Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Το Λειτουργικό Σύστημα MicroEmpix/FPGA για Εφαρμογές Ενσωματωμένων Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Χ. Νάκος

Επιβλέπων : Γεώργιος Κ. Παπακωνσταντίνου
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2006

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Το Λειτουργικό Σύστημα MicroEmpix/FPGA για Εφαρμογές Ενσωματωμένων Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Χ. Νάκος

Επιβλέπων : Γεώργιος Κ. Παπακωνσταντίνου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Δεκεμβρίου 2006.

.....	.....	.....
Γ. Παπακωνσταντίνου	Κ. Πεκμεστζή	Ν. Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2006

.....
Χαράλαμπος Χ. Νάκος

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

“Not everything that can be counted counts and
not everything that counts can be counted”
Albert Einstein

Στους γονείς μου και την αδελφή μου

Περίληψη

Το MicroEmpix είναι ένα Λειτουργικό Σύστημα ανοιχτού κώδικα που χρησιμοποιείται για εκπαιδευτικούς σκοπούς στο Εργαστήριο Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου. Το MicroEmpix έχει υλοποιηθεί για εκτέλεση σε επεξεργαστές της αρχιτεκτονικής x86 της Intel, καθώς το χαμηλού επιπέδου τμήμα του έχει γραφτεί στη συμβολική γλώσσα του επεξεργαστή 8086 ενώ το υψηλού επιπέδου τμήμα του έχει γραφτεί στη γλώσσα C. Αν και το MicroEmpix είναι σχετικά μικρό σε μέγεθος, διαθέτει τα πιο σημαντικά χαρακτηριστικά των πιο περίπλοκων Λειτουργικών Συστημάτων. Αυτά τα χαρακτηριστικά περιλαμβάνουν την πολυδιεργασία, το συγχρονισμό διεργασιών με τη χρήση σηματοφορέων και λειτουργίες Ε/Ε με τη χρήση περιφερειακών μονάδων.

Σε αυτή τη Διπλωματική Διατριβή, το MicroEmpix μεταφέρθηκε στον επεξεργαστή MicroBlaze και στη συνέχεια τροποποιήθηκε και επεκτάθηκε δημιουργώντας το MicroEmpix/FPGA. Το τελευταίο μπορεί να εκτελεστεί σε Ενσωματωμένα Συστήματα βασισμένα στην τεχνολογία Field Programmable Gate Array (FPGA) και πιο συγκεκριμένα σε πλακέτες κατασκευασμένες από τη Xilinx. Η Xilinx Spartan-3E Starter Board είναι η πλακέτα στόχος και το Xilinx Embedded Development Kit (EDK) έκδοση 8.1i χρησιμοποιείται για τον προγραμματισμό της πλακέτας. Το Xilinx EDK χρησιμοποιείται μέσω του Xilinx Platform Studio (XPS), του Graphical User Interface (GUI) που το EDK παρέχει. Το MicroEmpix/FPGA υποστηρίζει τη χρήση εξειδικευμένων συνιστωσών υλικού συνδεδεμένων στη διαπροσωπεία Fast Simplex Link (FSL) του επεξεργαστή MicroBlaze, του 32-bit επεξεργαστή που χρησιμοποιείται στα Xilinx FPGAs και είναι τμήμα του Xilinx EDK. Οι εξειδικευμένες συνιστώσες υλικού μπορούν να υλοποιηθούν σε γλώσσα Verilog ή VHDL και χρησιμοποιούνται για επιτάχυνση υλικού. Εκτός από τις δυνατότητες επιτάχυνσης υλικού, το MicroEmpix/FPGA υποστηρίζει το συγχρονισμό διεργασιών με τη χρήση σηματοφορέων, λειτουργίες Ε/Ε με τη χρήση ενός Universal Asynchronous Receiver Transmitter (UART) και λειτουργίες μέτρησης με τη χρήση ενός μετρητή. Το MicroEmpix/FPGA είναι εξαιρετικά κατανοητό όσον αφορά στην υλοποίησή του και είναι, συνεπώς, σε μεγάλο βαθμό ρυθμιζόμενο και επεκτάσιμο.

Λέξεις Κλειδιά: Λειτουργικά Συστήματα, FPGA, Επεξεργαστής MicroBlaze, Ενσωματωμένα Συστήματα

Abstract

MicroEmpix is an open source Operating System that is used for educational purposes in the Computing Systems Laboratory of the National Technical University of Athens. MicroEmpix was implemented for execution on processors of the Intel x86 architecture, since its low-level part was written in the 8086 assembly language while its high-level part was written in the C language. Even though MicroEmpix is relatively small in size, a fact that makes it ideal for use with Embedded Systems based on the x86 architecture, it possesses the most important characteristics of more complicated Operating Systems. These characteristics include multitasking, process synchronization using semaphores and I/O operations using peripheral units.

In this Diploma Thesis, MicroEmpix was ported to the MicroBlaze processor and was further modified and extended creating MicroEmpix/FPGA. The latter can be executed on Embedded Systems based on the Field Programmable Gate Array (FPGA) technology and more specifically on boards developed by Xilinx. The Xilinx Spartan-3E Starter Board is the target board and the Xilinx Embedded Development Kit (EDK) version 8.1i is used for programming the board. The Xilinx EDK is used through the Xilinx Platform Studio (XPS), the Graphical User Interface (GUI) that the EDK provides. MicroEmpix/FPGA supports the use of custom hardware components connected to the Fast Simplex Link (FSL) interface of the MicroBlaze processor, the 32-bit soft processor that is used in the Xilinx FPGAs and is part of the Xilinx EDK. Custom hardware components can be implemented in the Verilog or the VHDL language and are used for hardware acceleration. Apart from the hardware acceleration capabilities, MicroEmpix/FPGA supports process synchronization with the use of semaphores, I/O operations using a Universal Asynchronous Receiver Transmitter (UART) and counting functions with the use of a counter. MicroEmpix/FPGA is extremely comprehensible concerning its implementation and is, thus, highly configurable and extensible.

Keywords: Operating Systems, FPGA, MicroBlaze Processor, Embedded Systems

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζονται κατά σειρά ο σκοπός της διπλωματικής διατριβής, τα προβλήματα που παρουσιάστηκαν και αντιμετωπίστηκαν κατά τη διάρκεια της εκπόνησής της και τα αποτελέσματα που σημειώθηκαν. Στο τέλος του κεφαλαίου παρατίθεται μία σύνοψη της εργασίας με βάση την ανάπτυξή της, όπως αυτή ακολουθείται στα επόμενα κεφάλαια.

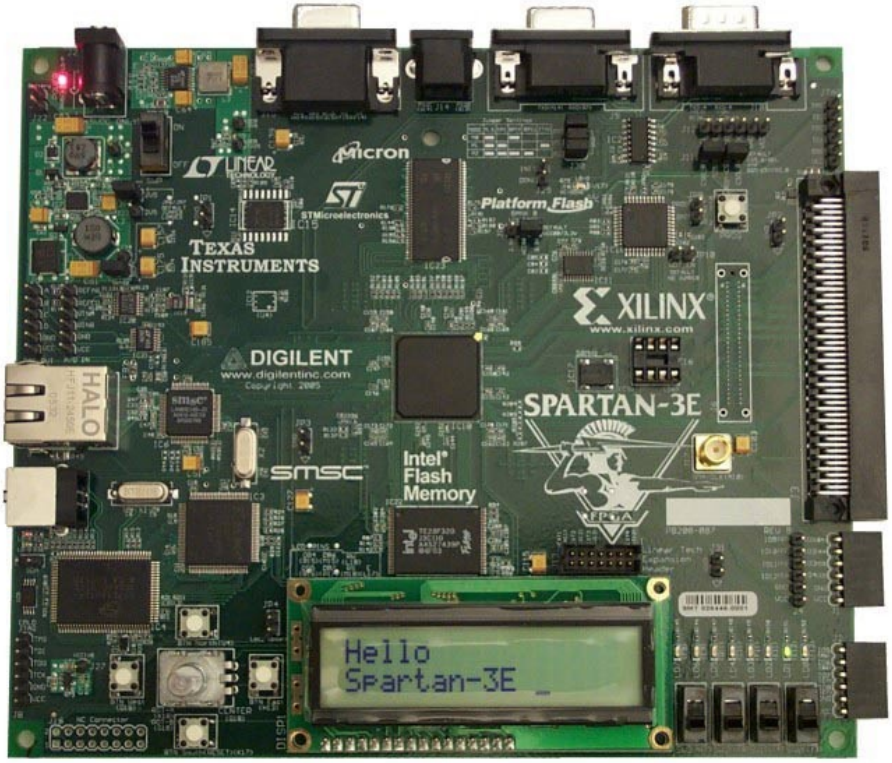
1.1 Σκοπός της Διπλωματικής Διατριβής

Το MicroEmpix είναι ένα μικρό λειτουργικό σύστημα ανοιχτού κώδικα που χρησιμοποιείται για εκπαιδευτικούς σκοπούς στο Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσοβίου Πολυτεχνείου. Αν και μικρό σε μέγεθος, διαθέτει, ωστόσο, τα σημαντικότερα χαρακτηριστικά όλων των πιο πολύπλοκων λειτουργικών συστημάτων, όπως τη δυνατότητα πολυπρογραμματισμού (multiprogramming), το συγχρονισμό διεργασιών με τη χρήση σηματοφορέων, καθώς και λειτουργίες Ε/Ε με περιφερειακές μονάδες.

Το MicroEmpix είναι ένα λειτουργικό σύστημα που μπορεί εύκολα να τροποποιηθεί και να επεκταθεί. Η χρήση του για αυτούς τους λόγους αλλά και λόγω του μικρού μεγέθους του ενδείκνυται κυρίως σε ενσωματωμένα συστήματα. Το MicroEmpix είναι γραμμένο εξ ολοκλήρου σε γλώσσα C και συμβολική γλώσσα (assembly) του επεξεργαστή 8086 της εταιρείας Intel. Λόγω της πολιτικής της συμβατότητας προς τα πίσω (backward compatibility) που ακολουθεί η Intel, το MicroEmpix μπορεί να εκτελεστεί σε οποιονδήποτε επεξεργαστή τεχνολογίας x86. Μπορεί, για παράδειγμα, να εκτελεστεί σε επεξεργαστές Pentium και Celeron, καθώς και στο ενσωματωμένο σύστημα DIMM-PC386, που έχει εσωτερική αρχιτεκτονική PC (Embedded PC).

