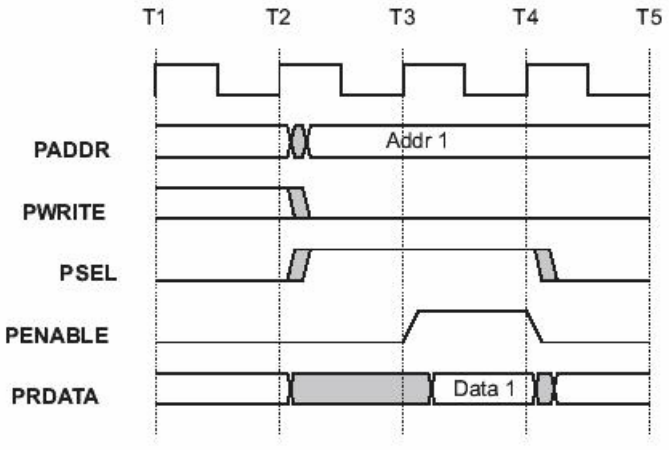
Ο αποκωδικοποιητής χρησιμοποιεί το σήμα **selectreg** για να προσδιορίζει το είδος της πληροφορίας που λαμβάνεται ή που πρέπει να τοποθετηθεί στο διάδρομο. Οι διάφορες περιπτώσεις συνοψίζονται στον παρακάτω πίνακα :

Selectreg	***en_3 (PWRITE)***	***Λειτουργία***	***Περιγραφή***
000	0	dataout <= control	Ανάγνωση Πληροφορίας Ελέγχου
000	1	control <= datain	Εγγραφή Πληροφορίας Ελέγχου
001	0	dataout <= reg_a	Ανάγνωση Πολλαπλασιαστέου Α
001	1	reg_a <= datain	Εγγραφή Πολλαπλασιαστέου Α
010	0	dataout <= reg_b	Ανάγνωση Πολλαπλασιαστέου Β
010	1	reg_b <= datain	Εγγραφή Πολλαπλασιαστέου Β
011	0	dataout <= reg_resulth	Ανάγνωση των 32 MSB του γινομένου
100	0	dataout <= reg_resultl	Ανάγνωση των 32 LSB του γινομένου
101	0	dataout <= sumh	Ανάγνωση των 32 MSB του τελικού αθροίσματος
110	0	dataout <= suml	Ανάγνωση των 32 LSB του τελικού αθροίσματος

Πίνακας 5.2 Χρήση του σήματος selectreg

Βλέπουμε, λοιπόν, πως ο αποκωδικοποιητής κατά τη διάρκεια των εγγραφών προωθεί τα δεδομένα από το διάδρομο στα κατάλληλα σήματα και αντίστροφα κατά τη διάρκεια των αναγνώσεων προωθεί στο διάδρομο τις τιμές των απαιτούμενων σημάτων. Σύμφωνα και με τις προδιαγραφές του διαδρόμου που αναφέρθηκαν παραπάνω, οι λειτουργίες αυτές εκτελούνται υπό την προϋπόθεση ότι τα σήματα en_1 (PSEL) και en_2 (PENABLE) είναι και αυτά 1.

Τέλος, ο αποκωδικοποιητής έχει σχεδιαστεί να δουλεύει όταν το ρολόι του διαδρόμου που λαμβάνει ως είσοδο είναι **μηδέν**. Αυτό οφείλεται στο γεγονός ότι σύμφωνα με τις προδιαγραφές του διαδρόμου κατά την εκτέλεση της ανάγνωσης διαβάζονται τα δεδομένα που έχουν τοποθετηθεί στο διάδρομο πριν από το αρνητικό μέτωπο του σήματος PENABLE, όπως φαίνεται και στο επόμενο σχήμα :



Σχήμα 5.5 Διάγραμμα Χρονισμού Ανάγνωσης Διαδρόμου APB

Όταν λοιπόν πρέπει να μεταφερθούν τα αποτελέσματα από τις ουρές εξόδου στον επεξεργαστή, ενεργοποιούμε το ρολόι των ουρών αυτών στο αρνητικό κομμάτι του παλμού T_2 του σχήματος.
Έχουμε δηλαδή :

en_Fres_clk <= en_1 and not en_2 and not en_3;

Έτσι στο αρνητικό κομμάτι του παλμού T_3 το αποτέλεσμα έχει μεταφερθεί από τις ουρές εξόδου στα σήματα **reg_resulth** και **reg_resultl** και τοποθετούνται στο διάδρομο από τον αποκωδικοποιητή, με αποτέλεσμα να διαβάζονται από τον επεξεργαστή. Αν χρησιμοποιούσαμε το θετικό κομμάτι του ρολογιού, δεν θα προλαβαίναμε να τοποθετήσουμε στο διάδρομο έγκαιρα τα απαραίτητα δεδομένα για την ανάγνωση. Αντίθετα, κατά την ανάγνωση από το διάδρομο δεν υπάρχει αντίστοιχη απαίτηση. Έτσι το ρολόι των ουρών αποθήκευσης των πολλαπλασιαστέων ενεργοποιείται στο αρνητικό κομμάτι του παλμού T_3. Έχουμε δηλαδή :

en_Fab_clk <= en_1 and en_2 and en_3;

5.6 Φίλτρο

Όλα τα παραπάνω κυκλώματα αποτελούν τμήματα της γενικότερης οντότητας του φίλτρου, η οποία περιγράφεται από τον εξής κώδικα :

```
entity filter_32_FIFO is
  port (
    rst     : in  std_logic;
    clk     : in  std_logic;
    psel    : in  Std_ULogic;
    penable : in  Std_ULogic;
    paddr   : in  Std_Logic_Vector(31 downto 0);
    pwrite  : in  Std_ULogic;
    pwdata  : in  Std_Logic_Vector(31 downto 0);
    prdata  : out Std_Logic_Vector(31 downto 0)
  );
end filter_32_FIFO;
```

Οι είσοδοι της οντότητας είναι :

- Ένα σήμα αρχικοποίησης (**rst**), το οποίο ταυτίζεται με το σήμα αρχικοποίησης του διαδρόμου.
- Το ρολόι **(clk)** του διαδρόμου.
- Τα σήματα **psel**, **penable**, και **pwrite**, των οποίων η λειτουργία εξηγήθηκε παραπάνω.
- Το σήμα **paddr** μήκους 32 bits, το οποίο περιέχει τη διεύθυνση του τμήματος με το οποίο επικοινωνεί ο επεξεργαστής μέσω του διαδρόμου. Τα bits 5-3 του σήματος αυτού ταυτίζονται με το σήμα selectreg του αποκωδικοποιητή.

- Τα δεδομένα (**pwdata**) που στέλνει ο επεξεργαστής.

Η έξοδος είναι το σήμα **prdata** μήκους 32 bits, το οποίο περιέχει κάθε φορά τα δεδομένα που τοποθετούνται στο διάδρομο, προκειμένου να τα διαβάσει ο επεξεργαστής.
Για τη σχεδίαση του φίλτρου χρησιμοποιήθηκε, όπως φαίνεται και από τον κώδικα που παρατίθεται στο παράρτημα, "δομική αρχιτεκτονική" (structural architecture). Δηλαδή, χρησιμοποιούνται όλα τα παραπάνω κυκλώματα με τη βοήθεια κατάλληλων σημάτων.

Βιβλιογραφία

1. Κ.Ζ.Πεκμεστζή : "*Ψηφιακά Συστήματα VLSI*"

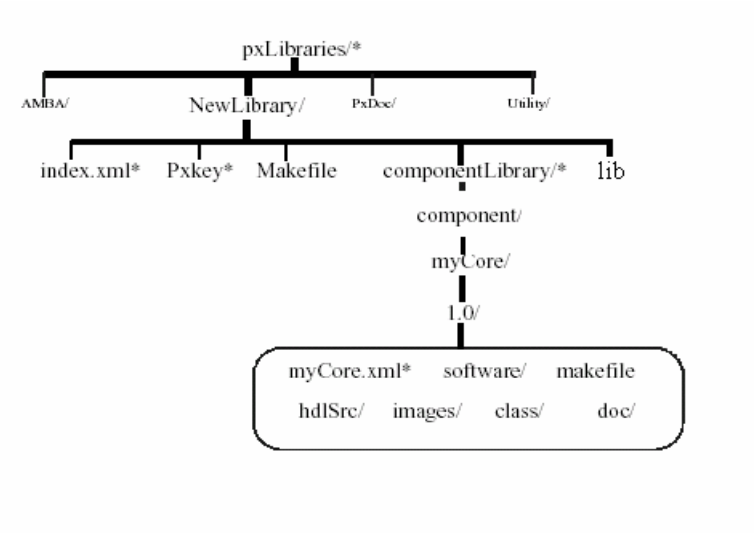
6. Ενσωμάτωση Συστήματος στο Platform Express

6.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται η διαδικασία ενσωμάτωσης του συστήματος που σχεδιάστηκε στο εργαλείο Platform Express™. Για την πραγματοποίηση της ενσωμάτωσης χρησιμοποιήθηκε το εργαλείο PxEdit, χρειάστηκαν όμως και κάποιες ακόμα ρυθμίσεις, οι οποίες παρουσιάζονται στη συνέχεια.

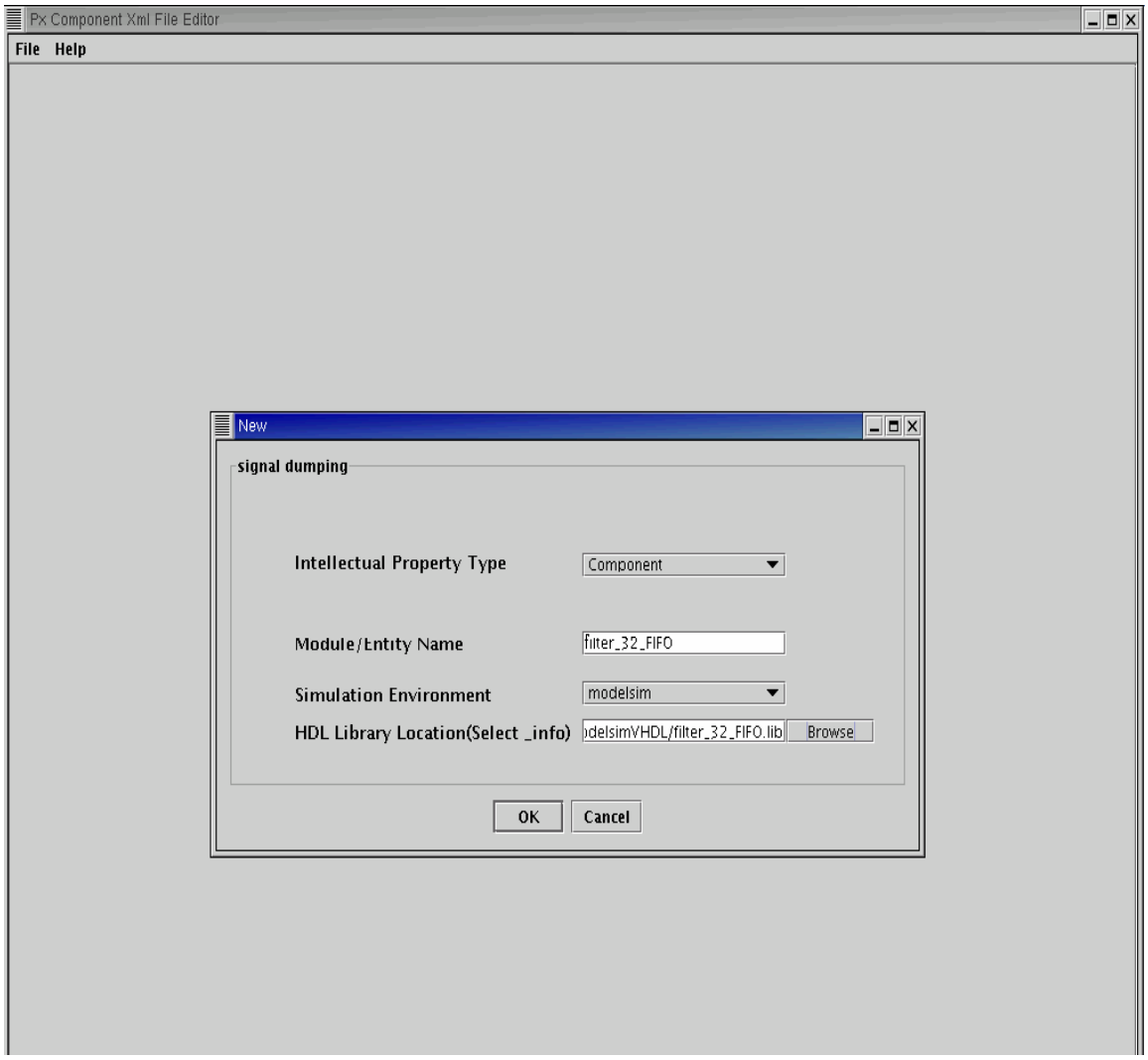
6.2 Χρήση του εργαλείου PxEdit

Αρχικά πρέπει να δημιουργήσουμε μια κατάλληλη δομή φακέλων και αρχείων. Αυτή πρέπει να ακολουθεί το εξής σχήμα :



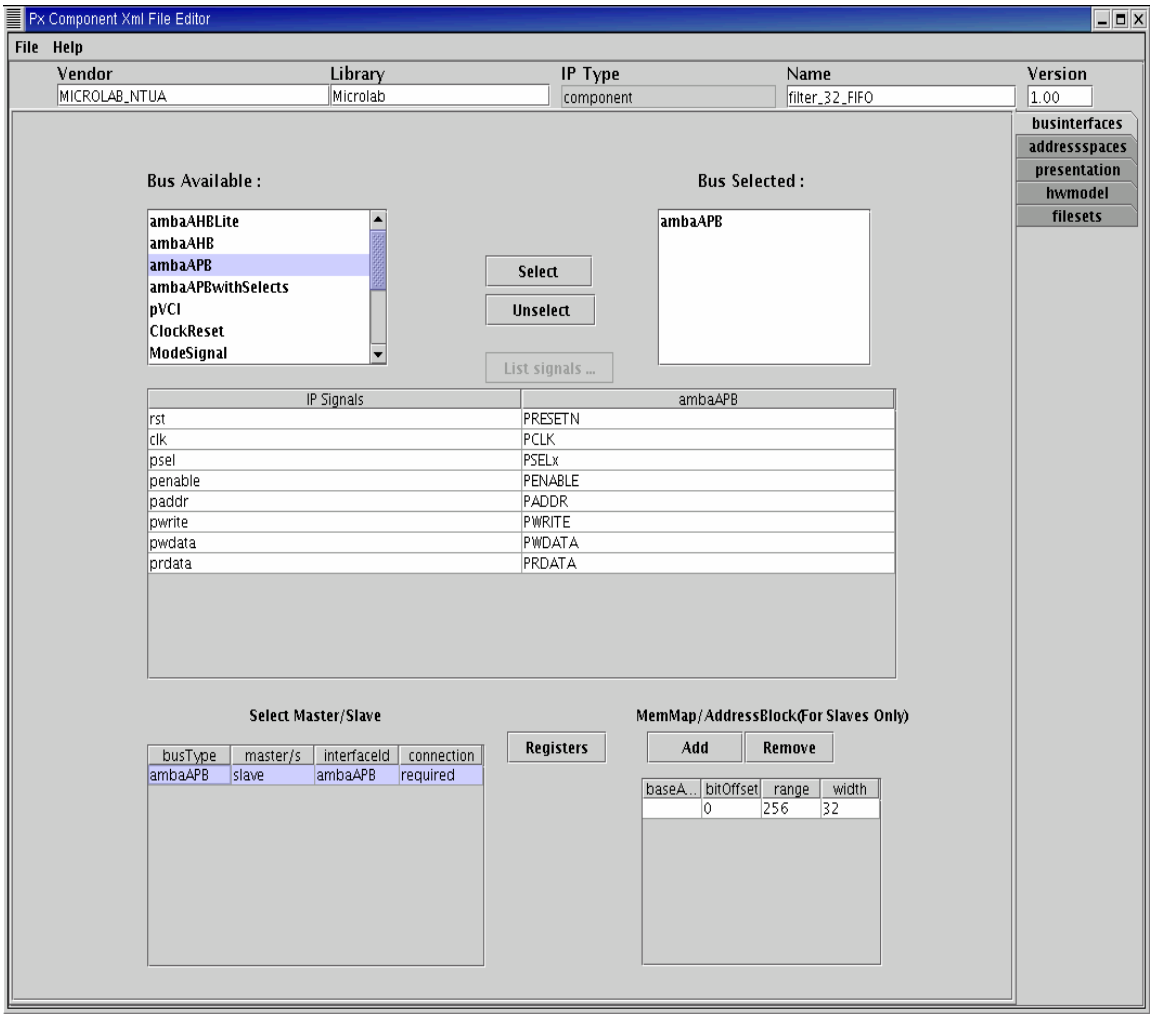
Σχήμα 6.1 Δομή Φακέλων

Στη συνέχεια τρέχουμε το πρόγραμμα PxEdit, στο οποίο δηλώνουμε ότι θα κατασκευάσουμε ένα καινούριο τμήμα υλικού (component) και παρέχουμε την τοποθεσία της βιβλιοθήκης όπου περιέχεται ο κώδικας περιγραφής του υλικού.