Σκοπός της συγκεκριμένης διπλωματικής διατριβής είναι η τροποποίηση, επέκταση και μεταφορά του MicroEmpix, με τα βασικότερα χαρακτηριστικά του, σε ενσωματωμένο σύστημα που μπορεί να το υποστηρίξει και υλοποιείται πάνω σε πλακέτα τεχνολογίας FPGA (Field Programmable Gate Array) της εταιρείας Xilinx. Με την κατάλληλη διαμόρφωση του λειτουργικού συστήματος MicroEmpix θα δημιουργηθεί ένα νέο λειτουργικό σύστημα, το MicroEmpix/FPGA, το οποίο θα

μπορεί να χρησιμοποιηθεί σε ενσωματωμένα συστήματα που βασίζονται στην τεχνολογία FPGA. Το MicroEmpix/FPGA θα υποστηρίζει την εκτέλεση πολλών διεργασιών και το συγχρονισμό τους με τη χρήση σηματοφορέων. Θα υποστηρίζει, επίσης, τη χρήση ενός μετρητή για την εκτέλεση μετρήσεων με σκοπό τον έλεγχο της απόδοσης του συστήματος, τη χρήση μίας σειριακής θύρας για την επικοινωνία με άλλες συσκευές και τη χρήση μονάδων υλικού υλοποιημένων σε κάποια γλώσσα περιγραφής υλικού με σκοπό την επιτάχυνση συγκεκριμένων λειτουργιών. Η πλακέτα που θα χρησιμοποιηθεί είναι η Spartan-3E Starter Board και φαίνεται στο Σχήμα 1.1. Η τεχνολογία FPGA προσφέρει τη δυνατότητα προγραμματισμού σύνθετων κυκλωμάτων, δυνατότητα παρόμοια με αυτή που προσφέρουν και τα απλά PLDs (Programmable Logic Devices), όπως τα PLAs (Programmable Logic Arrays) λόγου χάρη. Τα FPGAs έχουν, ωστόσο, πολύ περισσότερες δυνατότητες προγραμματισμού και προσανατολίζονται στον προγραμματισμό ενσωματωμένων συστημάτων, γεγονός που οφείλεται στη διαφορετική αρχιτεκτονική τους.



Σχήμα 1.1: Spartan-3E Starter Board

Για τους σκοπούς της εργασίας θα χρησιμοποιηθεί το EDK (Embedded Development Kit) της εταιρείας Xilinx μέσα από το γραφικό αναπτυξιακό περιβάλλον που το EDK παρέχει, το XPS (Xilinx Platform Studio). Το EDK περιέχει έτοιμες συνιστώσες υλικού που μπορούν να προγραμματιστούν στο

FPGA, στις οποίες συγκαταλέγεται και ο επεξεργαστής MicroBlaze, ένας RISC (Reduced Instruction Set Computer) επεξεργαστής της εταιρείας Xilinx. Το ενσωματωμένο σύστημα που θα προγραμματιστεί θα βασίζεται στον επεξεργαστή MicroBlaze και, συνεπώς, το MicroEmpix θα τροποποιηθεί κατάλληλα βάσει της συμβολικής γλώσσας του συγκεκριμένου επεξεργαστή, καθώς και των περιφερειακών μονάδων που θα απαρτίσουν το ενσωματωμένο σύστημα. Στο σκοπό της διπλωματικής εργασίας περιλαμβάνεται και η επίδειξη με παραδείγματα εκτέλεσης της ορθής λειτουργίας του ενσωματωμένου συστήματος που εκτελεί το MicroEmpix/FPGA, τόσο σε επίπεδο προσομοίωσης, όσο και σε επίπεδο πραγματικού συστήματος.

1.2 Προβλήματα που Παρουσιάστηκαν και Αντιμετωπίστηκαν

Η μεταφορά ενός προγράμματος από ένα σύστημα σε κάποιο άλλο είναι ένα σύνθετο θέμα που συνδέεται με τις αρχιτεκτονικές των δύο συστημάτων και τη φορητότητα του προγράμματος. Ένα πρόγραμμα που είναι, για παράδειγμα, γραμμένο σε γλώσσα C μπορεί να μεταφερθεί εύκολα σε οποιοδήποτε σύστημα, αν αυτό διαθέτει μεταγλωττιστή για τη συγκεκριμένη προτυποποίηση της γλώσσας (ANSI, λόγου χάρη, στην εν λόγω περίπτωση) που ακολουθήθηκε στην αρχική ανάπτυξη του προγράμματος. Τα πράγματα γίνονται λίγο πιο δύσκολα στην περίπτωση που η προτυποποίηση διαφέρει, καθώς επιβάλλεται σε αυτήν την περίπτωση κατάλληλη τροποποίηση του προγράμματος στα σημεία που διαφέρουν οι δύο προτυποποιήσεις. Αν δεν παρέχεται μεταγλωττιστής για την ίδια γλώσσα και παρέχεται για κάποια άλλη, τότε το ζήτημα είναι η υλοποίηση ενός νέου προγράμματος στη νέα γλώσσα που να κάνει την ίδια δουλειά με το παλιό πρόγραμμα. Αυτό είναι συχνά μία επίπονη διαδικασία που σχετίζεται με τις διαφορές ανάμεσα στις φιλοσοφίες που ακολουθούν οι δύο γλώσσες[1] και απαιτεί την πολύ καλή γνώση και των δύο γλωσσών.

Ένα λειτουργικό σύστημα είναι πιο περίπλοκο από οποιοδήποτε άλλο πρόγραμμα. Λόγω της φύσης του να παρεμβάλλεται ανάμεσα στο υλικό και το λογισμικό του εκάστοτε υπολογιστικού συστήματος, απαιτεί υλοποίηση τόσο σε κάποια γλώσσα υψηλού επιπέδου, όπως είναι η C στη συγκεκριμένη περίπτωση, όσο και σε επίπεδο συμβολικής γλώσσας. Ο κώδικας σε C του αρχικού MicroEmpix δεν υπέστη σημαντικές αλλαγές[2]. Το αρχικό MicroEmpix, όμως, έχει πολλά τμήματα

1 Θα μπορούσε κάποιος να σκεφτεί πόσο δύσκολη είναι η μετατροπή ενός προγράμματος που είναι γραμμένο σε γλώσσα C σε Pascal και πόσο πιο δύσκολη μπορεί να γίνει η μετατροπή του ίδιου προγράμματος σε κάποια συναρτησιακή γλώσσα όπως, για παράδειγμα, σε OCaml. Για να προχωρήσει το θέμα λίγο παραπέρα, θα μπορούσε κάποιος να σκεφτεί τη δυσκολία της μετατροπής ενός προγράμματος από C σε Prolog ή το αντίστροφο.

2 Δεν υπέστη σημαντικές αλλαγές, ώστε να λειτουργήσει, αλλά υπέστη αλλαγές και προσθήκες με σκοπό την επέκταση των υποστηριζόμενων λειτουργιών του.

γραμμένα σε συμβολική γλώσσα του επεξεργαστή 8086 και τα οποία έπρεπε να μετατραπούν στη συμβολική γλώσσα του επεξεργαστή MicroBlaze του ενσωματωμένου συστήματος. Αυτό προαπαιτούσε την πολύ καλή γνώση των δύο συμβολικών γλωσσών και της πλήρους κατανόησης της λειτουργίας του κάθε τμήματος του κώδικα του αρχικού MicroEmpix αλλά και την πολύ καλή γνώση του υλικού.

Ένα πρόβλημα ανάλογο με το προηγούμενο είναι η διαφοροποίηση ανάμεσα στην αρχιτεκτονική όλων των περιφερειακών που συνθέτουν το αρχικό και το νέο σύστημα, καθώς και οι διαφορές ανάμεσα στις μεθόδους χειρισμού αυτών. Ένα συγκεκριμένο πρόβλημα προέκυψε κατά τη διάρκεια της ανάπτυξης του οδηγού για τη σειριακή θύρα, όταν διαπιστώθηκε ότι η σειριακή θύρα που είναι διαθέσιμη δεν υποστηρίζει ορισμένες λειτουργίες σχετικές με τις παραγόμενες διακοπές, λειτουργίες που υποστήριζε η αρχική σειριακή θύρα του DIMM-PC386. Δεν έγινε, ωστόσο, καμία θυσία στη λειτουργικότητα του MicroEmpix/FPGA, καθώς με κατάλληλη τροποποίηση του κώδικα επιτεύχθηκε το ζητούμενο. Επίσης, ο επεξεργαστής του DIMM-PC386 είχε περισσότερες δυνατότητες για το χειρισμό διακοπών σε σχέση με τον MicroBlaze αλλά και αυτό αντιμετωπίστηκε με τον ίδιο τρόπο.

Κατά τη διάρκεια της εκπόνησης της διπλωματικής εργασίας χρησιμοποιήθηκαν δύο διαφορετικές εκδόσεις του Embedded Development Kit, η έκδοση 7.1i και η έκδοση 8.1i. Κατά τη μετάβαση από τη μία έκδοση στην άλλη παρατηρήθηκαν προβλήματα στη λειτουργία του λειτουργικού συστήματος MicroEmpix/FPGA στο επίπεδο της προσομοίωσης. Αλλά και όταν διορθώθηκαν τα προβλήματα αυτά, υπήρξαν κατόπιν προβλήματα στην εκτέλεση του MicroEmpix/FPGA στο επίπεδο του πραγματικού συστήματος, τα οποία δε συμβάδιζαν με την προσομοίωση που είχε προηγηθεί. Με τη χρήση μεθόδων αποσφαλμάτωσης (debugging) εξαλείφθηκαν τελικά αυτά τα προβλήματα (τα οποία προέκυψαν από σφάλματα του προγράμματος αλλά φανέρωσαν ταυτόχρονα ασυνέπειες στο EDK), ώστε η υλοποίηση να λειτουργεί σε κάθε περίπτωση.

Προσπάθεια έγινε, τέλος, για να διαχωριστούν πλήρως οι συναρτήσεις που ήταν γραμμένες σε γλώσσα C από αυτές που ήταν γραμμένες σε συμβολική γλώσσα. Σκοπός ήταν, δηλαδή, να μη χρησιμοποιηθεί inline assembly σε κανένα σημείο του κώδικα σε C του MicroEmpix/FPGA. Αυτό το εγχείρημα στέφθηκε με επιτυχία και οδήγησε σε σαφώς πιο αναγνώσιμο κώδικα, διευκολύνοντας με αυτόν τον τρόπο τις ενδεχόμενες μελλοντικές επεκτάσεις και τροποποιήσεις του MicroEmpix/FPGA. Η δομή και ο ρόλος του λειτουργικού συστήματος MicroEmpix/FPGA σε σχέση με τις εφαρμογές του χρήστη και το υποκείμενο υλικό φαίνονται γραφικά στο Σχήμα 1.2.

Σχήμα 1.2: Το MicroEmpix/FPGA

1.3 Αποτελέσματα

Το λειτουργικό σύστημα MicroEmpix/FPGA που υλοποιήθηκε, δοκιμάστηκε επιτυχώς τόσο σε επίπεδο προσομοίωσης όσο και σε επίπεδο πραγματικού συστήματος. Στην πλακέτα Spartan-3E Starter Board έτρεξαν εφαρμογές του MicroEmpix/FPGA με πολλές διεργασίες που συγχρονίζονταν με τη χρήση σηματοφορέων και επικοινωνούσαν με έναν προσωπικό υπολογιστή μέσω της σειριακής θύρας. Η σειριακή θύρα χρησιμοποιήθηκε για τη λήψη και την αποστολή δεδομένων ενώ ενδιάμεσα γινόταν επεξεργασία των δεδομένων από τις διεργασίες που έτρεχαν στο MicroEmpix/FPGA. Δοκιμάστηκε, επίσης, η δυνατότητα χρήσης από τις διεργασίες του λειτουργικού συστήματος συνιστωσών υλικού υλοποιημένων από το χρήστη σε κάποια γλώσσα περιγραφής υλικού, συνιστώσες που υλοποιούνται με σκοπό την επιτάχυνση συγκεκριμένων λειτουργιών. Τέλος, το MicroEmpix/FPGA δοκιμάστηκε σε μία εφαρμογή ελέγχου της κίνησης μίας μάζας, όπου η πλακέτα χρησιμοποιήθηκε ως ο ελεγκτής της κίνησης της μάζας. Εκτός από την επιθυμητή λειτουργικότητα του MicroEmpix/FPGA, επετεύχθη και η αναγνωσιμότητα της υλοποίησης, η οποία είναι δομημένη και διαθέτει πολλά σχόλια, έτσι ώστε να μπορεί να βοηθήσει τις ενδεχόμενες μελλοντικές τροποποιήσεις και επεκτάσεις του λειτουργικού συστήματος.

1.4 Σύνοψη της Διπλωματικής Διατριβής

Η συνέχεια της διπλωματικής διατριβής θα έχει την ακόλουθη δομή:

Κεφάλαιο 2. Σε αυτό το κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο της διπλωματικής εργασίας. Παρουσιάζονται οι βασικές αρχές των λειτουργικών συστημάτων και οι οποίες βρίσκουν εφαρμογή στην υλοποίηση του MicroEmpix/FPGA ενώ παρουσιάζονται και ορισμένα στοιχεία των ενσωματωμένων συστημάτων.

Κεφάλαιο 3. Το περιεχόμενο του κεφαλαίου αυτού είναι η περιγραφή της αρχιτεκτονικής του υλικού που συνθέτει το ενσωματωμένο σύστημα στο οποίο θα εκτελεστεί το λειτουργικό σύστημα MicroEmpix/FPGA.

Κεφάλαιο 4. Η υλοποίηση της εργασίας είναι το θέμα αυτού του κεφαλαίου. Παρουσιάζονται αναλυτικά η υλοποίηση του MicroEmpix/FPGA και η μέθοδος δημιουργίας εφαρμογών για το λειτουργικό σύστημα.

Κεφάλαιο 5. Εδώ παρουσιάζονται τα παραδείγματα εκτέλεσης με τα αποτελέσματά τους. Καλύπτονται πολλές περιπτώσεις χρήσης του Xilinx Platform Studio και δημιουργούνται εφαρμογές που επιδεικνύουν τις λειτουργίες του MicroEmpix/FPGA.

Κεφάλαιο 6. Τα συμπεράσματα από την εκπόνηση της διπλωματικής διατριβής είναι το θέμα του τελευταίου κεφαλαίου.

Παράρτημα Α. Παρουσιάζεται ο κώδικας του MicroEmpix/FPGA.

Παράρτημα Β. Παρουσιάζονται οι εντολές του επεξεργαστή MicroBlaze.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Σε αυτό το κεφάλαιο αναπτύσσεται το θεωρητικό υπόβαθρο της διπλωματικής εργασίας. Περιγράφονται κατά σειρά τα Λειτουργικά Συστήματα και τα Ενσωματωμένα Συστήματα και αναλύονται τα σημαντικότερα στοιχεία τους.

2.1 Λειτουργικά Συστήματα

Ένα Λειτουργικό Σύστημα (Operating System) είναι ένα πρόγραμμα που τρέχει σε ένα υπολογιστικό σύστημα. Αυτή η θεώρηση είναι, ωστόσο, αρκετά απλοποιημένη και παραπλανητική από μόνη της, καθώς το λειτουργικό σύστημα δε μοιάζει με κανένα άλλο πρόγραμμα του υπολογιστικού συστήματος. Αυτό που διαφοροποιεί ουσιαστικά το λειτουργικό σύστημα από ένα οποιοδήποτε πρόγραμμα είναι ότι το πρώτο υπάρχει για να εξυπηρετεί το δεύτερο και, κατ' επέκταση, τον προγραμματιστή του προγράμματος και τον χρήστη του υπολογιστικού συστήματος.

Για να γίνει σαφέστερη η λειτουργία του λειτουργικού συστήματος, καθώς και η σχέση του με τα υπόλοιπα προγράμματα, μπορεί να θεωρηθεί η περίπτωση ενός προγράμματος επεξεργασίας κειμένου. Το πρόγραμμα αυτό μπορεί να ανακτήσει, να επεξεργαστεί και να αποθηκεύσει ένα αρχείο κειμένου. Υποστηρίζει, δηλαδή, κάποιες πολύ συγκεκριμένες λειτουργίες τις οποίες υλοποιεί ο προγραμματιστής. Η ανάκτηση ενός αρχείου από το δίσκο προϋποθέτει, φυσικά, τη γνώση της μεθόδου αναζήτησης ενός αρχείου στο δίσκο, μίας διαδικασίας που προαπαιτεί τη γνώση της δομής του εκάστοτε Συστήματος Αρχείων (File System) που χρησιμοποιείται. Αν ο προγραμματιστής ασχολούνταν με λεπτομέρειες έστω και αυτού του επιπέδου, τότε θα είχε πολλά εμπόδια να ξεπεράσει και πολύ κώδικα να γράψει, μέχρι να φτάσει στις βασικές λειτουργίες του προγράμματός του. Επίσης, ένα ενδεχόμενο σφάλμα του προγράμματός του σε τέτοιο επίπεδο θα μπορούσε να αποβεί καταστροφικό για άλλα αρχεία που είναι αποθηκευμένα στον ίδιο δίσκο.

Τη λύση σε τέτοιου είδους προβλήματα τη δίνει το λειτουργικό σύστημα. Αυτό στο συγκεκριμένο παράδειγμα αναλαμβάνει οποιαδήποτε λειτουργία πάνω στο δίσκο ενώ ταυτόχρονα παρέχει στον προγραμματιστή συναρτήσεις για απλούστερη και ασφαλέστερη πρόσβαση σε αυτόν. Το λειτουργικό σύστημα παρέχει, συνεπώς, μία διαπροσωπεία στον προγραμματιστή ενώ αποκρύπτει τη

σύνθετη υλοποίηση πολλών λειτουργιών. Παρέχει, με άλλα λόγια, τη βάση για την ανάπτυξη και την εκτέλεση προγραμμάτων. Ένα άλλο πολύ σημαντικό στοιχείο του λειτουργικού συστήματος είναι η επέκταση των δυνατοτήτων των προγραμμάτων και των δυνατοτήτων του υπολογιστικού συστήματος. Ένας επεξεργαστής, για παράδειγμα, μπορεί να εκτελεί ένα μόνο πρόγραμμα κάθε στιγμή και, όπως είναι γνωστό, η εκτέλεση ενός προγράμματος είναι μία αμιγώς ντετερμινιστική διαδικασία. Υπάρχει, ωστόσο, σχεδόν πάντα η απαίτηση να μην είναι αναγκαίο να τελειώσει ένα πρόγραμμα, ώστε να τρέξει κάποιο άλλο, δηλαδή υπάρχει η απαίτηση να εκτελούνται δύο ή παραπάνω προγράμματα ταυτόχρονα[1]. Για αυτό το λόγο πρέπει με κάποιο τρόπο να είναι δυνατή η παρέμβαση στη σειρά εκτέλεσης των προγραμμάτων. Αυτή είναι άλλη μία λειτουργία που αναλαμβάνει να εκτελέσει το λειτουργικό σύστημα. Παρακάτω παρατίθενται αναλυτικότερα τα βασικότερα στοιχεία που χαρακτηρίζουν ένα λειτουργικό σύστημα.

2.1.1 Διαχείριση Διεργασιών

Πριν αναλυθεί αυτή η πολύ σημαντική λειτουργία ενός λειτουργικού συστήματος, θα πρέπει, προφανώς, να οριστεί πρώτα τι είναι η διεργασία. Είναι πολύ συνηθισμένη η σύγχυση όσον αφορά στη διαφοροποίηση μίας διεργασίας από το πρόγραμμα το οποίο αυτή εκτελεί. Μία διεργασία είναι, λοιπόν, το πρόγραμμα το οποίο εκτελεί, συμπεριλαμβανομένων, ωστόσο, των τρεχουσών τιμών του μετρητή προγράμματος, των καταχωρητών και των μεταβλητών, δηλαδή των περιεχομένων της μνήμης. Είναι φανερή η δυναμική φύση της διεργασίας σε σύγκριση με τη στατική φύση του προγράμματος, που δεν είναι παρά μία αλληλουχία εντολών. Κάθε διεργασία έχει, κατά κάποιο τρόπο, τη δική της εικονική κεντρική μονάδα επεξεργασίας, δηλαδή όσο την αφορά την ίδια τη διεργασία, υπάρχει για αυτή μία αφοσιωμένη μονάδα επεξεργασίας. Στην πραγματικότητα, όμως, και μιλώντας για τη συντριπτική πλειοψηφία των υπολογιστικών συστημάτων, η κεντρική μονάδα επεξεργασίας είναι μία και οι διεργασίες λαμβάνουν η μία μετά την άλλη τη θέση τους σε αυτή. Αυτή η αλλαγή στην εκτέλεση των διεργασιών γίνεται με πολύ αυστηρό τρόπο, ώστε να διατηρείται η σημασιολογία του εκτελέσιμου προγράμματος για κάθε διεργασία. Αυτή η αλλαγή ονομάζεται Μεταγωγή Περιεχομένου (Context Switching) και είναι από τις βασικότερες λειτουργίες ενός λειτουργικού συστήματος που υποστηρίζει πολυπρογραμματισμό, δηλαδή την ταυτόχρονη ύπαρξη και εκτέλεση περισσοτέρων της μίας διεργασιών.