Σχήμα 6.2

Είναι απαραίτητο το όνομα της οντότητας που δίνουμε να ταυτίζεται πλήρως με το όνομα της οντότητας που έχει χρησιμοποιηθεί κατά την σχεδίαση. Στη συνέχεια το πρόγραμμα επεξεργάζεται τα αρχεία της βιβλιοθήκης, προκειμένου να βρει τα σήματα που χρησιμοποιούνται στον ορισμό της οντότητας.
Μόλις η διαδικασία αυτή ολοκληρωθεί έχουμε πλέον την εξής εικόνα :



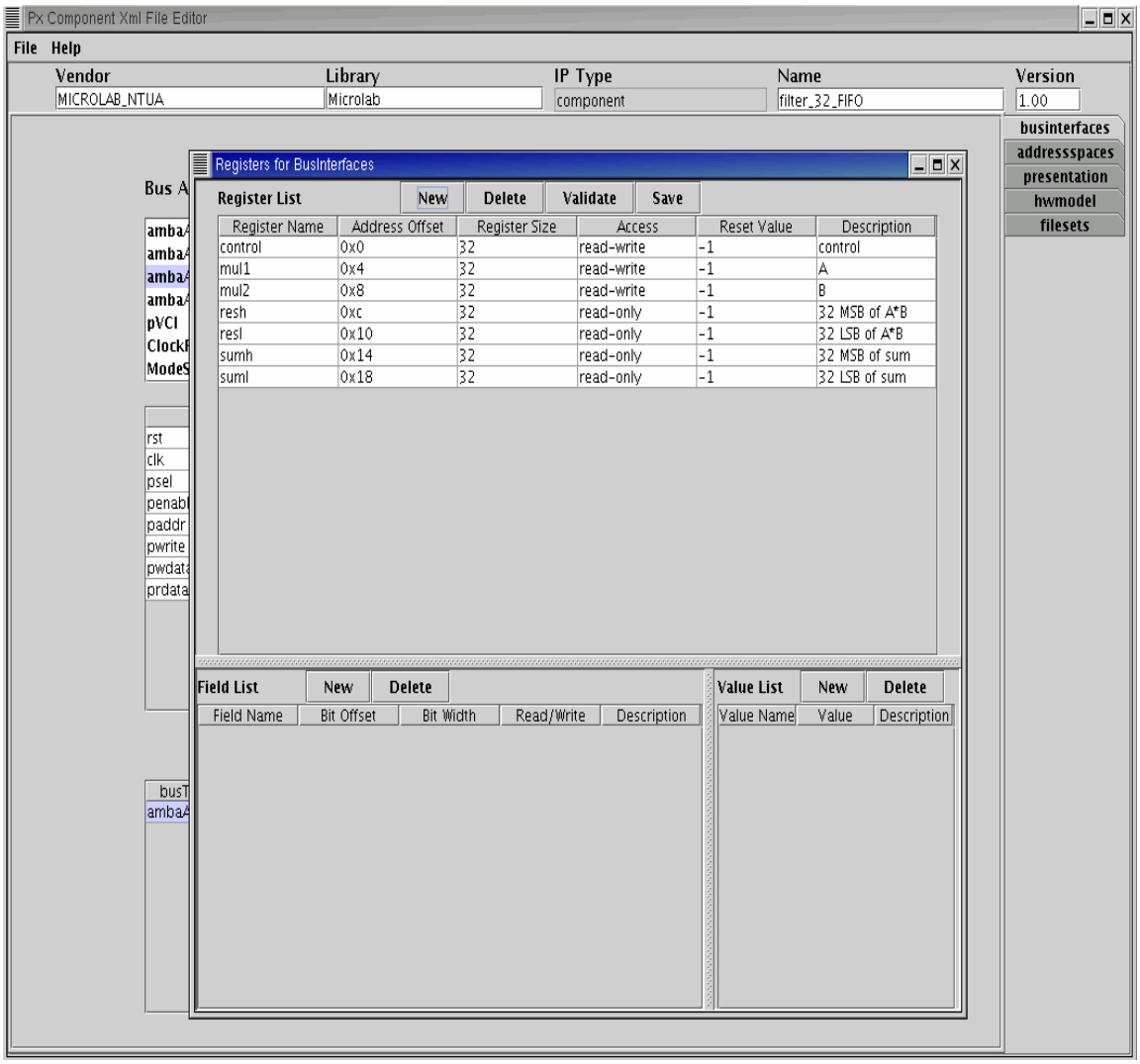
Σχήμα 6.3

Στο πεδίο **Vendor** δηλώνουμε το όνομα του κατασκευαστή, στο **Library** το όνομα της βιβλιοθήκης του Platform Express™ όπου θα συμπεριληφθεί το σύστημά μας, στο **Name** το όνομα που θα χρησιμοποιείται και στο **Version** την έκδοση.

Αμέσως μετά καλούμαστε να διαλέξουμε τον διάδρομο που θα χρησιμοποιηθεί για την επικοινωνία του επεξεργαστή με το σύστημά μας. Διαλέγουμε τον **ambaAPB**, δίνοντας το όνομα του διαδρόμου ως interfaceId και ορίζοντας το σύστημά μας ως "σκλάβο" (slave). Στη συνέχεια αντιστοιχίζουμε τα σήματα της οντότητας με τα σήματα που παρέχει το μοντέλο του διαδρόμου που επιλέχθηκε και ορίζουμε το είδος της σύνδεσης με το διάδρομο. Έχουμε δύο επιλογές, προαιρετική (**optional**) ή υποχρεωτική (**required**). Επιλέγουμε τη δεύτερη και προχωράμε στην απεικόνιση στη μνήμη (memory mapping).

Στο σημείο αυτό δίνουμε ως διεύθυνση (**baseAddress**) τη μηδενική, την οποία θα αλλάξουμε αργότερα, καθώς επιθυμούμε αυτή να παρέχεται από το χρήστη του Platform Express™, και ρυθμίζουμε τις υπόλοιπες παραμέτρους του block της μνήμης που θα χρησιμοποιεί το σύστημα μας. Έτσι δίνουμε ως **bitOffset** το 0, μέγεθος (**range**) 256 και μήκος λέξης (**width**) 32 bits. Οι τιμές αυτές είναι ενδεικτικές.

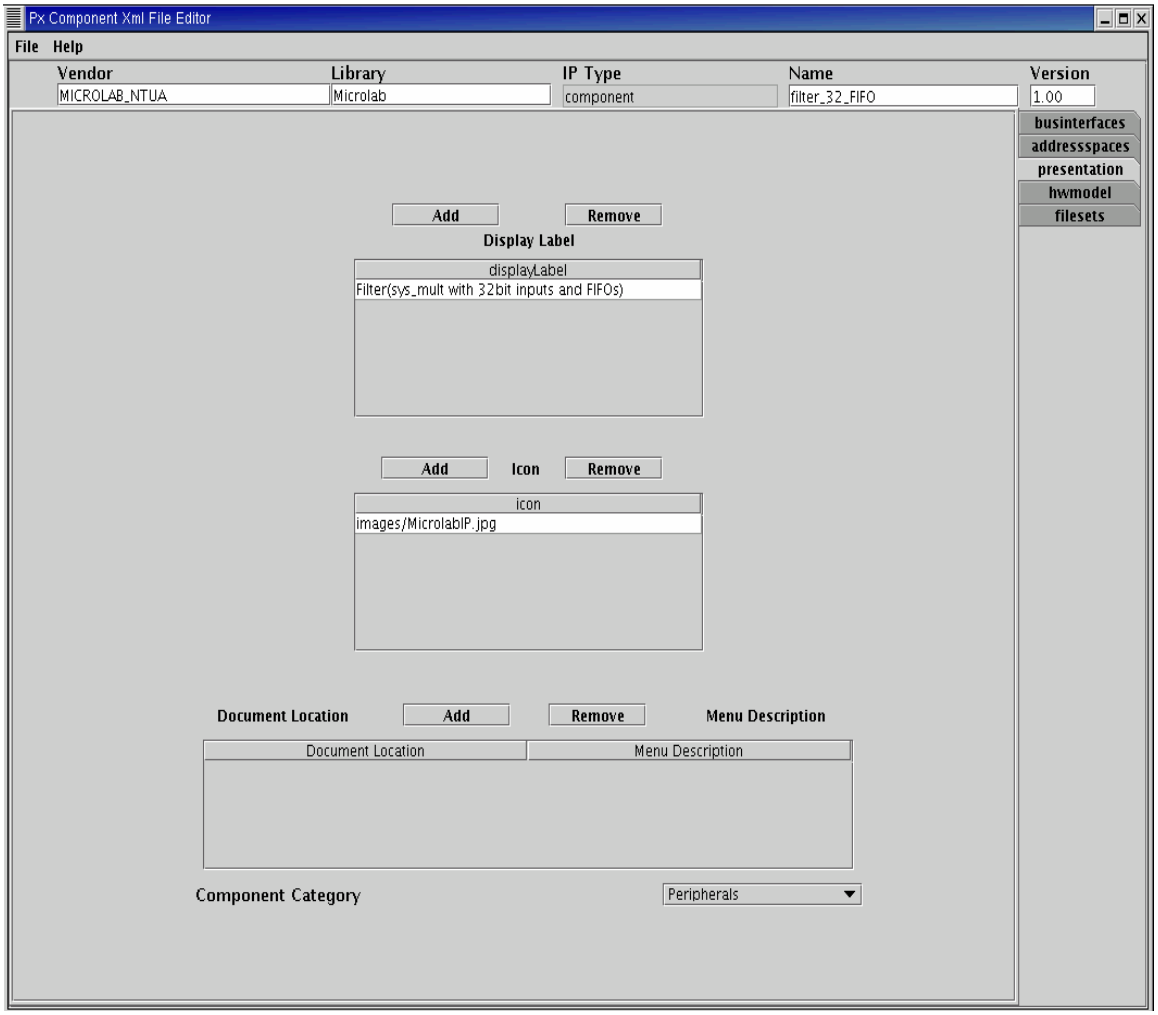
Στο block αυτό της μνήμης μπορούμε να ορίσουμε κάποιους καταχωρητές, προκειμένου να υλοποιήσουμε την επικοινωνία του επεξεργαστή με το σύστημα μας. Η διαδικασία αυτή εικονίζεται στο επόμενο σχήμα :



Σχήμα 6.4

Ορίζουμε λοιπόν τους καταχωρητές που φαίνονται στο παραπάνω σχήμα δίνοντας το κατάλληλο offset από την baseAddress του συστήματος. Τα bits 3,4,5 του offset αυτού ταυτίζονται με τις τιμές του σήματος **selectreg** που χρησιμοποιήθηκε στη σχεδίαση για την επιλογή των κατάλληλων σημάτων για εγγραφή ή ανάγνωση. Για να λειτουργεί σωστά το σύστημά μας όμως, πρέπει η baseAddress που θα δίνεται από το χρήστη του Platform Express™ να έχει τα **5 τελευταία bits μηδενικά**.

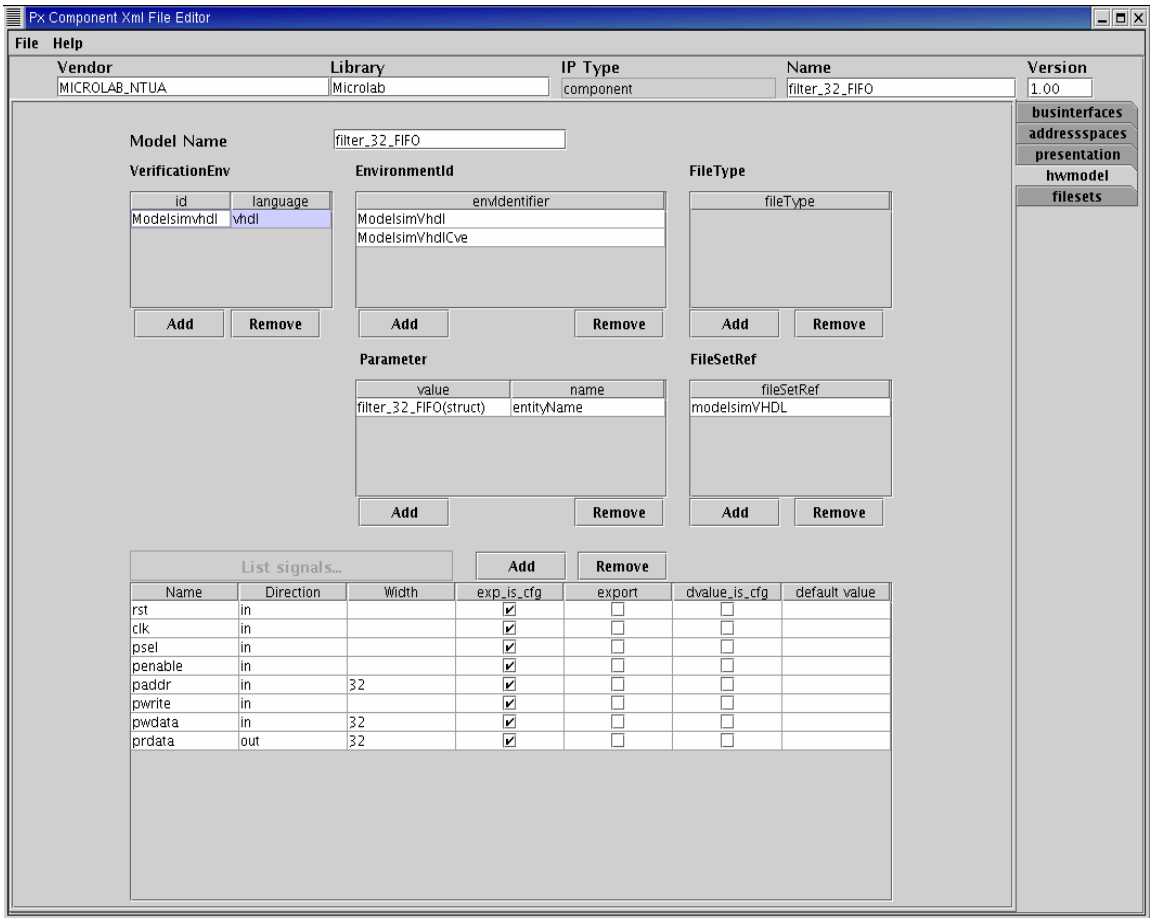
Αφού σώσουμε τους καταχωρητές προχωράμε στον ορισμό των παραμέτρων της απεικόνισης (**presentation)**.