1 Κυριολεκτικά αυτό είναι προφανώς αδύνατο σε μία αρχιτεκτονική ενός επεξεργαστή. Με το παραπάνω γίνεται αναφορά στην περίπτωση της ψευδοπαραλληλίας προγραμμάτων, κατά την οποία τα προγράμματα (ή πιο σωστά, όπως θα εξηγηθεί στη συνέχεια, οι διεργασίες οι οποίες τα εκτελούν) εναλλάσσονται στον επεξεργαστή και δίνουν την εντύπωση στον χρήστη ότι εκτελούνται ταυτόχρονα.

Είναι σκόπιμο σε αυτό το σημείο να δοθεί ένα απλό παράδειγμα για τη λειτουργία που πρέπει να επιτελεί η μεταγωγή περιεχομένου, για την καλύτερη κατανόηση της λειτουργίας αυτής. Έστω ότι κάποια στιγμή σε ένα υπολογιστικό σύστημα που είναι βασισμένο στον επεξεργαστή 8086 υπάρχουν δύο διεργασίες, οι P1 και P2. Έστω ότι ο κώδικας του προγράμματος που εκτελεί η διεργασία P1 είναι ο εξής:

```
...
; Σε αυτό το σημείο ο καταχωρητής ax περιέχει την τιμή της μεταβλητής x
inc     ax
add     ax, bx
; Μετά από αυτό το σημείο η μεταβλητή x παίρνει την τιμή του καταχωρητή ax
...
```

Έστω ότι ο κώδικας του προγράμματος που εκτελεί η διεργασία P2 είναι ο εξής:

```
...
; Σε αυτό το σημείο ο καταχωρητής ax περιέχει την τιμή της μεταβλητής y
dec     ax
add     ax, cx
; Μετά από αυτό το σημείο η μεταβλητή y παίρνει την τιμή του καταχωρητή ax
...
```

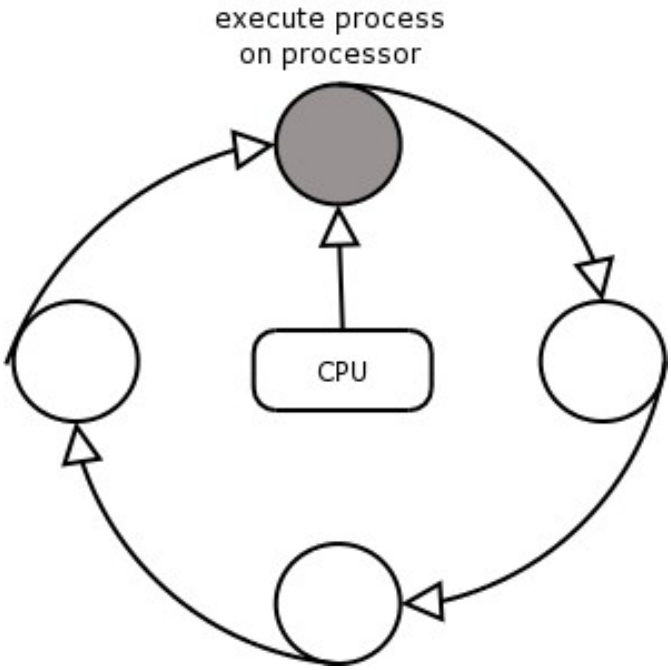
Στο πρώτο πρόγραμμα η μεταβλητή *x* αυξάνεται κατά την τιμή του καταχωρητή *bx* συν 1. Στο δεύτερο πρόγραμμα η μεταβλητή *y* αυξάνεται κατά την τιμή του καταχωρητή *cx* πλην 1. Έστω τώρα ότι μία διεργασία εκτελείται για ένα κύκλο ρολογιού κάθε φορά, δηλαδή εκτελείται μία μόνο εντολή από τη διεργασία κάθε φορά, και μετά ομοίως συνεχίζει την εκτέλεσή της η άλλη διεργασία. Το αποτέλεσμα θα είναι ο επεξεργαστής να εκτελέσει τις εξής εντολές:

```
...
inc     ax
...
dec     ax
```

```
...
add     ax, bx
...
add     ax, cx
...
```

Αν και ο καταχωρητής *ax* είναι μοναδικός στον επεξεργαστή 8086, χρησιμοποιείται, ωστόσο, και από τις δύο διεργασίες που εκτελούνται (και μάλιστα σε όλες τις εντολές που εκτελούν οι διεργασίες αυτές στα συγκεκριμένα τμήματα των προγραμμάτων τους). Είναι φανερό ότι θα πρέπει η τιμή του για τη μία διεργασία να φυλάσσεται για επόμενη χρήση και η παλιά τιμή του για την άλλη διεργασία να επαναφέρεται κατά την εναλλαγή της εκτέλεσης των δύο διεργασιών, ώστε να διατηρείται η σημασιολογία και των δύο προγραμμάτων. Αυτή η διαδικασία θα πρέπει να λαμβάνει χώρα ανάμεσα στην εκτέλεση εντολών διαφορετικών διεργασιών, γιατί σε διαφορετική περίπτωση τα αποτελέσματα της εκτέλεσης θα ήταν απροσδιόριστα. Οι καταχωρητές φυλάσσονται για κάθε διεργασία σε μία δομή του λειτουργικού συστήματος που ονομάζεται Σύνολο Ελέγχου Διεργασίας (Process Control Block) μαζί με άλλες πληροφορίες για τη διεργασία. Σε αυτή τη δομή αποθηκεύει η ρουτίνα μεταγωγής περιεχομένου το περιεχόμενο των καταχωρητών της διεργασίας της οποίας διακόπτεται η εκτέλεση και από αυτή τη δομή φορτώνει η ρουτίνα μεταγωγής περιεχομένου το περιεχόμενο των καταχωρητών της διεργασίας της οποίας συνεχίζεται η εκτέλεση.

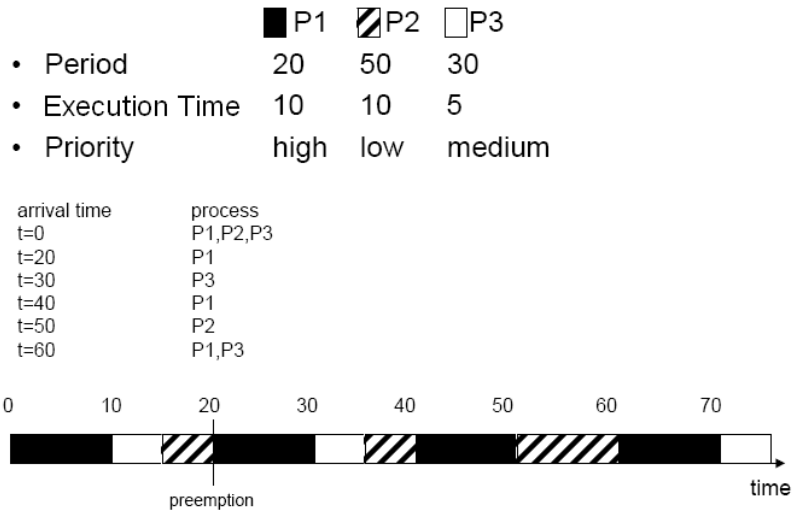
Η Διαχείριση Διεργασιών (Process Management) είναι εκείνη η λειτουργία του λειτουργικού συστήματος που αφορά στις λεπτομέρειες σχετικά με τη σειρά εκτέλεσης και το χρόνο εκτέλεσης που δίνονται στην κάθε διεργασία, ώστε να χρησιμοποιήσει τον επεξεργαστή. Σε ένα λειτουργικό σύστημα σε μία συγκεκριμένη χρονική στιγμή υπάρχει ένας Αλγόριθμος Χρονοδρομολόγησης (Scheduling Algorithm) που καθορίζει πότε θα γίνει η επόμενη εναλλαγή διεργασιών και ποια διεργασία θα είναι η επόμενη που θα εκτελεστεί. Για παράδειγμα, ο πιο απλός αλγόριθμος χρονοδρομολόγησης, ο αλγόριθμος Round-Robin που φαίνεται γραφικά στο Σχήμα 2.1, θεωρεί ότι οι διεργασίες εισάγονται αρχικά σε μία κυκλική λίστα. Στη συνέχεια, με κάποια συγκεκριμένη χρονική περίοδο να παρεμβάλλεται, η κάθε φορά τρέχουσα διεργασία δίνει τη θέση της στην επόμενη στη λίστα διεργασία. Η περίοδος που παρεμβάλλεται και κατά τη διάρκεια της οποίας η τρέχουσα διεργασία εκτελείται ονομάζεται Κβάντο Χρόνου (Time Quantum).