Σχήμα 6.5

Στο πεδίο **Display Label** δίνουμε το όνομα που θα εμφανίζεται στο Platform Express™. Επίσης, μας δίνεται η δυνατότητα να συμπεριλάβουμε κάποια εικόνα, η οποία θα χρησιμοποιηθεί από το Platform Express™ για την απεικόνιση του συστήματος μας. Τέλος, ορίζουμε την κατηγορία στη οποία ανήκει (**Component Category)**, επιλέγοντας τον χαρακτηρισμό "περιφερειακό" (**peripheral)**.

Στη συνέχεια περνάμε στον ορισμό παραμέτρων που αφορούν το υλικό, οι οποίες βρίσκονται στην επιλογή **hwmodel** και εικονίζονται στο παρακάτω σχήμα :

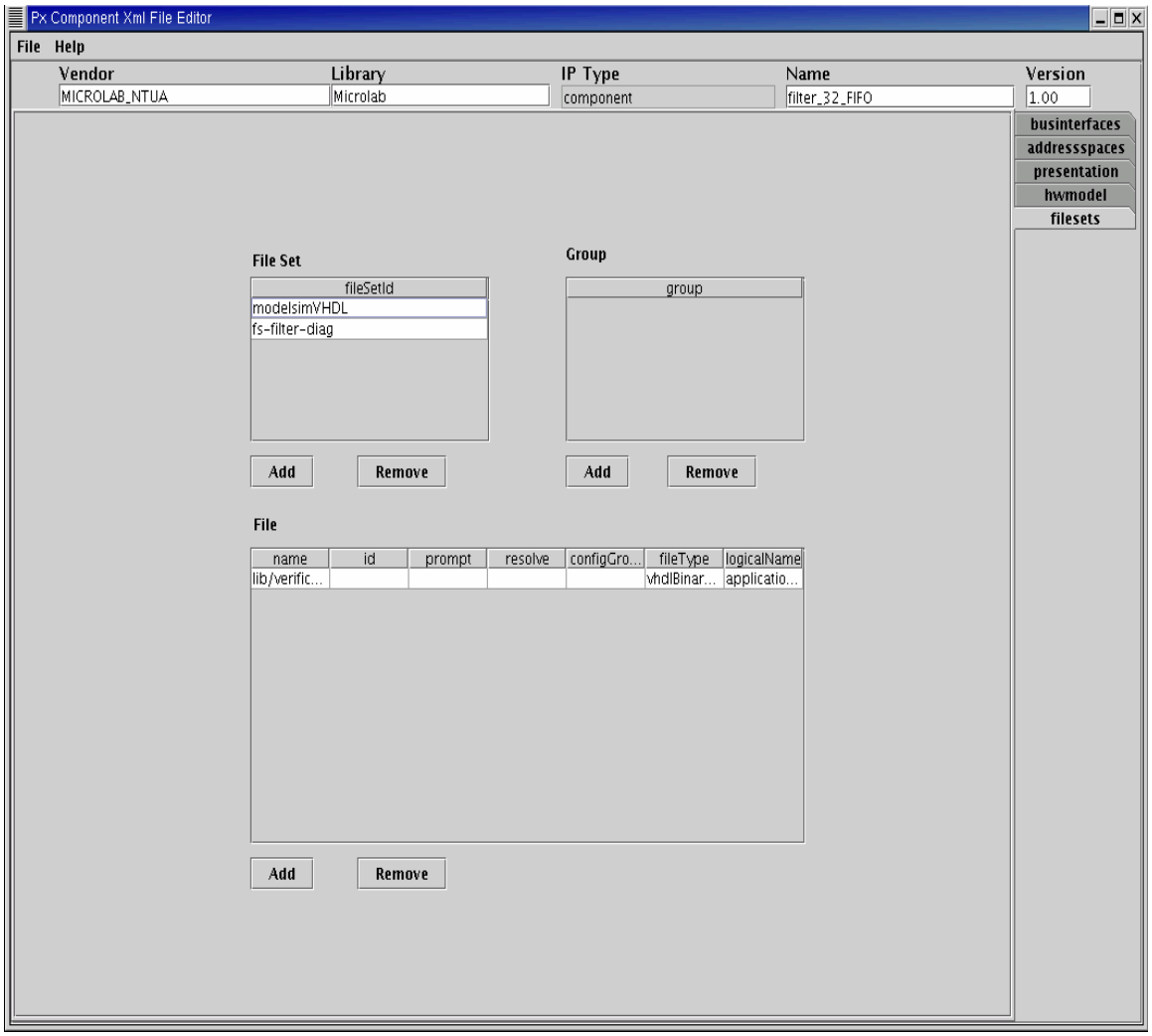


Σχήμα 6.6

Εδώ ορίζουμε το περιβάλλον της προσομοίωσης που θα χρησιμοποιηθεί από το Platform Express™ κατά τη διαδικασία της επαλήθευσης (verification) του συστήματος. Δίνουμε λοιπόν στο **id** του **VerificationEnv** το χαρακτηρισμό **Modelsimvhdl** και ορίζουμε ως γλώσσα που χρησιμοποιήθηκε για την περιγραφή του υλικού την **vhdl**. Στο πεδίο **EnvironmentId** επιλέγουμε τα **ModelsimVhdl** και **ModelsimVhdlCVE**, ενώ στο πεδίο **Parameter** δίνουμε το όνομα της οντότητας και της αρχιτεκτονικής που υπάρχουν στη σχεδίαση. Τέλος, στο πεδίο **fileSetRef** δίνουμε το όνομα του "δείκτη" που προσδιορίζει τα αρχεία τα οποία θα χρησιμοποιηθούν κατά την προσομοίωση του υλικού.

Στο κάτω μέρος του παραπάνω σχήματος διακρίνουμε τα σήματα, τα οποία θα υπάρχουν στις κυματομορφές που παράγονται κατά την προσομοίωση. Εφόσον το επιθυμούμε μπορούμε να προσθέσουμε σε αυτά και άλλα, κάτι το οποίο δεν χρειάστηκε στην περίπτωση μας.

Στη συνέχεια πρέπει να ορίσουμε τους απαραίτητους "δείκτες" που προσδιορίζουν τα διάφορα αρχεία που απαιτούνται για τη σωστή επαλήθευση του κυκλώματος. Αρχικά ορίζουμε το "δείκτη" που προσδιορίζει τη βιβλιοθήκη που απαιτείται για την προσομοίωση του υλικού. Η διαδικασία αυτή εικονίζεται στο επόμενο σχήμα :

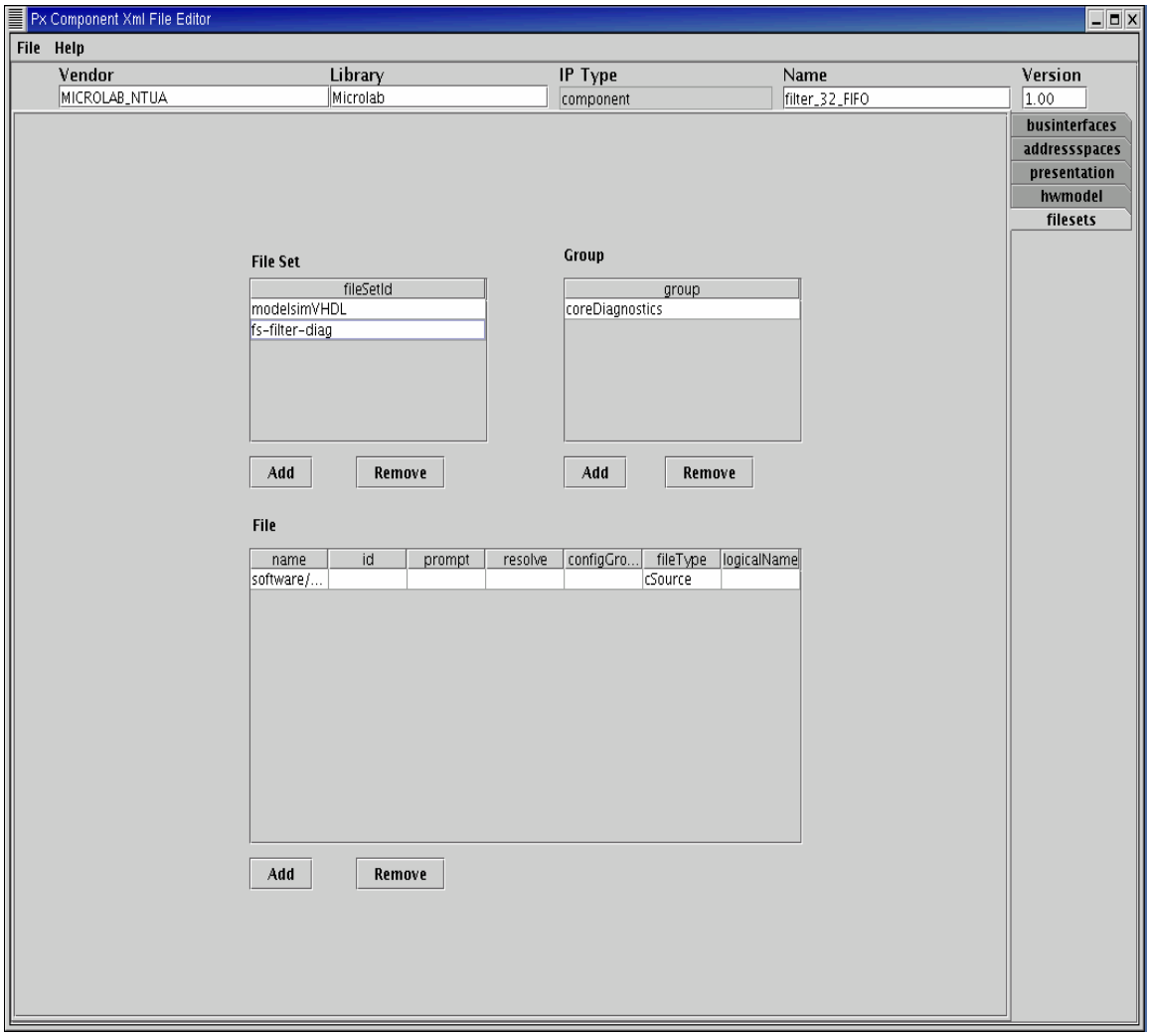


Σχήμα 6.7

Στο πεδίο **fileSetid** δίνουμε το όνομα **modelsimVHDL** που χρησιμοποιήθηκε όπως είδαμε προηγουμένως. Στο πεδίο **File** ρυθμίζουμε τις εξής παραμέτρους :

- **name** : Δίνουμε το πλήρες μονοπάτι μέχρι τη βιβλιοθήκη του υλικού, ξεκινώντας από το φάκελο **lib** που έχουμε δημιουργήσει.
- **fileType** : Επιλέγουμε τον τύπο των αρχείων. Στη συγκεκριμένη περίπτωση κάνουμε την επιλογή **vhdlBinary**.
- **logicalName** : Δίνουμε το όνομα της βιβλιοθήκης που είχαμε ορίσει κατά τη σχεδίαση. Τη βιβλιοθήκη που κατασκευάσαμε την είχαμε ονομάσει **application32** και αυτό χρησιμοποιούμε στο πεδιό αυτό.

Επίσης, μπορούμε να ορίσουμε και το αρχείο που θα χρησιμοποιηθεί κατά την προσομοίωση του λογισμικού. Η διαδικασία αυτή απεικονίζεται στο επόμενο σχήμα :



Σχήμα 6.8

Όπως φαίνεται και στην εικόνα αυτή ορίζουμε ως **fileSetid** το **fs-filter-diag**, το οποίο και εντάσσουμε στο group **coreDiagnostics**. Στο πεδίο **File** ορίζουμε τώρα τις εξής παραμέτρους :

- **name** : Δίνουμε το πλήρες μονοπάτι μέχρι το λογισμικό που θα χρησιμοποιηθεί, ξεκινώντας από το φάκελο **software** που έχουμε δημιουργήσει.
- **fileType** : Επιλέγουμε τον τύπο των αρχείων. Στη συγκεκριμένη περίπτωση κάνουμε την επιλογή **cSource**, καθώς το λογισμικό που θα χρησιμοποιήσουμε είναι ένα πρόγραμμα γραμμένο σε C.

Μετά την ολοκλήρωση των παραπάνω διαδικασιών σώζουμε σε ένα αρχείο xml και το PxEdit ελέγχει αν έχουν συμπληρωθεί με σωστό τρόπο τα διάφορα πεδία. Δεν ελέγχει αν οι ορισμοί που έχουμε δώσει είναι σωστοί, αλλά αν το xml αρχείο που δημιουργείται είναι σωστά δομημένο και συμπληρωμένο σύμφωνα με το πρότυπο που απαιτεί το Platform Express™.

6.3 Υπόλοιπες Ρυθμίσεις

Όπως έχει ήδη αναφερθεί η χρήση του PxEdit δεν επαρκεί για τη σωστή ενσωμάτωση του συστήματός μας στο Platform Express™. Είμαστε λοιπόν υποχρεωμένοι να πραγματοποιήσουμε κάποιες αλλαγές στο xml αρχείο που δημιουργήσαμε προηγουμένως.
Καταρχάς, επιθυμούμε η baseAddress που χρησιμοποιείται κατά το memory mapping του συστήματος μας να μην είναι σταθερή, αλλά να την επιλέγει κάθε φορά ο χρήστης του Platform Express™. Για να επιτευχθεί αυτό αλλάζουμε την baseAddress στο xml αρχείο ως εξής :

```
<memoryMap>
  <addressBlock>
      <baseAddress configGroups="requiredConfig" format="long" id="baseAddress" prompt="Base Address:" resolve="user"/>
      <bitOffset>0</bitOffset>
      <range>256</range>
      <width>32</width>
```

Πλέον, όταν ο χρήστης επιλέγει να χρησιμοποιήσει το σύστημα που έχουμε σχεδιάσει, το Platform Express™ του ζητάει να ορίσει την baseAddress, η οποία όμως πρέπει να τηρεί τον περιορισμό που αναφέρθηκε προηγουμένως, δηλαδή τα 5 τελευταία bits της να είναι μηδενικά.
Προκειμένου το Platform Express™ να εκτελεί το λογισμικό που έχουμε κατασκευάσει πρέπει να προσθέσουμε αρχικά την εξής γραμμή στο xml αρχείο :

```
            <fileSetRef>fs-filter-diag</fileSetRef>
        </slave>
    </busInterface>
```

Στη συνέχεια πρέπει να προσθέσουμε στον ορισμό του δείκτη **fs-filter-diag** τα παρακάτω :

```
<fileSet fileSetId="fs-filter-diag">
        <group>coreDiagnostics</group>
        <file fileId="fs-filter_32_FIFO-diag">
                <name>software/src/filter.c</name>
                <fileType>cSource</fileType>
        </file>
        <swFunction>
                <entryPoint>filterCoreDiagnostics</entryPoint>
                <fileRef>fs-filter_32_FIFO-diag</fileRef>
                <argument>
                        <name>baseAddress</name>
                        <dataType>unsigned int</dataType>
                        <value dependency="id('baseAddress')" resolve="dependent"/>
                </argument>
        </swFunction>
</fileSet>
```

Στο σημείο αυτό δίνουμε το όνομα της συνάρτησης που υλοποιεί το λογισμικό και η οποία χρησιμοποιείται για τον έλεγχο της ορθής λειτουργίας του κυκλώματος μας. Στην περίπτωση μας είναι **fiterCoreDiagnostics**.