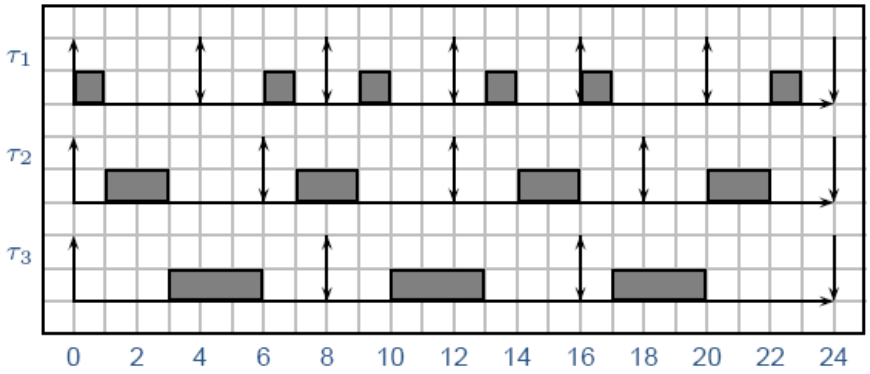
Σχήμα 2.1: Αλγόριθμος Χρονοδρομολόγησης Round-Robin

Υπάρχουν διάφοροι τύποι αλγορίθμων χρονοδρομολόγησης που χρησιμοποιούνται σε πολλών ειδών εφαρμογές. Ο τύπος της εφαρμογής καθορίζει ποιος αλγόριθμος ενδείκνυται για αυτή. Σε εφαρμογές πραγματικού χρόνου και για διεργασίες που δεν έχουν την ίδια προτεραιότητα ή επιθυμητή συχνότητα εκτέλεσης χρησιμοποιούνται αλγόριθμοι που λαμβάνουν τα παραπάνω υπόψη και έχουν, συνεπώς, πιο δυναμική συμπεριφορά από τον αλγόριθμο Round-Robin. Τέτοιοι αλγόριθμοι είναι, λόγου χάρη, ο RMS (Rate Monotonic Scheduling) και ο EDF (Earliest Deadline First). Στον αλγόριθμο χρονοδρομολόγησης RMS κάθε διεργασία λαμβάνει εξαρχής μία προτεραιότητα που εξαρτάται από την επιθυμητή συχνότητα εκτέλεσης της συγκεκριμένης διεργασίας. Όσο υψηλότερη είναι αυτή η συχνότητα εκτέλεσης, δηλαδή όσο μικρότερη η περίοδος εκτέλεσης, τόσο υψηλότερη είναι και η προτεραιότητα της διεργασίας. Στον RMS οι προτεραιότητες των διεργασιών είναι στατικές, δηλαδή δε μεταβάλλονται κατά την εκτέλεσή τους. Κατά τον αλγόριθμο RMS κάθε στιγμή εκτελείται η διεργασία που έχει την υψηλότερη προτεραιότητα και ταυτόχρονα δεν έχει ολοκληρώσει την εκτέλεσή της για την τρέχουσα περίοδό της. Η λειτουργία του RMS φαίνεται γραφικά στο Σχήμα 2.2. Στον αλγόριθμο χρονοδρομολόγησης EDF οι προτεραιότητες δίνονται δυναμικά στις διεργασίες, δηλαδή κατά την εκτέλεσή τους. Και σε αυτή την περίπτωση, όπως στην περίπτωση του RMS, οι διεργασίες έχουν μία επιθυμητή περίοδο εκτέλεσης. Τη μεγαλύτερη προτεραιότητα έχει κάθε στιγμή η διεργασία που δεν έχει ολοκληρώσει την εκτέλεσή της για την τρέχουσα περίοδό της και της οποίας πλησιάζει

όσο σε καμία άλλη διεργασία η λήξη της περιόδου αυτής, δηλαδή η λήξη της προθεσμίας που έχει η διεργασία για να εκτελεστεί. Αυτή η διεργασία επιλέγεται για εκτέλεση από τον αλγόριθμο EDF. Η λειτουργία του EDF φαίνεται γραφικά στο Σχήμα 2.3.



Σχήμα 2.2: Λειτουργία του RMS



Σχήμα 2.3: Λειτουργία του EDF

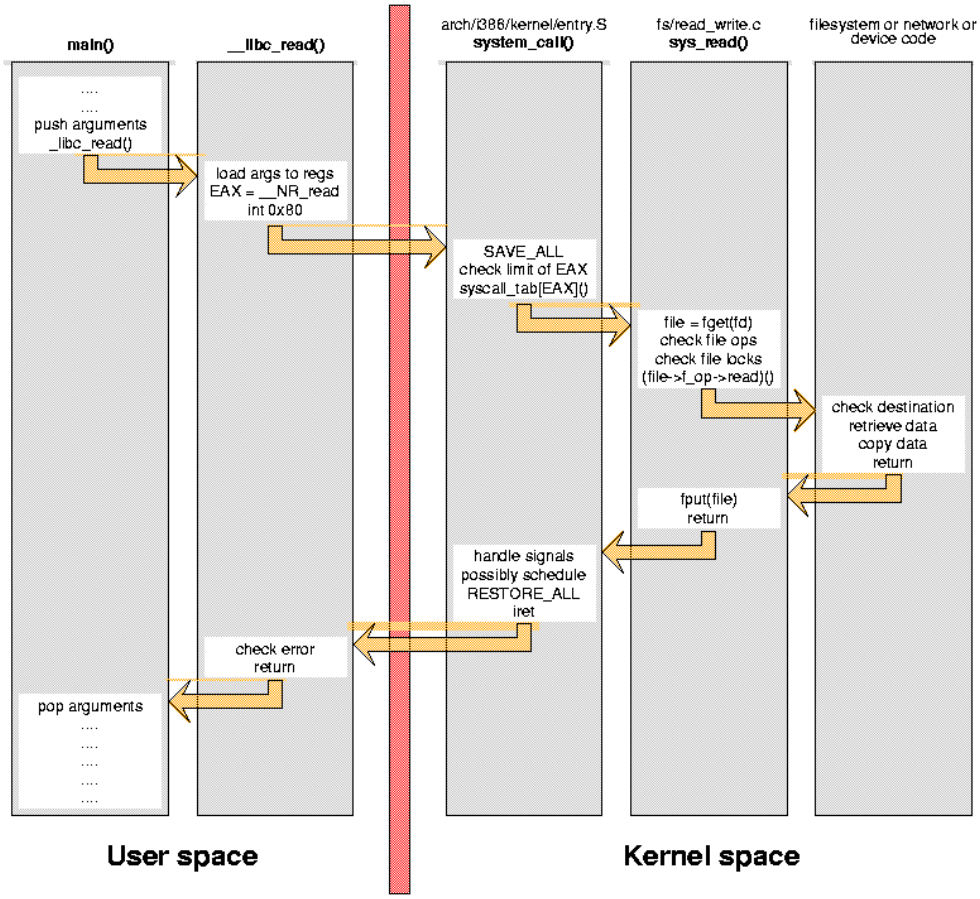
2.1.2 Κλήσεις Συστήματος

Όπως αναφέρθηκε προηγουμένως, ένα λειτουργικό σύστημα έχει ως στόχο τη διεύρυνση των δυνατοτήτων των προγραμμάτων που κατασκευάζονται για την αρχιτεκτονική η οποία το φιλοξενεί, αποσκοπώντας ταυτόχρονα στην πληρέστερη και αποδοτικότερη εκμετάλλευση της αρχιτεκτονικής. Κάποιες από τις λειτουργίες τις οποίες επιτελεί ένα λειτουργικό σύστημα, όπως για παράδειγμα η μεταγωγή περιεχομένου ή η γνωστή, όπως προτάθηκε από τον E. W. Dijkstra το 1965, λειτουργία P

πάνω σε έναν σηματοφορέα, η οποία χρησιμοποιείται για τον συγχρονισμό και τον αμοιβαίο αποκλεισμό (mutual exclusion) μεταξύ διεργασιών, είναι απαραίτητο να αποκρύπτονται πλήρως ή εν μέρει από τον προγραμματιστή. Στην περίπτωση της μεταγωγής περιεχομένου ο προγραμματιστής αφενός δε χρειάζεται να γνωρίζει απολύτως τίποτα για την υλοποίησή της, αφετέρου δεν επιθυμεί την ανάμειξή του στην εν λόγω λειτουργία. Το μόνο που τον ενδιαφέρει είναι, βεβαίως, να γίνεται σωστά και βασίζεται στο λειτουργικό σύστημα για αυτό. Στην περίπτωση, όμως, της λειτουργίας P πάνω σε έναν σηματοφορέα ο προγραμματιστής ενδιαφέρεται, προφανώς, να χρησιμοποιήσει τη λειτουργία αυτή στα προγράμματά του. Η υλοποίηση δεν τον ενδιαφέρει. Θα μπορούσε, ωστόσο, να υλοποιήσει και ο ίδιος τη λειτουργία αυτή. Η λειτουργία P, όμως, πρέπει να γίνεται αδιαίρετα (χωρίς να μπορεί να διακοπεί) και αυτό θα έπρεπε να το εξασφαλίσει ο προγραμματιστής ελέγχοντας τις διακοπές του συστήματος, εργαζόμενος, δηλαδή, σε πολύ χαμηλό, από προγραμματιστική σκοπιά, επίπεδο[1]. Κάτι τέτοιο θα ήταν χρονοβόρο για τον προγραμματιστή, θα αύξανε τη περιπλοκότητα του προγράμματος και θα ενείχε σημαντικό ενδεχόμενο σφάλματος. Είναι απαραίτητο, συνεπώς, ένα επίπεδο αφαίρεσης για ορισμένες στοιχειώδεις λειτουργίες, το οποίο εξυπηρετεί ταυτόχρονα τόσο την απλότητα όσο και την ασφάλεια στον προγραμματισμό. Αυτό παρέχεται από το λειτουργικό σύστημα με ένα κατάλληλο API (Application Programming Interface), τις Κλήσεις Συστήματος (System Calls).

Οι κλήσεις συστήματος είναι, λοιπόν, ένα σύνολο συναρτήσεων που παρέχονται από το λειτουργικό σύστημα στον προγραμματιστή και επεκτείνουν τις δυνατότητες των προγραμμάτων του. Ως παράδειγμα μπορεί να αναφερθεί η συνάρτηση *read* του Linux που χρησιμοποιείται για διάβασμα χαρακτήρων από κάποιο αρχείο και την οποία μπορεί κάποιος να χρησιμοποιήσει στα προγράμματά του, χωρίς να τον απασχολούν λεπτομέρειες σχετικά με την υλοποίησή της. Το λειτουργικό σύστημα έχει την ευθύνη της σωστής υλοποίησης του διαβάσματος χαρακτήρων και, στη γενική περίπτωση, της λειτουργίας που προσφέρει μέσω μίας κλήσης συστήματος. Το τελευταίο παράδειγμα φαίνεται γραφικά στο Σχήμα 2.4, όπου και διακρίνονται τα επίπεδα αφαίρεσης και η τυποποίηση που χαρακτηρίζουν τις κλήσεις συστήματος στο Linux.

1 Επίσης, οι λειτουργίες των σηματοφορέων εμπλέκουν και τους αλγορίθμους χρονοδρομολόγησης, οπότε τα πράγματα γίνονται ακόμα πιο δύσκολα.



Σχήμα 2.4: Κλήση Συστήματος read

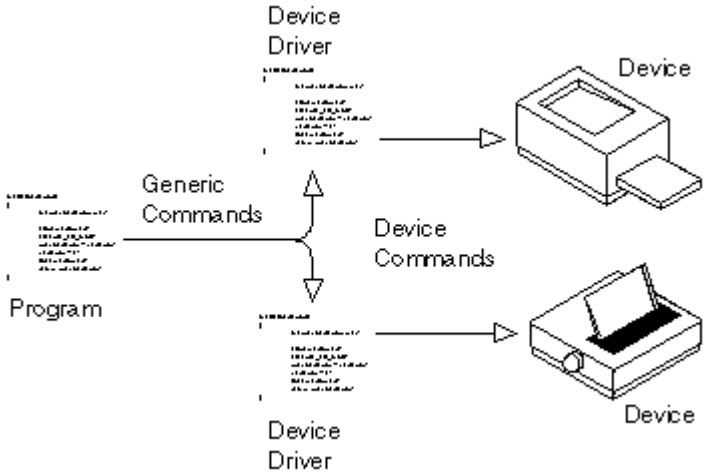
2.1.3 Οδηγοί Συσκευών

Κάθε υπολογιστικό σύστημα είναι προϊόν σύνθεσης ποικίλων συνιστωσών υλικού, δηλαδή ποικίλων συσκευών. Η αρχιτεκτονική και η μέθοδος χρήσης διαφέρουν σημαντικά, όχι μόνο ανάμεσα σε διαφορετικούς τύπους συσκευών, αλλά και ανάμεσα σε διαφορετικές συσκευές του ίδιου τύπου. Το ζητούμενο είναι, φυσικά, για κάθε συσκευή πώς μπορεί να χρησιμοποιηθεί μέσα από κάποιο πρόγραμμα, το οποίο εκτελείται στο περιβάλλον ενός λειτουργικού συστήματος. Τη λύση τη δίνει μία συγκεκριμένη κατηγορία λογισμικού, οι Οδηγοί Συσκευών (Device Drivers). Ένας οδηγός συσκευής είναι ένα σύνολο συναρτήσεων με κλήση των οποίων μέσα από ένα πρόγραμμα προκαλείται η εκτέλεση των επιθυμητών λειτουργιών της εκάστοτε συσκευής.

Η περιπλοκότητα ενός οδηγού συσκευής έγκειται στο γεγονός ότι στη γενική περίπτωση εξαρτάται τόσο από τη συγκεκριμένη συσκευή όσο και από το υποκείμενο λειτουργικό σύστημα. Ως παράδειγμα μπορεί να θεωρηθεί μία σειριακή θύρα που παράγει μία διακοπή κάθε φορά που έχει

ολοκληρώσει την αποστολή όλων των χαρακτήρων που είχε στον απομονωτή εξόδου της. Στην υλοποίηση ενός οδηγού για αυτήν τη συσκευή εμπλέκεται, εκτός φυσικά από την ίδια τη θύρα, και το σύστημα διακοπών, που είναι ένα πολύ ευαίσθητο στοιχείο ενός οποιουδήποτε λειτουργικού συστήματος.

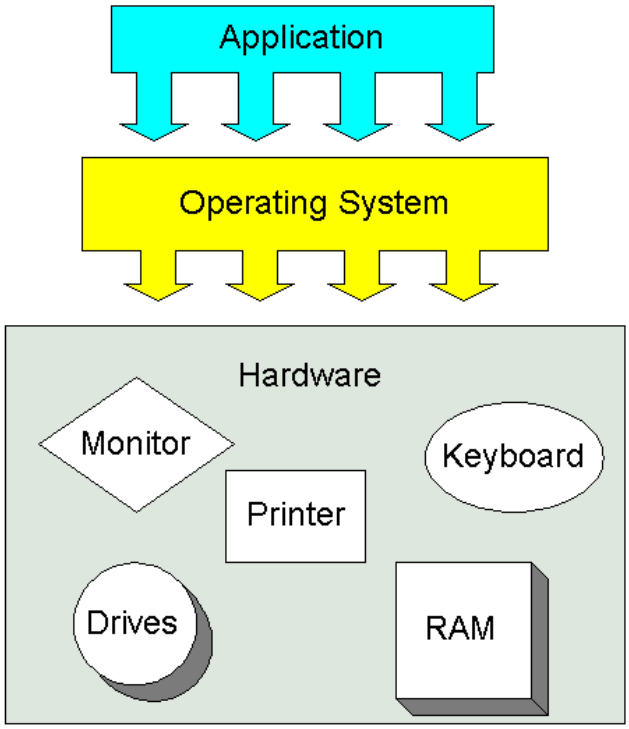
Με την εισαγωγή των οδηγών συσκευών παρέχεται τελικά στον προγραμματιστή ένα API που χρησιμοποιείται για την ανάπτυξη προγραμμάτων που χρησιμοποιούν την κάθε συσκευή. Όπως συμβαίνει με κάθε API, έτσι και με τους οδηγούς συσκευών στόχος είναι η επίτευξη του κατάλληλου επιπέδου αφαίρεσης, ώστε να είναι πλήρως λειτουργικοί αλλά ταυτόχρονα να μην μπλέκουν τον προγραμματιστή με λεπτομέρειες υλοποίησης που οφείλονται σε χαρακτηριστικά της συσκευής ή του λειτουργικού συστήματος που χρησιμοποιείται. Επίσης, με αυτόν τον τρόπο επιτυγχάνονται μεγαλύτερη ασφάλεια στον προγραμματισμό και μεγαλύτερη ευκολία στην ανίχνευση λαθών. Στο Σχήμα 2.5 φαίνεται με γραφικό τρόπο η χρήση των οδηγών μέσα από κάποιο πρόγραμμα για τον έλεγχο των αντίστοιχων συσκευών.



Σχήμα 2.5: Οδηγοί Συσκευών

Συνοψίζοντας, παραπάνω φάνηκε ο ρόλος ενός λειτουργικού συστήματος μέσα από τις βασικότερες εργασίες που αυτό επιτελεί. Και ήταν αφενός πιο εύκολο και αφετέρου πιο χρήσιμο να οριστεί μέσω αυτών των λειτουργιών του παρά με κάποιο πιο τυπικό τρόπο. Η λίστα, ωστόσο, των λειτουργιών που αναλύθηκαν και αφορούν σε κάθε λειτουργικό σύστημα δεν ήταν σε καμία περίπτωση εξαντλητική, καθώς ανάλογα με την εφαρμογή ένα λειτουργικό σύστημα μπορεί να διαφέρει σημαντικά από κάποιο άλλο. Το ίδιο συμβαίνει, εξάλλου, και με τα υπολογιστικά συστήματα. Αυτή η διαφοροποίηση, μάλιστα, μεταξύ των υπολογιστικών συστημάτων συνεπάγεται και τη

διαφοροποίηση μεταξύ των λειτουργικών συστημάτων που αναπτύσσονται για αυτά. Ο ρόλος του λειτουργικού συστήματος φαίνεται γραφικά στο Σχήμα 2.6.



Σχήμα 2.6: Ο Ρόλος του Λειτουργικού Συστήματος

2.2 Ενσωματωμένα Συστήματα

Τα Ενσωματωμένα Συστήματα (Embedded Systems) είναι υπολογιστικά συστήματα ειδικού σκοπού. Όπως και κάθε υπολογιστικό σύστημα, εκτελούν κάποιον υπολογισμό ή, πιο γενικά, κάποια εργασία. Αυτό που είναι ιδιαίτερα σημαντικό είναι το γεγονός ότι αυτός ο υπολογισμός είναι πολύ συγκεκριμένος και συνυφασμένος με το ενσωματωμένο σύστημα που τον εκτελεί. Αυτό το στοιχείο κάνει ιδιαίτερα σημαντικά τα ενσωματωμένα συστήματα, καθώς έχει ποικίλες συνέπειες που θα αναλυθούν παρακάτω, αλλά και τα διαφοροποιεί από τα υπολογιστικά συστήματα γενικού σκοπού, όπως τους προσωπικούς υπολογιστές.

Ένα αντιπροσωπευτικό παράδειγμα ενσωματωμένου συστήματος είναι το κινητό τηλέφωνο. Ένα σύγχρονο κινητό τηλέφωνο έχει επεξεργαστή, μνήμη, λειτουργικό σύστημα και υποστηρίζει την εκτέλεση πολλών λειτουργιών ακόμα και παράλληλα, όπως την ομιλία και την περιήγηση στα

περιεχόμενά του. Το κινητό τηλέφωνο έχει μικρότερο μέγεθος και καταναλώνει λιγότερη ισχύ σε σύγκριση με έναν φορητό προσωπικό υπολογιστή, ο οποίος, εξοπλισμένος με τα κατάλληλα περιφερειακά και τις κατάλληλες υπηρεσίες, μπορεί να εκτελέσει τις λειτουργίες του κινητού τηλεφώνου και ακόμα περισσότερες. Οι περισσότερες, όμως, λειτουργίες του φορητού υπολογιστή δεν είναι απαραίτητες στο χρήστη του, αν αυτός θέλει να τον χρησιμοποιήσει μόνο αντί κινητού τηλεφώνου, και η μικρή αυτονομία του σε συνδυασμό με το μεγάλο του μέγεθος τον καθιστούν τελικά ακατάλληλο για την εν λόγω χρήση. Κάπως έτσι γεννιέται η ανάγκη κατασκευής ενός ενσωματωμένου συστήματος, ενός συστήματος, που θα ικανοποιεί συγκεκριμένες προδιαγραφές και θα εκτελεί συγκεκριμένες λειτουργίες. Στη συνέχεια του κεφαλαίου αναπτύσσονται τα σημαντικότερα θέματα που αφορούν στα ενσωματωμένα συστήματα.

2.2.1 Χαρακτηριστικά Μεγέθη Σχεδίασης Συστημάτων

Πριν γίνει αναφορά στις τεχνολογίες που χρησιμοποιούνται σε ενσωματωμένα συστήματα, είναι σκόπιμο να αναλυθούν ορισμένοι παράγοντες που λαμβάνονται υπόψη στη σχεδίαση των συστημάτων γενικά. Ο σχεδιαστής ενός συστήματος πρέπει, προφανώς, να επιδιώξει και να εξασφαλίσει ότι το σύστημά του έχει μία συγκεκριμένη λειτουργικότητα. Εξίσου σημαντικό είναι, ωστόσο, να βελτιστοποιήσει ορισμένα μεγέθη που χαρακτηρίζουν τη διαδικασία ανάπτυξης του συστήματος και των οποίων μία όχι εξαντλητική λίστα είναι η εξής:

Κόστος σχεδίασης (NRE cost[1]). Το κόστος της διαδικασίας σχεδίασης του συστήματος. Μετά το τέλος της διαδικασίας αυτής, οποιοσδήποτε αριθμός μονάδων του συστήματος και να παραχθεί δεν εισάγει πρόσθετο σχεδιαστικό κόστος.