Μετά την ολοκλήρωση των παραπάνω μετατροπών του xml αρχείου το σύστημά μας έχει πλέον ενσωματωθεί και μπορεί να χρησιμοποιηθεί από το περιβάλλον του Platform Express™ για την κατασκευή embedded συστημάτων.

7. Επαλήθευση (Verification) Συστήματος

7.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται το λογισμικό που κατασκευάστηκε για την επαλήθευση του συστήματος και η χρήση του Platform Express ™ και του Seamless CVE ™ για την επαλήθευση ενός απλού συστήματος που περιλαμβάνει και το υλικό που σχεδιάσαμε.

7.2 Λογισμικό

Προκειμένου να επαληθευτεί η σωστή επικοινωνία του επεξεργαστή με το σύστημα που σχεδιάσαμε, κρίθηκε απαραίτητη η κατασκευή ενός απλού λογισμικού, το οποίο γράφτηκε σε γλώσσα C και παρουσιάζεται αναλυτικά στη συνέχεια.

Το πρόγραμμα που κατασκευάζουμε χρησιμοποιείται από το Platform Express™ μαζί με άλλα για την υλοποίηση ενός γενικότερου εκτελέσιμου αρχείου. Δηλαδή παράγεται ένα εκτελέσιμο αρχείο, το οποίο περιέχει το κατάλληλο λογισμικό για τον έλεγχο όλων των τμημάτων του συστήματος που σχεδιάζει ο χρήστης του Platform Express™. Επομένως, εμείς καλούμαστε να υλοποιήσουμε μια συνάρτηση, η οποία θα εκτελεί τους κατάλληλους ελέγχους.
Το όνομα αυτής της συνάρτησης πρέπει να ταυτίζεται με το όνομα που δόθηκε κατά την ενσωμάτωση του υλικού στο Platform Express™ (Κεφάλαιο 6). Έτσι έχουμε τον εξής ορισμό :

```
#ifndef FUNCNAME
#define FUNCNAME filterCoreDiagnostics
#endif
```

Στη συνέχεια πρέπει να ορίσουμε μια σειρά από καταχωρητές προκειμένου να είναι δυνατή η επικοινωνία του επεξεργαστή με το memory mapped σύστημά μας. Επομένως, έχουμε :

```
#define CONTROL *(unsigned long *) (baseAddress + 0x0)
#define MUL1    *(unsigned long *) (baseAddress + 0x4)
#define MUL2    *(unsigned long *) (baseAddress + 0x8)
#define RESH    *(unsigned long *) (baseAddress + 0xC)
#define RESL    *(unsigned long *) (baseAddress + 0x10)
#define SUMH    *(unsigned long *) (baseAddress + 0x14)
#define SUML    *(unsigned long *) (baseAddress + 0x18)
```

Γίνεται εύκολα αντιληπτό ότι οι καταχωρητές αυτοί ταυτίζονται με αυτούς που ορίστηκαν κατά την ενσωμάτωση του υλικού στο Platform Express™. Όπως έχει αναφερθεί η τιμή του control αποτελεί πληροφορία ελέγχου του συστήματος. Ορίζουμε λοιπόν τις εξής τιμές :

```
#define RESET    0x0
#define SEND_MUL1 0x1
#define SEND_MUL2 0x2
#define MULT     0x3
#define READ     0x4
```

Οι τιμές αυτές ταυτίζονται όπως είναι φανερό με τις τιμές του σήματος state, το οποίο όπως είδαμε κατά τη σχεδίαση προσδιορίζει την κατάσταση λειτουργίας του κυκλώματος.
Επίσης ορίζουμε και τις παρακάτω μάσκες :

```
#define READY    0x8
#define OVERFLOW  0x10
```

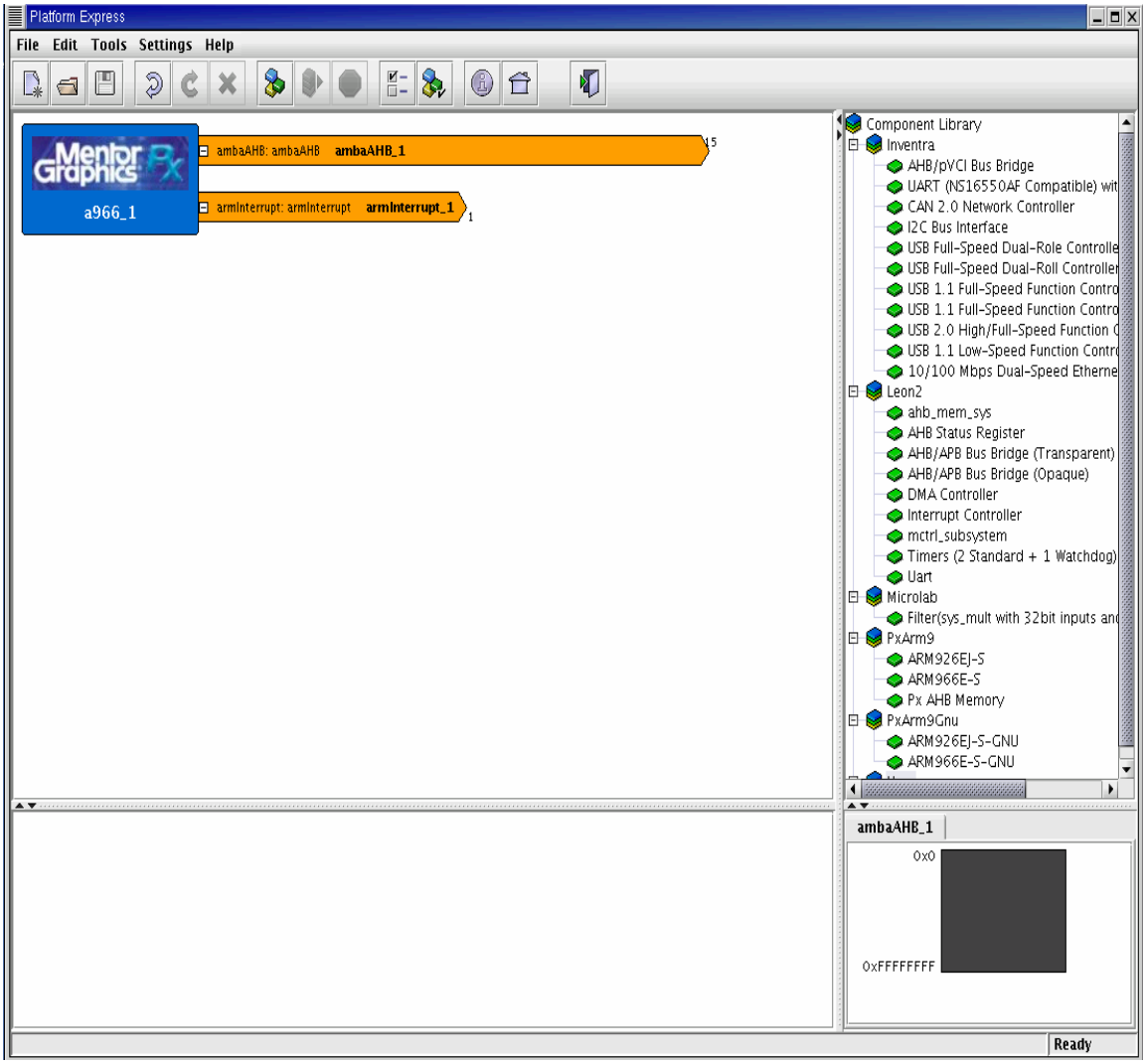
Οι μάσκες αυτές θα χρησιμοποιηθούν για τον έλεγχο του $4^{ου}$ και $5^{ου}$ bit αντίστοιχα της τιμής του control, προκειμένου να δούμε πότε έχουν παραχθεί τα αποτελέσματα και αν υπάρχει υπερχείλιση στο τελικό άθροισμα.
Το πρόγραμμά μας λοιπόν αρχικοποιεί το φίλτρο που έχουμε σχεδιάσει και στέλνει τις τιμές του πολλαπλασιαστέου A. Στη συνέχεια στέλνει τις τιμές του πολλαπλασιαστέου B και ξεκινά τον πολλαπλασιασμό. Αμέσως μετά ελέγχει συνέχεια την τιμή του control μέχρι το $4^{ο}$ του bit να γίνει 1, δηλαδή να τελειώσει η αποθήκευση των γινομένων στις ουρές εξόδου. Τότε, ο επεξεργαστής διαβάζει τα δύο τμήματα των γινομένων και του αθροίσματος και τυπώνει τις τιμές τους.
Το Platform Express περιέχει μια συνάρτηση, την **printToPort**, η οποία χρησιμοποιείται για να τυπώνει ακολουθίες χαρακτήρων (strings) σε ένα παράθυρο του debugger, που χρησιμοποιείται ως οθόνη. Εμείς επιθυμούμε να τυπώνουμε κάθε φορά αριθμούς μήκους 32 bit**,** την τιμή των οποίων πρέπει να μετατρέπουμε στην κατάλληλη ακολουθία χαρακτήρων.
Η απαιτούμενη αυτή μετατροπή γίνεται από μια άλλη συνάρτηση που κατασκευάζουμε, την **string_from_int**. Η συνάρτηση αυτή λαμβάνει ως όρισμα τον αριθμό μήκους 32 bit. Όπως είναι γνωστό οι απρόσημοι ακέραιοι της C μπορούν να πάρουν τιμές στο διάστημα 0 ως 2^{32}-1. Τα όρια αυτά δεν επαρκούν όμως για τη σωστή αναπαράσταση των 32 MSB των γινομένων και του αθροίσματος. Για το λόγο αυτό, δεν θα χρησιμοποιήσουμε το δεκαδικό σύστημα για την αναπαράσταση των αριθμών αλλά το δεκαεξαδικό. Η συνάρτηση string_from_int διαβάζει λοιπόν κάθε φορά 4 bits του αριθμού και τοποθετεί

τον αντίστοιχο δεκαεξαδικό χαρακτήρα σε ένα πίνακα χαρακτήρων. Μόλις διαβαστούν και τα 32 bits ο πίνακας τυπώνεται.

Ο πλήρης κώδικας των παραπάνω συναρτήσεων που κατασκευάσαμε παρατίθεται στο παράρτημα.

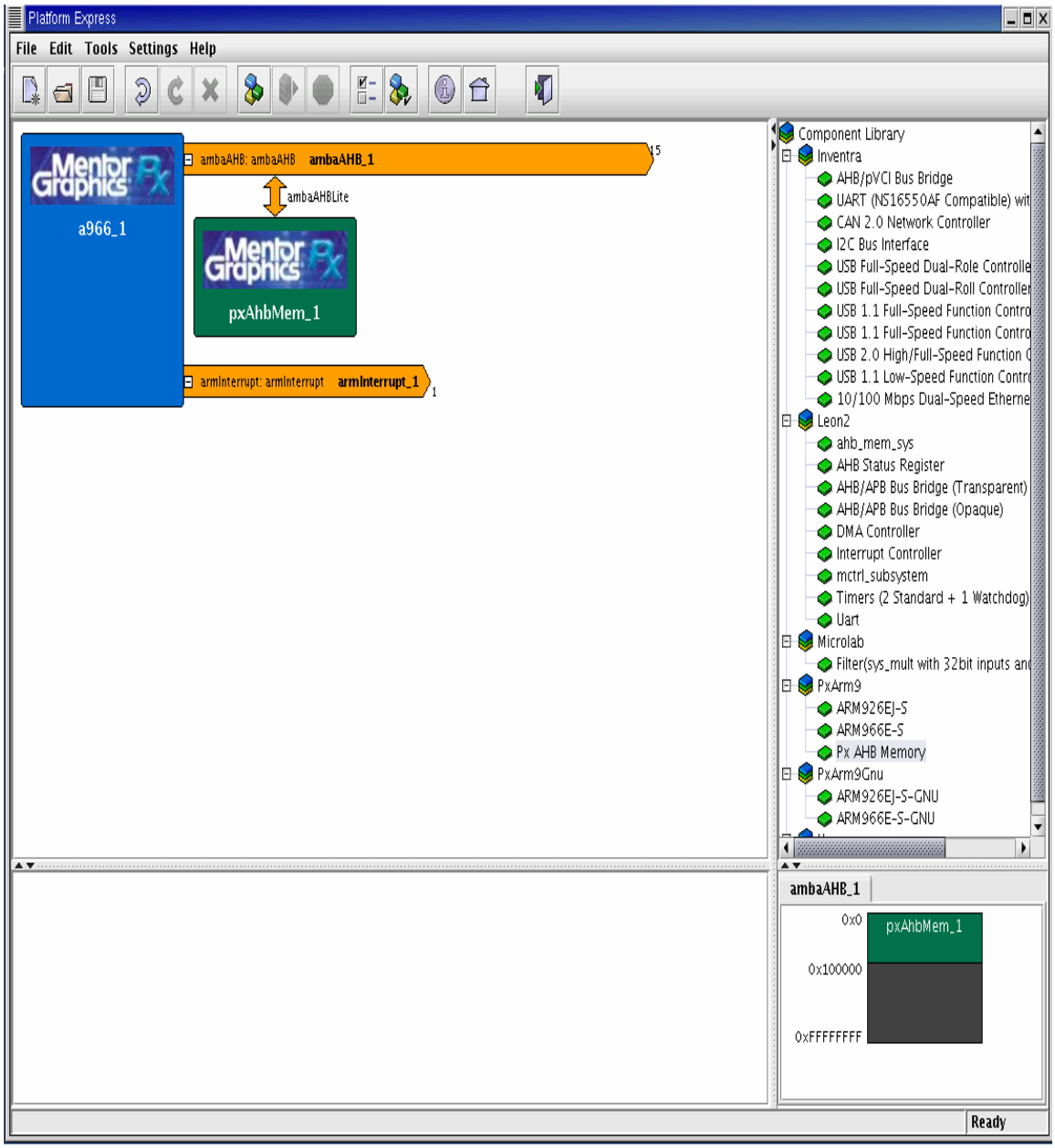
7.3 Σχεδίαση Συστήματος στο Platform Express

Θα κατασκευάσουμε ένα σύστημα που θα αποτελείται από ένα επεξεργαστή **ARM 966**, μνήμη και το φίλτρο που έχουμε σχεδιάσει. Στο περιβάλλον του Platform Express™ επιλέγουμε τον επεξεργαστή :