Κόστος μονάδας (unit cost). Το κόστος της παραγωγής ενός κομματιού του συστήματος. Δεν περιέχεται, βέβαια, το κόστος σχεδίασης στο κόστος μονάδας.

Μέγεθος (size). Ο φυσικός χώρος που απαιτείται από το σύστημα. Συχνά μετριέται σε bytes για το λογισμικό και σε πύλες ή τρανζίστορ για το υλικό.

Απόδοση (performance). Ο χρόνος εκτέλεσης ή ο ρυθμός διεκπεραίωσης εργασιών (throughput) του συστήματος.

Ισχύς (power). Η καταναλισκόμενη από το σύστημα ισχύς, που καθορίζει τη διάρκεια ζωής της μπαταρίας ή τις απαιτήσεις ψύξεως των ολοκληρωμένων κυκλωμάτων, καθώς περισσότερη ισχύς

1 Non-Recurring Engineering cost

σημαίνει και περισσότερη θερμότητα.

Ευελιξία (flexibility). Η δυνατότητα τροποποίησης της λειτουργικότητας του συστήματος χωρίς την επιβολή μεγάλου πρόσθετου σχεδιαστικού κόστους. Τυπικά το λογισμικό θεωρείται πολύ ευέλικτο.

Χρόνος άφιξης στην αγορά (time-to-market). Ο χρόνος που χρειάζεται για τη σχεδίαση και την κατασκευή του συστήματος μέχρι το σημείο που το σύστημα μπορεί να πουληθεί στους πελάτες.

Χρόνος για την κατασκευή πρωτοτύπου (time-to-prototype). Ο χρόνος που χρειάζεται για την κατασκευή μίας λειτουργικής έκδοσης του συστήματος, η οποία μπορεί να είναι μεγαλύτερη ή πιο ακριβή από την τελική υλοποίηση του συστήματος. Μπορεί να χρησιμοποιηθεί, ωστόσο, για την εξακρίβωση της χρησιμότητας και της ορθότητας του συστήματος, καθώς και για την εκλέπτυνση της λειτουργικότητας του συστήματος.

Ορθότητα (correctness). Η πεποίθηση ότι η λειτουργικότητα του συστήματος έχει υλοποιηθεί σωστά. Ο έλεγχος της λειτουργικότητας μπορεί να γίνει καθ' όλη τη διάρκεια της σχεδίασης του συστήματος και μπορεί, επίσης, να γίνει εισαγωγή κυκλωμάτων ελέγχου, ώστε να εξακριβωθεί ότι η κατασκευή έγινε σωστά.

Ασφάλεια (safety). Η πιθανότητα ότι το σύστημα δε θα προκαλέσει οποιαδήποτε βλάβη.

Είναι, ίσως, αρκετά εύλογο το γεγονός ότι αυτά τα μεγέθη ανταγωνίζονται το ένα το άλλο και η βελτίωση του ενός πολύ συχνά συνεπάγεται την επιδείνωση του άλλου. Για παράδειγμα, αν μειωθεί το μέγεθος, μπορεί να υπάρξει απώλεια σε απόδοση. Επίσης, αν δοθεί βάρος στην εξασφάλιση της ορθότητας, μπορεί να καθυστερήσει να βγει το προϊόν στην αγορά. Σκοπός του σχεδιαστή είναι να βρει τη χρυσή τομή ανάμεσα σε όλους αυτούς τους παράγοντες που υπεισέρχονται στη διαδικασία της σχεδίασης. Γι' αυτό θα πρέπει να είναι καλός γνώστης της τεχνολογίας που υφίσταται κάθε χρονική στιγμή, ώστε να μπορεί να κάνει την καταλληλότερη επιλογή.

2.2.2 Τεχνολογία Ολοκληρωμένων Κυκλωμάτων

Κάθε σύστημα πρέπει τελικά να υλοποιηθεί σε ένα ολοκληρωμένο κύκλωμα (integrated circuit). Η τεχνολογία ολοκληρωμένων κυκλωμάτων αναφέρεται στον τρόπο κατά τον οποίο απεικονίζεται μία ψηφιακή υλοποίηση επιπέδου πύλης σε ένα ολοκληρωμένο κύκλωμα. Ένα ολοκληρωμένο κύκλωμα είναι μία διάταξη ημιαγωγών που αποτελείται από ένα σύνολο από transistors, καθώς και από άλλες διατάξεις. Υπάρχει ένας αριθμός από διαφορετικές διεργασίες για την

κατασκευή ημιαγωγών, η πιο δημοφιλής από τις οποίες είναι η CMOS (Complementary Metal Oxide Semiconductor). Οι τεχνολογίες ολοκληρωμένων κυκλωμάτων διαφέρουν ως προς το πόσο εξειδικευμένο είναι το ολοκληρωμένο κύκλωμα για μία συγκεκριμένη υλοποίηση.

Για την κατανόηση των διαφορών ανάμεσα στις τεχνολογίες ολοκληρωμένων κυκλωμάτων, πρέπει να αναγνωριστεί πρώτα ότι οι ημιαγωγοί αποτελούνται από διάφορα στρώματα. Τα κατώτερα στρώματα σχηματίζουν τα transistors. Τα μεσαία στρώματα σχηματίζουν τις λογικές πύλες. Τα ανώτερα στρώματα συνδέουν αυτές τις πύλες με καλώδια. Ένας τρόπος για τη δημιουργία αυτών των στρωμάτων είναι με την εναπόθεση φωτοευαίσθητων χημικών πάνω στην επιφάνεια του chip και με τη μετέπειτα ρίψη φωτός μέσω μασκών για την επεξεργασία των χημικών. Η διαδικασία της δημιουργίας των στρωμάτων βασίζεται, συνεπώς, σε αυτή του σχεδιασμού των κατάλληλων μασκών. Ένα σύνολο από μάσκες συχνά αποκαλείται layout. Για κάθε τεχνολογία ολοκληρωμένων κυκλωμάτων όλα τα στρώματα πρέπει τελικά να κατασκευαστούν για τη δημιουργία ενός λειτουργικού ολοκληρωμένου κυκλώματος. Το θέμα είναι ποιος και πότε κατασκευάζει το κάθε στρώμα και αυτά τα θέματα αναλύονται παρακάτω.

2.2.2.1 Full-custom/VLSI

Σε μία full-custom τεχνολογία ολοκληρωμένων κυκλωμάτων βελτιστοποιούνται όλα τα στρώματα για την ψηφιακή υλοποίηση του συγκεκριμένου ενσωματωμένου συστήματος. Μία τέτοια βελτιστοποίηση περιλαμβάνει την τοποθέτηση των transistors για την ελαχιστοποίηση των μηκών των διασυνδέσεων, τη ρύθμιση του μεγέθους των transistors για τη βελτιστοποίηση των μεταδόσεων των σημάτων και τη δρομολόγηση των καλωδίων ανάμεσα στα transistors. Μετά την ολοκλήρωση του σχεδιασμού των μασκών γίνεται η αποστολή των προδιαγραφών αυτών στην εργοστασιακή μονάδα που κατασκευάζει τα ολοκληρωμένα κυκλώματα. Η full-custom σχεδίαση ολοκληρωμένων κυκλωμάτων, που συχνά αναφέρεται ως σχεδίαση VLSI (Very Large Scale of Integration), έχει πολύ υψηλό κόστος σχεδίασης και συνοδεύεται από μεγάλο χρονικό διάστημα μέχρι το ολοκληρωμένο κύκλωμα να γίνει διαθέσιμο. Μπορεί, ωστόσο, να έχει εξαιρετική απόδοση με μικρό μέγεθος και χαμηλή κατανάλωση ισχύος. Χρησιμοποιείται συνήθως σε εφαρμογές μεγάλων ποσοτήτων ή σε εφαρμογές κρίσιμες σε απόδοση.

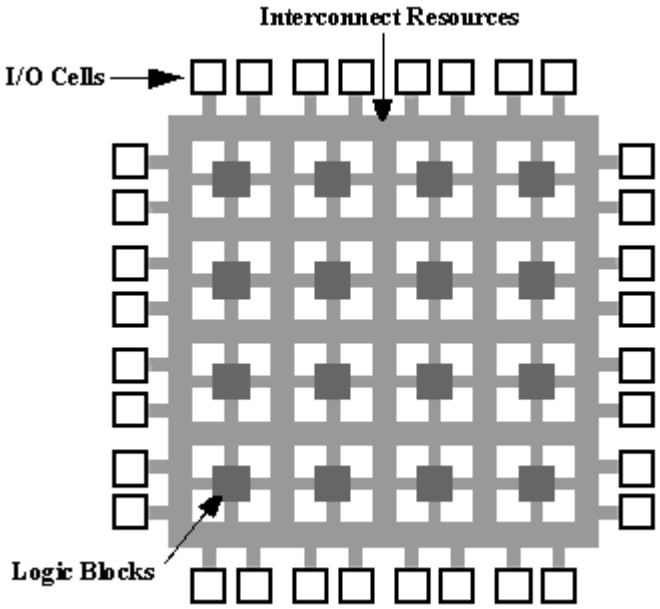
2.2.2.2 Semi-custom/ASIC

Σε μία τεχνολογία ASIC (Application-Specific Integrated Circuit) τα χαμηλότερα στρώματα κατασκευάζονται πλήρως ή μερικώς, αφήνοντας τα ανώτερα στρώματα να ολοκληρωθούν αργότερα. Σε μία τεχνολογία gate array οι μάσκες για τα επίπεδα των transistors και των πυλών έχουν ήδη κατασκευαστεί, δηλαδή το ολοκληρωμένο κύκλωμα ήδη αποτελείται από συστοιχίες πυλών. Η διεργασία που απομένει είναι η σύνδεση αυτών των πυλών για την επίτευξη της συγκεκριμένης υλοποίησης. Σε μία τεχνολογία standard cell τα κύτταρα του λογικού επιπέδου, όπως μία πύλη ΚΑΙ ή ένας συνδυασμός ΚΑΙ-Ή-ΑΝΤΙΣΤΡΟΦΗ, έχουν τα τμήματα των μασκών τους προσχεδιασμένα. Η διεργασία που απομένει είναι η οργάνωση αυτών των τμημάτων σε ολοκληρωμένες μάσκες για το επίπεδο πύλης και η μετέπειτα σύνδεση των κυττάρων. Τα ASICs είναι με διαφορά η πιο δημοφιλής τεχνολογία ολοκληρωμένων κυκλωμάτων, καθώς παρέχουν καλή απόδοση και μέγεθος με πολύ μικρότερο κόστος σχεδίασης από τα full-custom ολοκληρωμένα κυκλώματα.