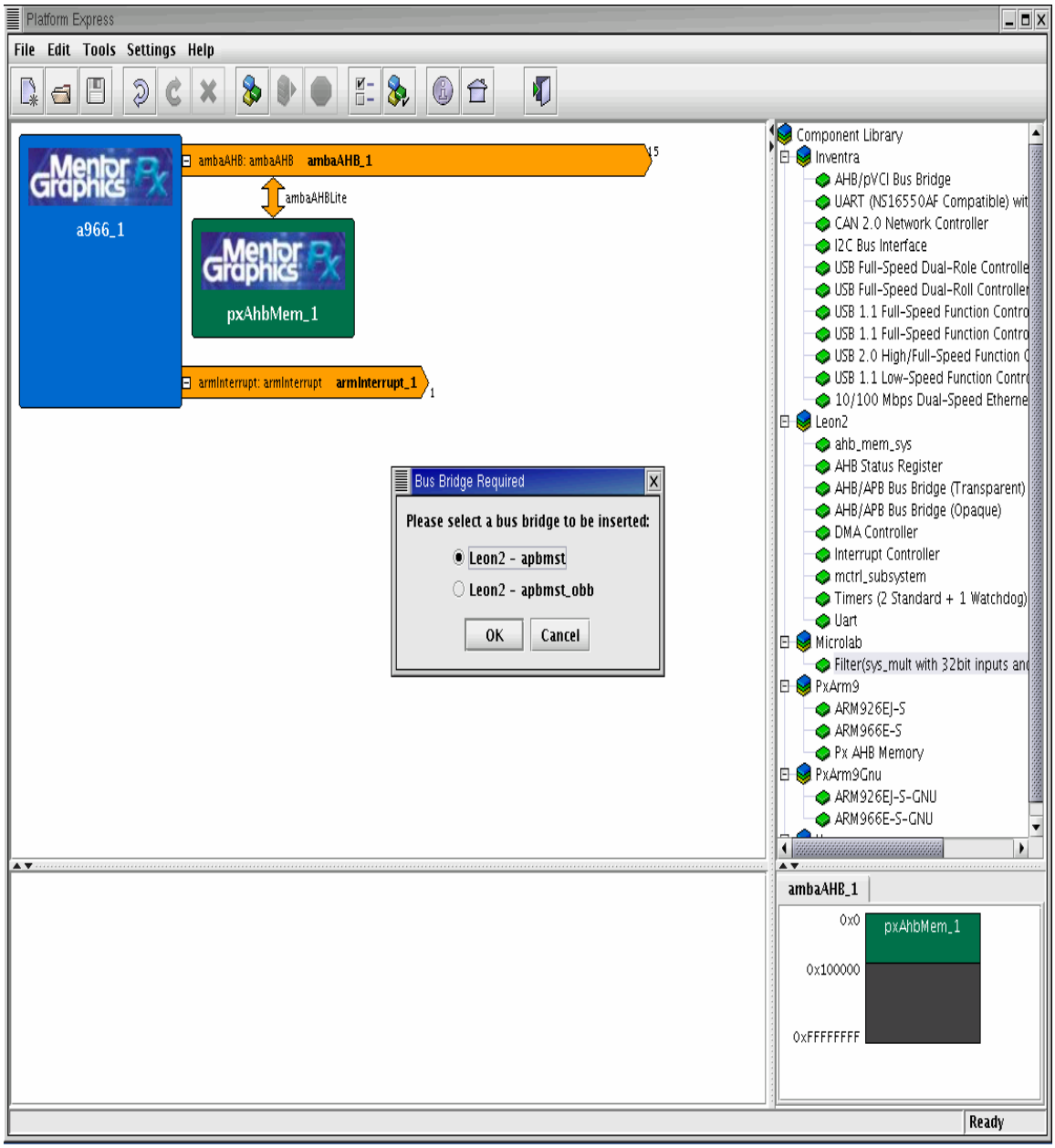
Σχήμα 7.1

Από τα components που εμφανίζονται στο δεξί μέρος της παραπάνω οθόνης επιλέγουμε τη μνήμη από τη βιβλιοθήκη **PxArm9**, της οποίας ορίζουμε το μέγεθος της ίσο με **1M**.



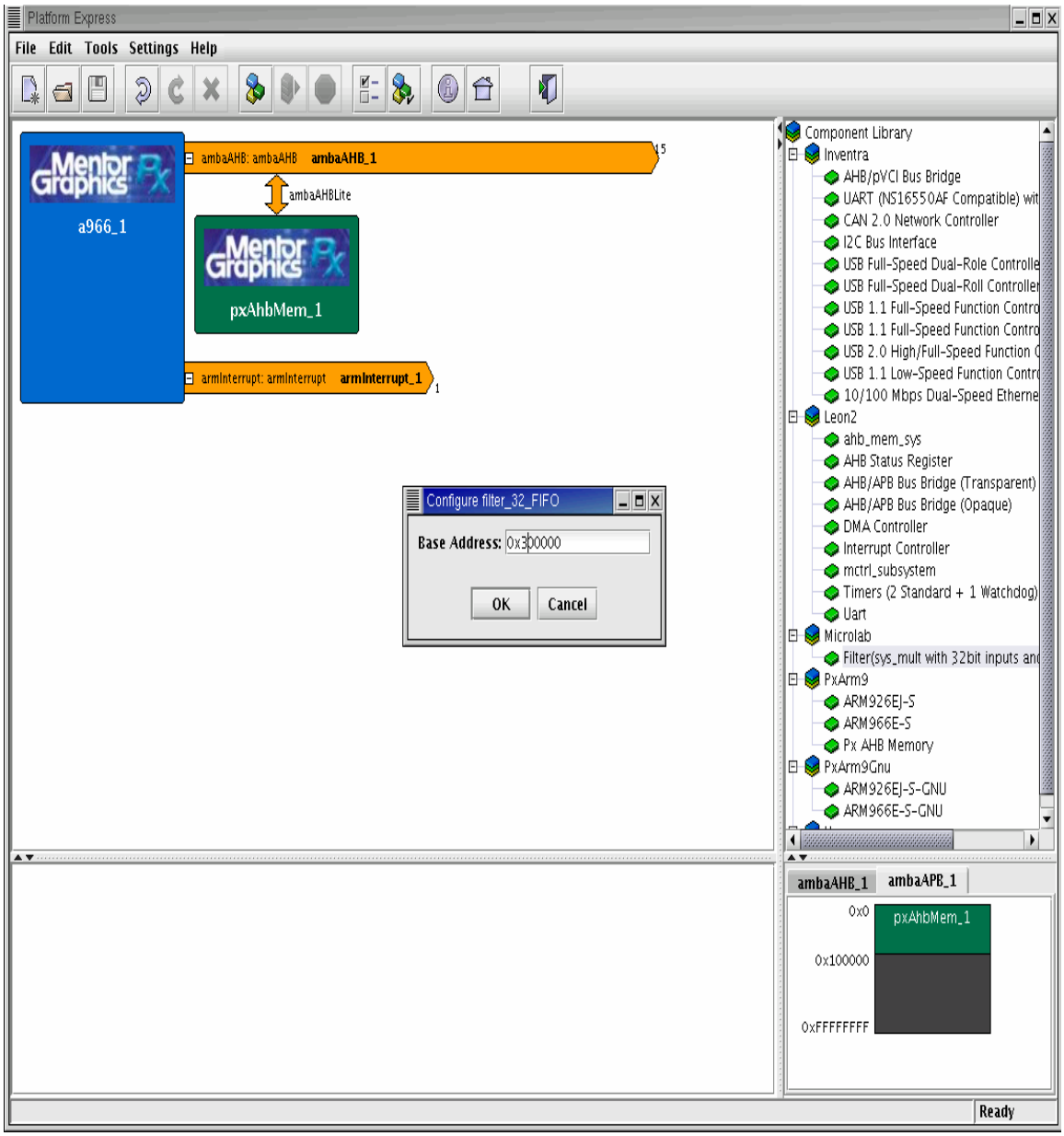
Σχήμα 7.2

Στη συνέχεια επιλέγουμε από τη βιβλιοθήκη **Microlab** το φίλτρο που έχουμε κατασκευάσει. Όπως έχει ήδη αναφερθεί, το φίλτρο αυτό έχει σχεδιαστεί για διάδρομο **APB**. Για να συνδεθεί με το διάδρομο **ambaAHB** πρέπει να χρησιμοποιήσουμε μια γέφυρα.



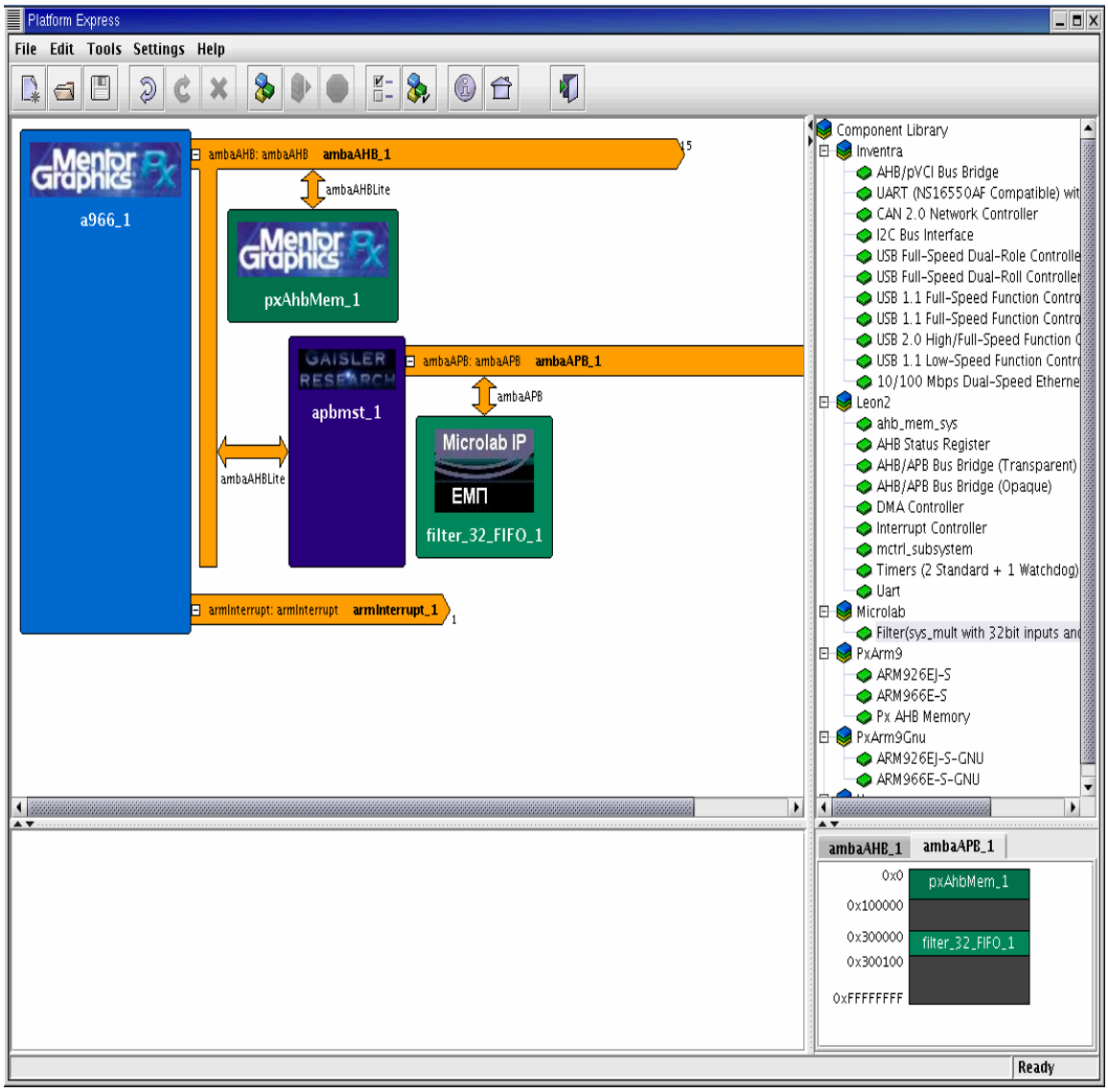
Σχήμα 7.3

Αφού επιλέξουμε τη γέφυρα που θα χρησιμοποιηθεί, πρέπει στη συνέχεια να επιλέξουμε τη διεύθυνση της μνήμης, στην οποία θα "απεικονίζεται" το φίλτρο. Επιλέγουμε τη διεύθυνση **0x300000**.



Σχήμα 7.4

Το τελικό σύστημα εικονίζεται στο παρακάτω σχήμα :



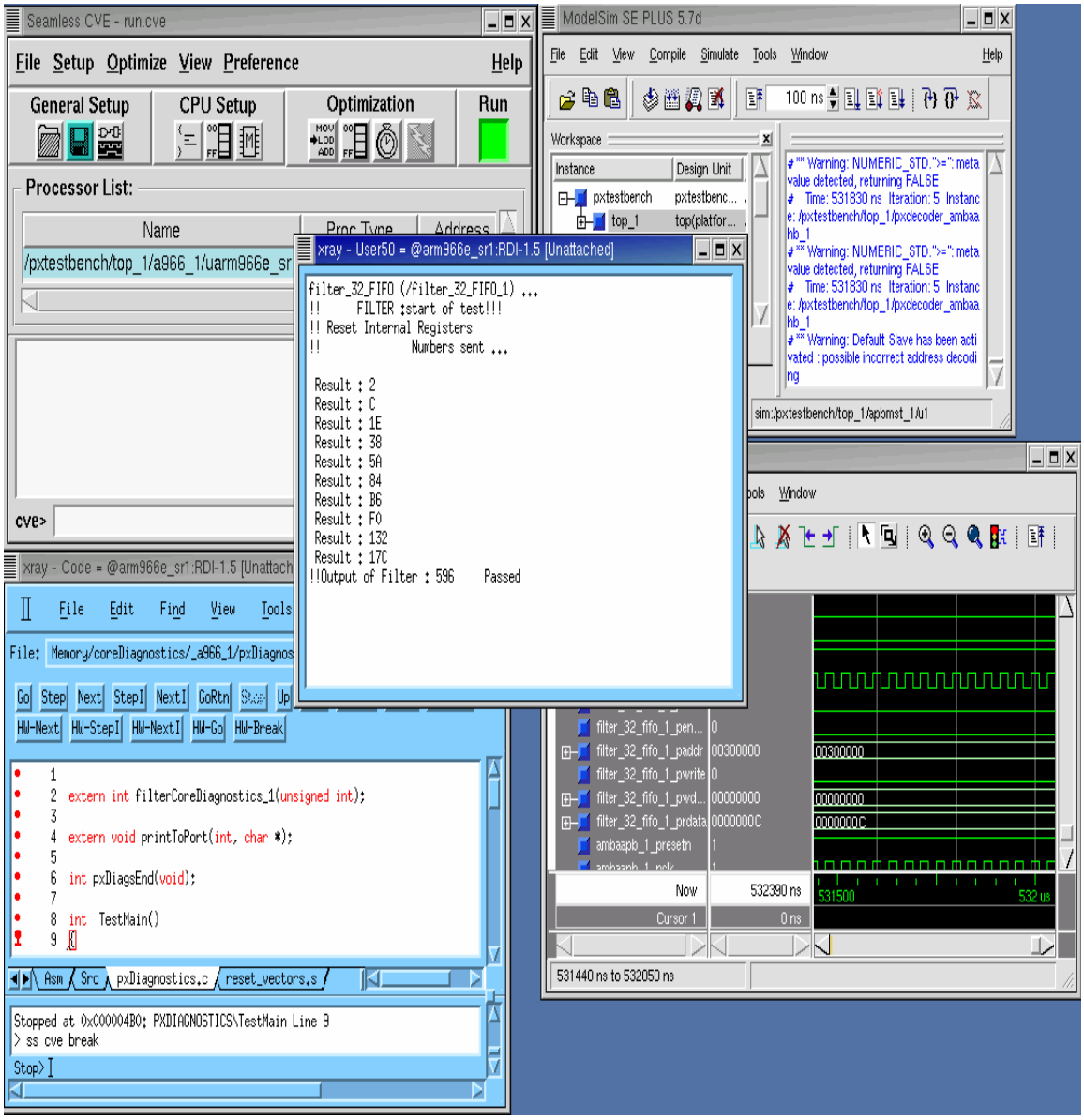
Σχήμα 7.5

Στο κάτω δεξί παράθυρο της οθόνης βλέπουμε τη δομή του διαδρόμου, όπου στη διεύθυνση **0x0** έχει τοποθετηθεί η μνήμη του συστήματος και στη διεύθυνση **0x300000** το φίλτρο που έχουμε σχεδιάσει.

7.4 Επαλήθευση (Verification) Συστήματος

Αφού κάνουμε build το παραπάνω σχέδιο εκτελούμε την προσομοίωση. Το λογισμικό που χρησιμοποιούμε πραγματοποιεί τον πολλαπλασιασμό των αριθμών 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 με τους αριθμούς 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 αντίστοιχα. Οι ουρές και ο αθροιστής έχουν ρυθμιστεί με βάθος **10**. Θα μπορούσαμε βέβαια να εισάγουμε λιγότερα από 10 ζευγάρια πολλαπλασιαστέων, αφού κατά την αρχικοποίηση του συστήματος μηδενίζονται τα περιεχόμενα των ουρών και του αθροιστή. Έτσι, το τελικό άθροισμα θα υπολογιζόταν σωστά, απλά κατά το διάβασμα των τελικών γινομένων θα διαβάζαμε και κάποια μηδενικά.

Τα αποτελέσματα της προσομοίωσης παρουσιάζονται στο επόμενο σχήμα :



Σχήμα 7.6 Αποτελέσματα Προσομοίωσης

Τα αποτελέσματα είναι τα αναμενόμενα. Δηλαδή το σύστημα μας λειτουργεί σωστά !!!

8. Συμπεράσματα – Μελλοντικές Προεκτάσεις

8.1 Ανακεφαλαίωση – Συμπεράσματα

Στην παρούσα διπλωματική εργασία πραγματοποιήθηκε μια πρώτη γνωριμία με τη μεθοδολογία σχεδίασης embedded συστημάτων **"Hardware – Software Codesign"**. Η μεθοδολογία αυτή στηρίζεται σε πολύ μεγάλο βαθμό σε διάφορα εργαλεία, τα οποία εκτελούν τις προσομοιώσεις του υλικού και του λογισμικού και βοηθούν το σχεδιαστή να λάβει τις κατάλληλες αποφάσεις, ώστε να σχεδιαστεί το βέλτιστο σύστημα που θα υλοποιεί τις επιθυμητές εφαρμογές.
Η εκμάθηση λοιπόν των εργαλείων αυτών αποτελεί το πρώτο σημαντικό βήμα. Για το λόγο αυτό επιλέχθηκαν εργαλεία, τα οποία απολαμβάνουν την εκτίμηση του μεγαλύτερου μέρους της βιομηχανίας και θεωρούνται από τα κορυφαία στο είδος τους. Στη συνέχεια σχεδιάστηκε και υλοποιήθηκε με τα εργαλεία αυτά μια σχετικά απλή εφαρμογή, η οποία μας έδωσε τη δυνατότητα να εξερευνήσουμε τις δυνατότητες των εργαλείων. Το αποτέλεσμα ήταν η σχεδίαση ενός πολλαπλασιαστή – φίλτρου, ο οποίος δέχεται δεδομένα μήκους 32 bits μέσω ενός διαδρόμου από έναν επεξεργαστή ARM και επιστρέφει τα αποτελέσματα μήκους 64 bits, ενώ ο έλεγχος του πολλαπλασιαστή γίνεται από το χρήστη γράφοντας ένα απλό πρόγραμμα σε γλώσσα C.
Κατά τη διάρκεια της σχεδίασης έγινε φανερή η μεγάλη βοήθεια που προσφέρουν τα εργαλεία αυτά, καθώς από την αρχή μας παρέχουν τη δυνατότητα να εκτελούμε το επιθυμητό λογισμικό στο επιθυμητό υλικό και έτσι να ανακαλύπτουμε τα λάθη που υπήρχαν στη σχεδίαση. Κάτι, που χωρίς τα εργαλεία αυτά, θα ήταν δυνατό μόνο μετά την κατασκευή του υλικού, οπότε και θα ήταν αργά για να αντιμετωπιστούν κάποια προβλήματα.

8.2 Μελλοντικές Προεκτάσεις

Όπως αναφέρθηκε προηγουμένως η εκμάθηση των εργαλείων αποτελεί το πρώτο βήμα. Η εμπειρία που αποκτήθηκε λοιπόν από την παρούσα διπλωματική εργασία μπορεί στη συνέχεια να αξιοποιηθεί με διάφορους τρόπους.
Καταρχάς, το επόμενο βήμα πρέπει ίσως να είναι η σύνθεση και κατασκευή της συγκεκριμένης εφαρμογής. Με αυτό τον τρόπο θα μας δοθεί η δυνατότητα να παρατηρήσουμε τις διαφορές μεταξύ της πραγματικής συμπεριφοράς του συστήματος και της συμπεριφοράς του σύμφωνα με τις προσομοιώσεις, αξιολογώντας έτσι την αξία των εργαλείων αυτών.
Επίσης, στη συγκεκριμένη περίπτωση χρησιμοποιήθηκε ένας συστολικός ripple carry πολλαπλασιαστής. Μπορεί λοιπόν να χρησιμοποιηθούν διαφορετικοί πολλαπλασιαστές και με τη βοήθεια των προσομοιώσεων να πραγματοποιηθούν μετρήσεις της επίδοσης

του συστήματος, προκειμένου να βρεθεί η πιο αποδοτική υλοποίηση. Στο πνεύμα αυτό μπορεί να σχεδιαστεί και να μελετηθεί ένας πολύ μεγάλος αριθμός συστημάτων, αξιοποιώντας τις βιβλιοθήκες έτοιμων τμημάτων υλικού που προσφέρουν τα εργαλεία και σχεδιάζοντας βέβαια τα δικά μας κυκλώματα.

Τα εργαλεία αυτά δίνουν επίσης τη δυνατότητα προσομοίωσης τμημάτων υλικού που έχουν περιγραφεί σε γλώσσα που στηρίζεται στη γλώσσα προγραμματισμού C, τα οποία ονομάζονται C-Bridge Models. Θα έπρεπε ίσως λοιπόν να αξιοποιηθεί αυτή η δυνατότητα και να σχεδιαστούν κυκλώματα με τον τρόπο αυτό.

Τέλος, τα εργαλεία που χρησιμοποιήθηκαν συνεισφέρουν μεν σημαντικά στη διαδικασία της σχεδίασης, αλλά από την άλλη μεριά είναι σχετικά καινούρια και επιδέχονται βελτιώσεις. Σε παγκόσμιο επίπεδο, διάφορα ιδρύματα και εταιρίες προσπαθούν να παρουσιάσουν βελτιωμένα εργαλεία. Θα μπορούσαν λοιπόν να υπάρξουν ερευνητικές προσπάθειες και προς αυτή την κατεύθυνση.

ΠΑΡΑΡΤΗΜΑ

Α. Κώδικας Περιγραφής Υλικού

Α1. Φίλτρο

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library application32;
use application32.amba.all;

entity filter_32_FIFO is
  port (
    rst     : in  std_logic;
    clk     : in  std_logic;
    psel    : in  Std_ULogic;                       -- slave select
    penable : in  Std_ULogic;                       -- strobe
    paddr   : in  Std_Logic_Vector(31 downto 0);    -- address bus
    pwrite  : in  Std_ULogic;                       -- write
    pwdata  : in  Std_Logic_Vector(31 downto 0);    -- write data bus
    prdata  : out Std_Logic_Vector(31 downto 0)     -- read data bus
);
end filter_32_FIFO;

architecture struct of filter_32_FIFO is

component mul_u_cp_systolic
generic (
             NA : positive :=16;
             NB : positive :=16
);
port (
       reset : in  std_logic;
       clock : in  std_logic;
       a     : in  std_logic_vector(NA-1 downto 0);
       b     : in  std_logic_vector(NB-1 downto 0);
       ready : out std_logic;
       p     : out std_logic_vector(NA+NB-1 downto 0)
);
end component;


component FIFO
generic (    LENGTH : positive :=32;
             DEPTH  : positive :=10
);
port (
       reset  : in  std_logic;
       clock  : in  std_logic;
       input  : in  std_logic_vector(LENGTH-1 downto 0);
```

```
        output : out std_logic_vector(LENGTH-1 downto 0);
        en_in  : in  std_logic;
        en_out : in  std_logic;
        empty  : out std_logic;
        full   : out std_logic
);
end component;

component decoder
port (
        clk           : in    std_logic;
        selectreg     : in    std_logic_vector(2 downto 0);   -- Decode register from paddr
        state         : in    std_logic_vector(2 downto 0);    -- Decode state from control
        en_1          : in    std_logic;                       -- psel
        en_2          : in    std_logic;                       -- penable
        en_3          : in    std_logic;                       -- pwrite
        datain        : in    std_logic_vector(31 downto 0);   -- pwdata
        dataout       : out   std_logic_vector(31 downto 0);   -- prdata
        rst_fifo      : out   std_logic;
        en_Fa_in      : out   std_logic;
        en_Fb_in      : out   std_logic;
        en_Fab_out    : out   std_logic;                       -- send_mul
        en_Fab_clk    : out   std_logic;                 -- Enable clock for input FIFOs
        en_mul        : out   std_logic;                 -- Enable multiplier
        Fres_rd_in    : in    std_logic;                 -- Multiplier has produced  result
        en_Fres_in    : out   std_logic;                 -- Enable input for output FIFOs
        en_Fresh_out  : out   std_logic;                 -- send_resh
        en_Fresl_out  : out   std_logic;                 -- send_resl
        en_Fres_clk   : out   std_logic;                 -- Enable clock for output FIFO
        reg_a         : inout std_logic_vector(31 downto 0);   -- mul1
        reg_b         : inout std_logic_vector(31 downto 0);   -- mul2
        control       : inout std_logic_vector(31 downto 0);
        reg_resulth   : in std_logic_vector(31 downto 0);      -- result 32 MSB
        reg_resultl   : in std_logic_vector(31 downto 0);      -- result 32 LSB
        sig_final     : in std_logic;  -- Signal to show that processor can read from the output FIFOs
        sumh          : in std_logic_vector(31 downto 0);      -- Sum 32 MSB
        suml          : in std_logic_vector(31 downto 0);      -- Sum 32 LSB
        overflow      : in std_logic                           -- Indicates overflow of sum
);
end component;

component ADDER
generic (
             WIDTH : positive :=32;
             DEPTH : positive :=10

);
port (
        reset  : in  std_logic;
        input  : in  std_logic_vector(WIDTH-1 downto 0);
        output : out std_logic_vector(WIDTH-1 downto 0);
        overf  : out std_logic;
        enable : in  std_logic
```

```
);
end component;

-- Signals for the FIFOs

signal rst_fifo          : std_logic;            -- Reset for the FIFOs and the adder
signal mul1,mul2         : std_logic_vector(31 downto 0); -- Inputs for the input FIFOs
signal out_veca,out_vecb: std_logic_vector(31 downto 0); -- Outputs from the inputs FIFOs
                                                         -- and inputs to the multiplier
signal en_FIFO_in_a,en_FIFO_in_b  : std_logic;  -- Enable signals for input in the input FIFOs
signal send_mul          : std_logic;     -- Enable signals for output from the input FIFOs
signal en_FIFO_in_clk    : std_logic;     -- Enable clock for input into the input FIFOs
signal fifo_ab_clock     : std_logic;     -- Clock for the input FIFOs
signal en_FIFO_res       : std_logic;     -- Enable input into the output FIFOs
signal send_resh,send_resl : std_logic;  -- Enable signals for output from the output FIFOs
signal en_FIFO_out_clk   : std_logic;     -- Enable clock for input into the output FIFOs
signal fifo_res_clock    : std_logic;     -- Clock for the output FIFOs
signal resh,resl         : std_logic_vector(31 downto 0);-- Outputs from the outputt FIFOs
signal full_a,full_b,full_resh,full_resl: std_logic; - Signals to indicate that the FIFOs are
                                                         --full
signal empty_a,empty_b,empty_resh,empty_resl: std_logic; -- Signals to indicate that the
                                                         --FIFOs are empty
signal read              : std_logic;     -- Indicate that the processor can read the results

--Signals for the multiplier

signal out_vec       : std_logic_vector(63 downto 0);  -- Output of the multiplier
signal en_mul        : std_logic;                      -- Enable signal for the multiplier
signal mul_clock     : std_logic;                      -- Clock for the multiplier
signal en_FIFO_out   : std_logic;  -- Signal that indicates that the multiplier - has produced a result
signal nrst          : std_logic;                      -- Reset for the multiplier

--Signals for the adder

signal result: std_logic_vector(63 downto 0);  -- Input to the adder
signal sum   : std_logic_vector(63 downto 0);  -- Output of the adder
signal overf : std_logic;                      -- Indicates if there is an overflow from the adder
signal en_add: std_logic;                      -- Enable adder

-- Other Signals

signal control: std_logic_vector(31 downto 0); -- Control signal


FOR ALL : mul_u_cp_systolic      USE ENTITY application32.mul_u_cp_systolic;
FOR ALL : FIFO                   USE ENTITY application32.FIFO;
FOR ALL : decoder                USE ENTITY application32.decoder;
FOR ALL : ADDER                  USE ENTITY application32.ADDER;

begin

read <= full_resh and full_resl;  --Processor can read when the output FIFOs are full
```

```
DEC: decoder
       port map (
                     clk           => clk,
                     selectreg     => paddr(4 downto 2),
                     state         => control(2 downto 0),
                     en_1          => psel,
                     en_2          => penable,
                     en_3          => pwrite,
                     datain        => pwdata,
                     dataout       => prdata,
                     rst_fifo      => rst_fifo,
                     en_Fa_in      => en_FIFO_in_a,
                     en_Fb_in      => en_FIFO_in_b,
                     en_Fab_out    => send_mul,
                     en_Fab_clk    => en_FIFO_in_clk,
                     en_mul        => en_mul,
                     Fres_rd_in    => en_FIFO_out,
                     en_Fres_in    => en_FIFO_res,
                     en_Fresh_out  => send_resh,
                     en_Fresl_out  => send_resl,
                     en_Fres_clk   => en_FIFO_out_clk,
                     reg_a         => mul1,
                     reg_b         => mul2,
                     control       => control,
                     reg_resulth   => resh,
                     reg_resultl   => resl,
                     sig_final     => read,
                     sumh          => sum(63 downto 32),
                     suml          => sum(31 downto 0),
                     overflow      => overf
);

-- Input FIFOs are working when processor sends a or b and the decoder allows them to store them
-- or when they send the numbers to the multiplier and they are not empty

fifo_ab_clock <= not clk and (((en_FIFO_in_a or en_FIFO_in_b)
                 and en_FIFO_in_clk) or (send_mul and not (empty_a
                 and empty_b)));

FIFOA: FIFO
       generic map (
                     LENGTH => 32,
                     DEPTH  => 10
       )
       port map (
                     reset  => rst_fifo,
                     clock  => fifo_ab_clock,
                     input  => mul1,
                     output => out_veca,
                     en_in  => en_FIFO_in_a,
                     en_out => send_mul,
                     empty  => empty_a,
                     full   => full_a
);
```

```
FIFOB: FIFO
       generic map (
                     LENGTH => 32,
                     DEPTH  => 10
       )
       port map (
                     reset  => rst_fifo,
                     clock  => fifo_ab_clock,
                     input  => mul2,
                     output => out_vecb,
                     en_in  => en_FIFO_in_b,
                     en_out => send_mul,
                     empty  => empty_b,
                     full   => full_b
);

-- All the components reset when reset is low except for multiplier, which resets when reset is high

nrst          <= not rst_fifo ;
mul_clock     <= not clk and en_mul;

mul : mul_u_cp_systolic
       generic map (
                     NA => 32,
                     NB => 32
       )
       port map (
                     reset => nrst,
                     clock => mul_clock,
                     a     => out_veca,
                     b     => out_vecb,
                     ready => en_FIFO_out,
                     p     => out_vec
);

-- Output FIFOs are working when multiplier sends results and they are not full
-- or when the processor reads the results and the decoder allows them to export them

fifo_res_clock <= not clk and ((en_FIFO_res and not (full_resh or full_resl))
                   or ((send_resh or send_resl) and en_FIFO_out_clk));

FIFORH: FIFO
       generic map (
                     LENGTH => 32,
                     DEPTH  => 10
       )
       port map (
                     reset => rst_fifo,
                     clock => fifo_res_clock,
                     input => out_vec(63 downto 32),
                     output=> resh,
                     en_in => en_FIFO_res,
                     en_out=> send_resh,
                     empty => empty_resh,
                     full  => full_resh
```

```
);
 FIFORL: FIFO
       generic map (
                     LENGTH => 32,
                     DEPTH  => 10
       )
       port map (
                     reset => rst_fifo,
                     clock => fifo_res_clock,
                     input => out_vec(31 downto 0),
                     output=> resl,
                     en_in => en_FIFO_res,
                     en_out=> send_resl,
                     empty => empty_resl,
                     full  => full_resl
 );


-- Adder is working when te results are stored into the output FIFOs

en_add <= not clk and (en_FIFO_res and not (full_resh or full_resl));

ADD :  ADDER
       generic map (
                     WIDTH => 64,
                     DEPTH => 10
       )
       port map (
                     reset => rst_fifo,
                     input => out_vec,
                     output=> sum,
                     overf => overf,
                     enable=> en_add
);

end struct;
```

Α2. Αποκωδικοποιητής

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
port (
	clk		: in	std_logic;
	selectreg	: in	std_logic_vector(2 downto 0); -- Decode register from- paddr
	state		: in	std_logic_vector(2 downto 0); -- Decode state from control
	en_1		: in	std_logic;			-- psel
	en_2		: in	std_logic;			-- penable
	en_3		: in	std_logic;			-- pwrite
	datain		: in	std_logic_vector(31 downto 0);-- pwdata
	dataout		: out	std_logic_vector(31 downto 0);-- prdata
	rst_fifo	: out	std_logic;
	en_Fa_in	: out	std_logic;		-- en_FIFO_in_a
	en_Fb_in	: out	std_logic;		-- en_FIFO_in_b
	en_Fab_out	: out	std_logic;		-- send_mul
	en_Fab_clk	: out	std_logic;		-- Enable clock for input FIFOs
	en_mul		: out	std_logic;		-- Begin multiplier
	Fres_rd_in	: in	std_logic;		-- The multiplier has produced a result
	en_Fres_in	: out	std_logic;		-- Enable input for output FIFOs
	en_Fresh_out	: out	std_logic;		-- send_resh
	en_Fresl_out	: out	std_logic;		-- send_resl
	en_Fres_clk	: out	std_logic;		-- Enable clock for output FIFO
	reg_a		: inout	std_logic_vector(31 downto 0);	-- mul1
	reg_b		: inout	std_logic_vector(31 downto 0);	-- mul2
	control		: inout	std_logic_vector(31 downto 0);
	reg_resulth	: in	std_logic_vector(31 downto 0);	-- result 32 MSB
	reg_resultl	: in	std_logic_vector(31 downto 0);	-- result 32 LSB
	sig_final	: in	std_logic; -- Signal to show that processor can read from the output
					--FIFOs
	sumh		: in	std_logic_vector(31 downto 0);	-- Sum 32MSB
	suml		: in	std_logic_vector(31 downto 0);	-- Sum 32LSB
	overflow	: in	std_logic	-- Signal to indicate an overflow for the sum
);
end decoder;



architecture rtl of decoder is

begin

process(clk)
variable startm,starti: integer;
begin
	if clk='0' then

	--Write transactions
		if en_1='1' and en_2='1' and en_3='1' then
```

```
            if    selectreg="001" then reg_a  <= datain;
            elsif selectreg="010" then reg_b  <= datain;
            elsif selectreg="000" then control<= datain;
            end if;
      end if;

-- Read transactions

      if en_1='1' and en_2='1' and en_3='0' then
            if    selectreg="000" then dataout <= control;
            elsif selectreg="001" then dataout <= reg_a;
            elsif selectreg="010" then dataout <= reg_b;
            elsif selectreg="011" then dataout <= reg_resulth;
            elsif selectreg="100" then dataout <= reg_resultl;
            elsif selectreg="101" then dataout <= sumh;
            elsif selectreg="110" then dataout <= suml;
            end if;
      end if;

en_Fab_clk  <= en_1 and en_2 and en_3;          -- When to write to the input FIFOs
en_Fres_clk <= en_1 and not en_2 and not en_3; -- When to read from the output FIFOs

-- Decode the value of control

      if (state="000") then              -- CONTROL = 0 => RESET
            rst_fifo   <='0';
            en_Fa_in   <='0';
            en_Fb_in   <='0';
            en_Fab_out <='0';
            en_mul     <='0';
            en_Fresh_out<='0';
            en_Fresl_out<='0';
            startm := 0;          -- Flag to show when to start to multiply
            starti := 0;          -- Flag to show when to put numbers into the output FIFOs

      elsif (state="001") then           -- CONTROL = 1 => PROCESSOR SENDS A
            rst_fifo     <='1';
            if selectreg="001" then     -- Enable the fifo only if we refer to A
               en_Fa_in  <='1';
            else
               en_Fa_in   <='0';
            end if;
            en_Fb_in     <='0';
            en_Fab_out   <='0';
            en_mul       <='0';
            en_Fresh_out<='0';
            en_Fresl_out<='0';

      elsif (state="010") then           -- CONTROL = 2 => PROCESSOR SENDS B
            rst_fifo    <='1';
            en_Fa_in    <='0';
            if selectreg="010" then     -- Enable the fifo only if we refer to B
               en_Fb_in <='1';
            else
```

```
                en_Fb_in <='0';
            end if;
            en_Fab_out  <='0';
            en_mul     <='0';
            en_Fresh_out<='0';
            en_Fresl_out<='0';

        elsif (state="011") then          -- CONTROL = 3 => BEGIN TO MULTIPLY
            rst_fifo    <='1';
            en_Fa_in    <='0';
            en_Fb_in    <='0';
            en_Fab_out  <='1';

        -- We must begin to multiply one clock after we begin to send numbers from the input FIFOs

            if startm=1 then en_mul <= '1';
                else
                en_mul <= '0';
                startm := startm + 1;
            end if;

            en_Fresh_out<='0';
            en_Fresl_out<='0';

        elsif (state="100") then          -- CONTROL = 4 => READ RESULTS FROM
THE
                                          -- OUTPUT FIFOs
            rst_fifo    <='1';
            en_Fa_in    <='0';
            en_Fb_in    <='0';
            en_Fab_out  <='0';

            -- Enable the fifo only when we refer to 32 MSB of result
            if en_1='1' and en_2='0' and selectreg="011" then
             en_Fresh_out<='1';
            else
             en_Fresh_out<='0';
            end if;

            -- Enable the fifo only when we refer to 32 LSB of result
            if en_1='1' and en_2='0' and selectreg="100" then
                    en_Fresl_out<='1';
            else
             en_Fresl_out<='0';
            end if;
        end if;

        else

            en_Fab_clk  <= '0';
            en_Fres_clk <= '0';

        end if;
```

```
                -- When the OUTPUT FIFOs ARE FULL WE CAN READ THE RESULTS
                if sig_final='1' then control(3)<='1';
                end if;

                -- We begin to put the result in the output FIFOs one clock after it has been produced
                if starti=1 then en_Fres_in <= '1';
                       else en_Fres_in <= '0';
                end if;
                if Fres_rd_in='1' then starti:=1;
                end if;

                -- Notify if we have overflow
                if overflow='1' then control(4)<='1';
                       end if;

        end process;
end rtl;
```

Α3. Ουρά αποθήκευσης FIFO

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY FIFO IS
   GENERIC(
      LENGTH : positive := 4;
      DEPTH  : positive := 3
   );
   PORT(
      reset  : IN  std_logic;
      clock  : IN  std_logic;
      input  : IN  std_logic_vector (LENGTH-1 DOWNTO 0);
      output : OUT std_logic_vector (LENGTH-1 DOWNTO 0);
      en_in  : IN  std_logic;
      en_out : IN  std_logic;
      empty  : OUT std_logic;
      full   : OUT std_logic
   );
END FIFO ;

architecture rtl of FIFO is

subtype a_word is std_logic_vector(LENGTH-1 downto 0);
type a_word_array is array(natural range <>) of a_word;

SIGNAL ai : a_word_array(DEPTH-1 downto 0);
begin
       process (clock,reset)
       variable count_in,count_out :integer;
       begin
              if count_in=DEPTH then full <= '1';
              else full <= '0';
              end if;
              if count_out=0 then empty <= '1';
              else empty <='0';
              end if;

              if reset='0' then
                     for i in 0 to DEPTH-1 loop
                            ai(i) <=(others=>'0');
                     end loop;
                     count_in := 0;
                     count_out:= 10;
              elsif clock'event and clock='1' then
                     if ((en_in='1') and (count_in<DEPTH)) then
                            for i in DEPTH-1 downto 1 loop
                                   ai(i)<=ai(i-1);
                            end loop;
                            ai(0)<= input;
                            count_in := count_in + 1;
                     elsif ((en_out='1') and (count_out>0)) then
                            output <= ai(DEPTH-1);
```

```
                        for i in DEPTH-1 downto 1 loop
                              ai(i)<=ai(i-1);
                        end loop;
                        ai(0)<= (others=>'0');
                        count_out := count_out - 1;
                  end if;
            end if;
      end process;
end rtl;
```

Α4. Αθροιστής

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ADDER is
	generic (
		WIDTH : positive :=4;
		DEPTH : positive :=3
	);
	port (
		reset  : in  std_logic;
		input  : in  std_logic_vector(WIDTH-1 downto 0);
		output : out std_logic_vector(WIDTH-1 downto 0);
		overf  : out std_logic;
		enable : in  std_logic
	);
end ADDER;

architecture rtl of ADDER is

signal mid_res,temp,temp_input    : std_logic_vector (WIDTH downto 0);
signal rev_output                 : std_logic_vector (WIDTH-1 downto 0);
signal ov                         : std_logic;

begin
	process (enable,reset,mid_res)
	variable flag : integer;
	begin
		if reset='0' then
			mid_res <= (others=>'0');
			temp    <= (others=>'0');
			ov      <= '0';
			flag := DEPTH;
		elsif enable'event and flag>0 then
			if enable='0' then
				mid_res <= temp + temp_input;

				flag := flag - 1;
			elsif enable='1' then
				temp_input(WIDTH-1 downto 0)<= input;
				temp_input(WIDTH)<='0';
				temp (WIDTH-1 downto 0) <= mid_res (WIDTH-1 downto
0);
				temp (WIDTH) <='0';
			end if;
		end if;
		if mid_res(WIDTH)='1' then ov<='1';
		end if;
	end process;
	output <= mid_res(WIDTH-1 downto 0);
	overf  <= ov;
end rtl;
```

Α5. Πολλαπλασιαστής

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
LIBRARY application32;


ENTITY mul_u_cp_systolic IS
   GENERIC(
      NA : positive := 32;
      NB : positive := 32
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      a     : IN     std_logic_vector (NA-1 DOWNTO 0);
      b     : IN     std_logic_vector (NB-1 DOWNTO 0);
      ready : OUT    std_logic;
      p     : OUT    std_logic_vector (NA+NB-1 DOWNTO 0)
   );
END mul_u_cp_systolic ;

ARCHITECTURE structural OF mul_u_cp_systolic IS

   subtype a_word is std_logic_vector(NA-1 downto 0);
   type a_word_array is array(natural range <>) of a_word;

   SIGNAL ai : a_word_array(NB-1 downto 0);

   SIGNAL ao : a_word_array(NB-1 downto 0);

   SIGNAL bi : a_word_array(NB-1 downto 0);

   SIGNAL bo : a_word_array(NB-1 downto 0);

   SIGNAL so : a_word_array(NB-1 downto 0);

   SIGNAL ci : a_word_array(NB-1 downto 0);


   SIGNAL si : a_word_array(NB-1 DOWNTO 0);
   SIGNAL co : a_word_array(NB-1 DOWNTO 0);
   signal r,dummy  : std_logic;

  -- Component Declarations
   COMPONENT mul_cell_systolic
   GENERIC (
      delay_a : positive := 1;
      delay_b : positive := 1;
      delay_s : positive := 1;
      delay_c : positive := 1
   );
   PORT (
      reset : IN     std_logic ;
```

```
      clock : IN     std_logic ;
      ai    : IN     std_logic ;
      bi    : IN     std_logic ;
      si    : IN     std_logic ;
      ci    : IN     std_logic ;
      ao    : OUT    std_logic ;
      bo    : OUT    std_logic ;
      so    : OUT    std_logic ;
      co    : OUT    std_logic
   );
   END COMPONENT;
   COMPONENT shift_register
   GENERIC (
      depth : natural := 1
   );
   PORT (
      reset : IN     std_logic ;
      clock : IN     std_logic ;
      datai : IN     std_logic ;
      datao : OUT    std_logic
   );
   END COMPONENT;

   FOR ALL : mul_cell_systolic USE ENTITY application32.mul_cell_systolic;
   FOR ALL : shift_register USE ENTITY application32.shift_register;

BEGIN
   gsi:    si(0) <= (others => '0');
   r <= '1';
 -- output connections
      p(NA+NB-1) <= co(NB-1)(NA-1);

   read: shift_register
            generic map (
                        depth => 2*NB+NA-2
            )
            port map (
                        reset => reset,
                        clock => clock,
                        datai => r,
                        datao => dummy
   );

   ready<='1' when dummy='1' else '0';

   -- Instance port mappings.

   gpb: for i in 0 to NB-1 GENERATE
      gpbsr : shift_register
         GENERIC MAP (
            depth => 2*NB+NA-3-2*i
         )
         PORT MAP (
             reset=>reset,
             clock=>clock,
```

```
            datao=>p(i),
            datai => so(i)(0)
        );
END GENERATE gpb;

gcb: for i in 0 to NB-1 GENERATE
   gca: for j in 0 to NA-1 GENERATE
      gc : mul_cell_systolic
         GENERIC MAP (
            delay_a => 2,
            delay_b => 1,
            delay_s => 1,
            delay_c => 1
         )
         PORT MAP (
          reset=>reset,
                clock=>clock,
               ai => ai(i)(j),
               bi => bi(i)(j),
               si => si(i)(j),
               ci => ci(i)(j),
               ao => ao(i)(j),
               bo => bo(i)(j),
               so => so(i)(j),
               co => co(i)(j)
         );
   END GENERATE gca;

END GENERATE gcb;

gasw: for i in 1 to NB-1 GENERATE
   gssr : shift_register
      GENERIC MAP (
         depth => 1
      )
      PORT MAP (
            reset=>reset,
            clock=>clock,
            datai => co(i-1)(NA-1),
            datao => si(i)(NA-1)
      );

   ai(i) <= ao(i-1);
   si(i)(NA-2 downto 0) <= so(i-1)(NA-1 downto 1);

END GENERATE gasw;

gbciw: for i in 0 to NB-1 GENERATE
   gbcjw: for j in 1 to NA-1 GENERATE

      bi(i)(j) <= bo(i)(j-1);
      ci(i)(j) <= co(i)(j-1);

   END GENERATE gbcjw;
```

```
   END GENERATE gbciw;

   gai: for j in 0 to NA-1 GENERATE
      gaisr : shift_register
         GENERIC MAP (
            depth => j
         )
         PORT MAP (
                reset=>reset,
               clock=>clock,
                datai=>a(j),
                datao => ai(0)(j)
         );
   END GENERATE gai;

   gbi: for i in 0 to NB-1 GENERATE
      gbisr : shift_register
         GENERIC MAP (
            depth => 2*i
         )
         PORT MAP (
               reset=>reset,
                clock=>clock,
                datai=>b(i),
                datao => bi(i)(0)
         );
   END GENERATE gbi;

   gci: for i in 0 to NB-1 GENERATE

      ci(i)(0) <= '0';

   END GENERATE gci;

   gpa: for j in 1 to NA-1 GENERATE
      gpasr : shift_register
         GENERIC MAP (
            depth => NA-1-j
         )
         PORT MAP (
                reset=>reset,
               clock=>clock,
                datao=>p(j-1+NB),
                datai => so(NB-1)(j)
         );
   END GENERATE gpa;

END structural;
```

Α6. Κύτταρο Πολλαπλασιαστή

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY mul_cell_systolic IS
   GENERIC(
      delay_a : positive := 1;
      delay_b : positive := 1;
      delay_s : positive := 1;
      delay_c : positive := 1
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      ai    : IN     std_logic;
      bi    : IN     std_logic;
      si    : IN     std_logic;
      ci    : IN     std_logic;
      ao    : OUT    std_logic;
      bo    : OUT    std_logic;
      so    : OUT    std_logic;
      co    : OUT    std_logic
   );
END mul_cell_systolic ;

architecture dataflow of mul_cell_systolic is
       signal ab     : std_logic;
       signal ao_dly : std_logic_vector(delay_a downto 0);
       signal bo_dly : std_logic_vector(delay_b downto 0);
       signal so_dly : std_logic_vector(delay_s downto 0);
       signal co_dly : std_logic_vector(delay_c downto 0);
begin
       ab <= ai and bi;
       so_dly(0) <= ab xor si xor ci;
       co_dly(0) <= (ab and si) or (ab and ci) or (si and ci);
       ao_dly(0) <= ai;
       bo_dly(0) <= bi;
       ao <= ao_dly(delay_a);
       bo <= bo_dly(delay_b);
       so <= so_dly(delay_s);
       co <= co_dly(delay_c);
       ao_dly(delay_a downto 1) <= (others => '0') when reset = '1'
              else ao_dly(delay_a-1 downto 0) when rising_edge(clock)
              else ao_dly(delay_a downto 1);
       bo_dly(delay_b downto 1) <= (others => '0') when reset = '1'
              else bo_dly(delay_b-1 downto 0) when rising_edge(clock)
              else bo_dly(delay_b downto 1);
       so_dly(delay_s downto 1) <= (others => '0') when reset = '1'
              else so_dly(delay_s-1 downto 0) when rising_edge(clock)
              else so_dly(delay_s downto 1);
       co_dly(delay_c downto 1) <= (others => '0') when reset = '1'
              else co_dly(delay_c-1 downto 0) when rising_edge(clock)
              else co_dly(delay_c downto 1);
end dataflow;
```

Α7. Καταχωρητής Ολίσθησης

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY shift_register IS
   GENERIC(
      depth : natural := 1
   );
   PORT(
      reset : IN     std_logic;
      clock : IN     std_logic;
      datai : IN     std_logic;
      datao : OUT    std_logic
   );
END shift_register ;

architecture dataflow of shift_register is
signal data : std_logic_vector(depth downto 0);
begin
       data(0) <= datai;
       datao <= data(depth);
g:     if depth > 0 generate
              process(reset,clock)
              begin
                     if reset = '1' then
                            data(depth downto 1) <= (others => '0');
                     elsif clock'event and clock = '1' then
                            data(depth downto 1) <= data(depth-1 downto 0);
                     end if;
              end process;
       end generate;
end dataflow;
```

Β. Λογισμικό

```
#ifndef FUNCNAME
#define FUNCNAME filterCoreDiagnostics
#endif

/* Register Definitions */

#define CONTROL     *(unsigned long *) (baseAddress + 0x0)
#define MUL1        *(unsigned long *) (baseAddress + 0x4)
#define MUL2        *(unsigned long *) (baseAddress + 0x8)
#define RESH        *(unsigned long *) (baseAddress + 0xC)
#define RESL        *(unsigned long *) (baseAddress + 0x10)
#define SUMH        *(unsigned long *) (baseAddress + 0x14)
#define SUML        *(unsigned long *) (baseAddress + 0x18)


#define RESET       0x0
#define SEND_MUL1   0x1
#define SEND_MUL2   0x2
#define MULT        0x3
#define READ        0x4
#define READY       0x8
#define OVERFLOW    0x10

#define MASK 0xF

extern void printToPort (int, char *);
long my_div(long D,long d);
long my_mod(long D,long d);
char * string_from_int( unsigned long i);

int FUNCNAME( unsigned int baseAddress ) {

    int returnStatus = 0;
    unsigned long r1,r2,k;
    int i;

    char * s;

    printToPort (0, "\n!! \tFILTER :start of test!!!");
    printToPort (0, "\n!! Reset Internal Registers ");
    CONTROL = RESET ;                   /*Reset internal registers*/

    CONTROL = SEND_MUL1;                /* Pass the numbers  */

    MUL1=1;
    for (i=0;i!=9;i++) MUL1=MUL1+2;

    CONTROL= SEND_MUL2;

    MUL2 = 2;
    for (i=0;i!=9;i++) MUL2=MUL2+2;
```

```
    CONTROL = MULT;                              /* Start multiply */

    printToPort (0, "\n!! \t\t Numbers sent ... \n");

    k=0;
    while (k==0) {                               /*Wait until results are ready*/
       k=CONTROL ;
       k= k & READY ;
    }

   CONTROL = READ;
   for (k=0;k!=10;k++) {

          printToPort (0,"\n Result : ");
          r1=RESH;
          r2=RESL;
          s=string_from_int(r1);
          i=0;
          while ((s[i]=='0')&&(i<8)) i++;
          if (i==8)  printToPort (0,"");
          else printToPort(0,&s[i]);
          s=string_from_int(r2);
          i=0;
          while ((s[i]=='0')&&(i<8)) i++;
          if (i==8)  printToPort (0,"");
          else printToPort(0,&s[i]);
    }

    r1=SUMH;
    r2=SUML;
    printToPort(0,"\n!!Output of Filter : ");
    s=string_from_int(r1);
    i=0;
    while ((s[i]=='0')&&(i<8)) i++;
    if (i==8)  printToPort (0,"");
    else printToPort(0,&s[i]);
    s=string_from_int(r2);
    i=0;
    while ((s[i]=='0')&&(i<8)) i++;
    if (i==8)  printToPort (0,"");
    else printToPort(0,&s[i]);

    k=CONTROL;
    if ((k & OVERFLOW)==OVERFLOW)
        printToPort(0,"\n!!!There was an overflow.Ignore the result!!!");
    returnStatus=0;
    return returnStatus;
}

char  * string_from_int(unsigned long i) {

       char tempString[9];        /*We need 8 HEX characters for 32 bits*/
       unsigned long help;
       int runner;
```

```
	tempString[8]=0;

	if (i==0)
		for (runner=0;runner!=8;runner++) tempString[runner]='0';
	else {
		runner=7;
		while ((runner>-1)) {
			help = i & MASK;
			if (help<10) tempString[runner]=help+'0';
			else tempString[runner]= (help-10)+'A';
			i  = i >> 4;
			runner--;
		}
		runner++;
	}
	return (&tempString[0]);
}
```

Γ. Αρχείο XML

```
<?xml version="1.0" encoding="UTF-8"?>
<component xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
              xsi:noNamespaceSchemaLocation="schema/2.1/pxComponents.xsd">
    <vendor>MICROLAB_NTUA</vendor>
    <library>Microlab</library>
    <name>filter_32_FIFO</name>
    <version>1.00</version>
    <busInterfaces>
        <busInterface interfaceId="ambaAPB">
            <busType library="AMBA" name="ambaAPB" vendor="Mentor"/>
            <slave>
                <memoryMap>
                    <addressBlock>
                        <baseAddress configGroups="requiredConfig"
                                    format="long" id="baseAddress"
                                     prompt="Base Address:" resolve="user"/>
                        <bitOffset>0</bitOffset>
                        <range>256</range>
                        <width>32</width>
                        <register>
                            <name>control</name>
                            <addressOffset>0x0</addressOffset>
                            <size>32</size>
                            <access>read-write</access>
                            <resetValue>-1</resetValue>
                            <description>control</description>
                        </register>
                        <register>
                            <name>mul1</name>
                            <addressOffset>0x4</addressOffset>
                            <size>32</size>
                            <access>read-write</access>
                            <resetValue>-1</resetValue>
                            <description>A</description>
                        </register>
                        <register>
                            <name>mul2</name>
                            <addressOffset>0x8</addressOffset>
                            <size>32</size>
                            <access>read-write</access>
                            <resetValue>-1</resetValue>
                            <description>B</description>
                        </register>
                        <register>
                            <name>resh</name>
                            <addressOffset>0xc</addressOffset>
                            <size>32</size>
                            <access>read-only</access>
                            <resetValue>-1</resetValue>
                            <description>32 MSB of A*B</description>
                        </register>
                        <register>
```

```
                        <name>resl</name>
                        <addressOffset>0x10</addressOffset>
                        <size>32</size>
                        <access>read-only</access>
                        <resetValue>-1</resetValue>
                        <description>32 LSB of A*B</description>
                    </register>
                    <register>
                        <name>sumh</name>
                        <addressOffset>0x14</addressOffset>
                        <size>32</size>
                        <access>read-only</access>
                        <resetValue>-1</resetValue>
                        <description>32 MSB of sum</description>
                    </register>
                    <register>
                        <name>suml</name>
                        <addressOffset>0x18</addressOffset>
                        <size>32</size>
                        <access>read-only</access>
                        <resetValue>-1</resetValue>
                        <description>32 LSB of sum</description>
                    </register>
                </addressBlock>
            </memoryMap>
            <connection>required</connection>
            <signalMap>
                <signalName busSignal="PRESETN">rst</signalName>
                <signalName busSignal="PCLK">clk</signalName>
                <signalName busSignal="PSELx">psel</signalName>
                <signalName busSignal="PENABLE">penable</signalName>
                <signalName busSignal="PADDR">paddr</signalName>
                <signalName busSignal="PWRITE">pwrite</signalName>
                <signalName busSignal="PWDATA">pwdata</signalName>
                <signalName busSignal="PRDATA">prdata</signalName>
            </signalMap>
          <fileSetRef>fs-filter-diag</fileSetRef>
        </slave>
    </busInterface>
</busInterfaces>
<presentation>
  <displayLabel>Filter(sys_mult with 32bit inputs and FIFOs)</displayLabel>
  <icon>images/MicrolabIP.jpg</icon>
  <componentCategory>Peripherals</componentCategory>
</presentation>
<hwModel>
    <name>filter_32_FIFO</name>
    <verificationEnvironment id="Modelsimvhdl">
        <envIdentifier>ModelsimVhdl</envIdentifier>
        <envIdentifier>ModelsimVhdlCve</envIdentifier>
        <language>vhdl</language>
        <fileSetRef>modelsimVHDL</fileSetRef>
        <parameter name="entityName">filter_32_FIFO(struct)</parameter>
    </verificationEnvironment>
    <signalList>
```

```
        <signal>
            <name>rst</name>
            <direction>in</direction>
            <export configGroups="export" id="sig_rst" prompt="rst"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>clk</name>
            <direction>in</direction>
            <export configGroups="export" id="sig_clk" prompt="clk"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>psel</name>
            <direction>in</direction>
            <export configGroups="export" id="sig_psel" prompt="psel"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>penable</name>
            <direction>in</direction>
            <export configGroups="export" id="sig_penable" prompt="penable"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>paddr</name>
            <direction>in</direction>
            <width>32</width>
            <export configGroups="export" id="sig_paddr" prompt="paddr"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>pwrite</name>
            <direction>in</direction>
            <export configGroups="export" id="sig_pwrite" prompt="pwrite"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>pwdata</name>
            <direction>in</direction>
            <width>32</width>
            <export configGroups="export" id="sig_pwdata" prompt="pwdata"
                                          resolve="user">false</export>
        </signal>
        <signal>
            <name>prdata</name>
            <direction>out</direction>
            <width>32</width>
            <export configGroups="export" id="sig_prdata" prompt="prdata"
                                          resolve="user">false</export>
        </signal>
    </signalList>
</hwModel>
<fileSets>
 <fileSet fileSetId="modelsimVHDL">
```

```
     <file>
       <name>lib/verificationEnvironment/modelsimVHDL/filter_32_FIFO.lib</name>
       <fileType>vhdlBinaryLibrary</fileType>
        <logicalName>application32</logicalName>
     </file>
    </fileSet>
    <fileSet fileSetId="fs-filter-diag">
       <group>coreDiagnostics</group>
       <file fileId="fs-filter_32_FIFO-diag">
              <name>software/src/filter.c</name>
              <fileType>cSource</fileType>
       </file>
       <swFunction>
              <entryPoint>filterCoreDiagnostics</entryPoint>
              <fileRef>fs-filter_32_FIFO-diag</fileRef>
              <argument>
                     <name>baseAddress</name>
                     <dataType>unsigned int</dataType>
                     <value dependency="id('baseAddress')" resolve="dependent"/>
              </argument>
       </swFunction>
     </fileSet>
    </fileSets>
    <persistentInstanceData id="persistentData" resolve="user"/>
</component>
```