2.2.2.3 PLD

Σε μία τεχνολογία PLD (Programmable Logic Device) όλα τα στρώματα υπάρχουν ήδη, οπότε είναι δυνατή η αγορά του πραγματικού ολοκληρωμένου κυκλώματος. Τα στρώματα υλοποιούν ένα προγραμματιζόμενο κύκλωμα, όπου ο προγραμματισμός έχει σημασία χαμηλότερου επιπέδου από ένα πρόγραμμα λογισμικού. Ο προγραμματισμός που λαμβάνει χώρα μπορεί να αποτελείται από τη δημιουργία ή την καταστροφή συνδέσεων ανάμεσα σε καλώδια που συνδέουν πύλες, είτε καίγοντας μία ασφάλεια είτε θέτοντας ένα bit σε ένα προγραμματιζόμενο διακόπτη. Μικρές συσκευές, που ονομάζονται προγραμματιστές, συνδεδεμένες σε ένα προσωπικό υπολογιστή μπορούν να επιτελέσουν αυτόν τον προγραμματισμό. Τα PLDs μπορούν να διαιρεθούν σε δύο τύπους, τα απλά και τα σύνθετα. Ένας τύπος απλού PLD είναι ένα PLA (Programmable Logic Array), το οποίο αποτελείται από μία προγραμματιζόμενη συστοιχία από πύλες ΚΑΙ και μία προγραμματιζόμενη συστοιχία από πύλες Ή. Άλλος ένας τύπος είναι ένα PAL (Programmable Array of Logic), το οποίο χρησιμοποιεί μία μόνο προγραμματιζόμενη συστοιχία για τη μείωση του αριθμού των ακριβών προγραμματιζόμενων συνιστωσών. Ένας τύπος σύνθετου PLD, που γίνεται όλο και πιο δημοφιλής, είναι το FPGA (Field Programmable Gate Array), το οποίο παρέχει πολύ γενικότερη συνδεσιμότητα ανάμεσα σε blocks λογικής από τις απλές συστοιχίες λογικής των PLAs και των PALs. Είναι, με αυτόν τον τρόπο, δυνατή η υλοποίηση πολύ πιο σύνθετων κατασκευών. Η δομή ενός FPGA φαίνεται στο Σχήμα 2.7. Τα PLDs προσφέρουν πολύ χαμηλό κόστος σχεδίασης και σχεδόν άμεση διαθεσιμότητα ολοκληρωμένων κυκλωμάτων. Είναι, ωστόσο, μεγαλύτερα συνήθως από τα ASICs, μπορεί να έχουν υψηλότερο κόστος

ανά μονάδα, να καταναλώνουν περισσότερη ισχύ και να είναι πιο αργά, ειδικά τα FPGAs. Παρέχουν, παρόλα αυτά, πολύ ικανοποιητική απόδοση και ενδείκνυνται για ταχύτατη κατασκευή πρωτοτύπου.



Σχήμα 2.7: Δομή ενός FPGA

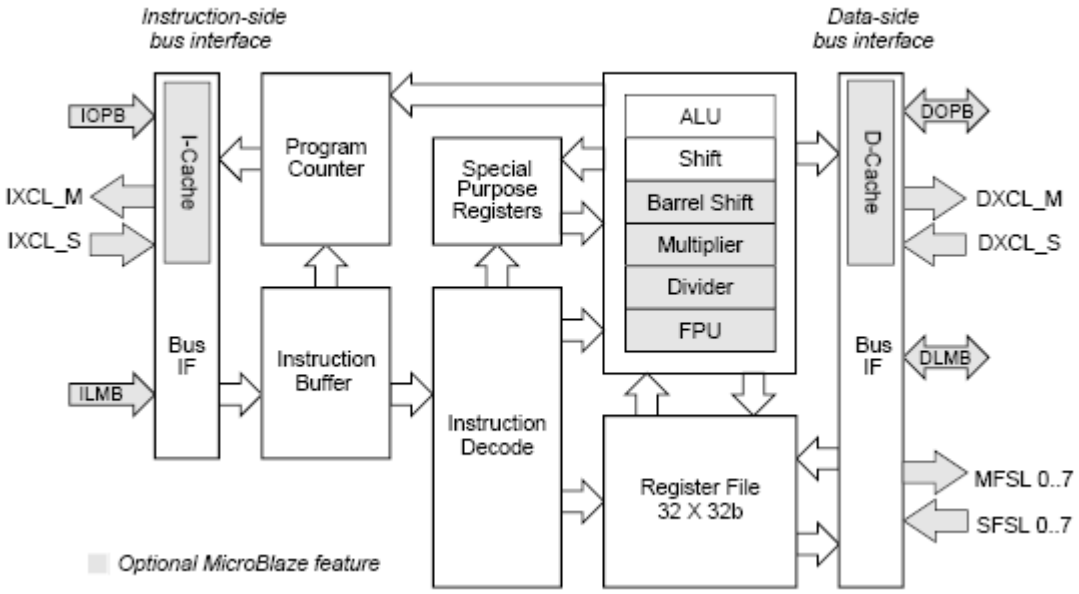
Κεφάλαιο 3

Αρχιτεκτονική του Υλικού

Στις πλακέτες της εταιρείας Xilinx μπορούν να προγραμματιστούν με τη χρήση του Xilinx Platform Studio ποικίλες συνιστώσες υλικού, είτε αυτές είναι υλοποιημένες από το χρήστη του αναπτυξιακού περιβάλλοντος σε κάποια γλώσσα περιγραφής υλικού, όπως σε VHDL ή Verilog, είτε διατίθενται από την ίδια την εταιρεία. Σε αυτό το κεφάλαιο περιγράφονται οι συνιστώσες υλικού που διαθέτει η Xilinx και οι οποίες θα χρησιμοποιηθούν στην παρούσα εργασία. Αυτό που ενδιαφέρει είναι η κατανόηση της λειτουργίας τους με σκοπό τη χρήση τους σε ενσωματωμένα συστήματα. Θα περιγραφούν κατά σειρά ο επεξεργαστής MicroBlaze Processor, ο ελεγκτής διακοπών OPB Interrupt Controller, ο χρονιστής/μετρητής OPB Timer/Counter και η σειριακή θύρα OPB UART Lite.

3.1 MicroBlaze Processor

Ο MicroBlaze είναι ένας RISC (Reduced Instruction Set Computer) επεξεργαστής που μπορεί να χρησιμοποιηθεί στα FPGAs της εταιρείας Xilinx. Πολλές παράμετροι του MicroBlaze μπορούν να ρυθμιστούν ανάλογα με τις ανάγκες του κάθε συστήματος. Τα σταθερά χαρακτηριστικά του MicroBlaze περιέχουν, ωστόσο, τριάντα δύο 32-bit καταχωρητές γενικού σκοπού, 32-bit εντολές με τρία ορίσματα και δύο τρόπους αναφοράς στη μνήμη, 32-bit διάδρομο διευθύνσεων και σωλήνωση που υποστηρίζει την εκτέλεση μίας εντολής κάθε φορά (single issue pipeline). Το διάγραμμα του MicroBlaze φαίνεται στο Σχήμα 3.1.



Σχήμα 3.1: Διάγραμμα του MicroBlaze

3.1.1 Τύποι Δεδομένων και Endianness

Ο MicroBlaze χρησιμοποιεί τον big-endian, bit-reversed τρόπο αναπαράστασης δεδομένων. Υποστηρίζει τους τύπους δεδομένων word, half word και byte, που ισούνται αντίστοιχα με 4, 2 και 1 byte. Για την κατανόηση των παραπάνω, στο Σχήμα 3.2 φαίνεται ο τρόπος αναπαράστασης του τύπου half word.

Byte address	n	n+1
Byte label	0	1
Byte significance	MSByte	LSByte
Bit label	0	15
Bit significance	MSBit	LSBit

Σχήμα 3.2: Τύπος Δεδομένων Half Word

3.1.2 Εντολές

Όλες οι εντολές του MicroBlaze είναι 32-bit και η καθεμία είναι είτε Τύπου A είτε Τύπου B. Οι

εντολές Τύπου Α έχουν μέχρι και δύο καταχωρητές πηγές και έναν καταχωρητή προορισμού ως ορίσματα. Οι εντολές Τύπου Β έχουν έναν καταχωρητή πηγή, μία 16-bit άμεση τιμή (που μπορεί να επεκταθεί στα 32 bit με την τοποθέτηση μίας εντολής IMM, δηλαδή Immediate, πριν από την εντολή Τύπου Β) και έναν καταχωρητή προορισμού ως ορίσματα. Οι εντολές ανήκουν στις επόμενες λειτουργικές κατηγορίες: αριθμητικές, λογικές, διακλάδωσης, φόρτωσης/αποθήκευσης και ειδικές.

3.1.3 Καταχωρητές

3.1.3.1 Καταχωρητές Γενικού Σκοπού

Ο MicroBlaze έχει τριάντα δύο 32-bit καταχωρητές γενικού σκοπού. Οι καταχωρητές αυτοί αριθμούνται R0 μέχρι και R31 και μηδενίζονται κατά το bitstream download. Δε μηδενίζονται, ωστόσο, από τις εξωτερικές εισόδους reset και debug_rst. Ο R0 έχει την τιμή 0 πάντοτε. Οι R1 έως και R13, καθώς και οι R18 έως και R31 είναι καταχωρητές γενικού σκοπού. Το ίδιο και ο R15. Αν ο MicroBlaze δεν έχει ρυθμιστεί να υποστηρίζει εξαιρέσεις υλικού (Hardware Exceptions), τότε και ο R17 είναι καταχωρητής γενικού σκοπού. Διαφορετικά φορτώνεται σε αυτόν η διεύθυνση επιστροφής από την εξαίρεση υλικού. Στον καταχωρητή R14 φορτώνεται η διεύθυνση επιστροφής από διακοπή (interrupt) και στον R16 από break. Τα παραπάνω φαίνονται στο Σχήμα 3.3.

Bits	Name	Description	Reset Value
0:31	R0	R0 is defined to always have the value of zero. Anything written to R0 is discarded.	0x00000000
0:31	R1 through R13	R1 through R13 are 32-bit general purpose registers	-
0:31	R14	32-bit used to store return addresses for interrupts	-
0:31	R15	32-bit general purpose register	-
0:31	R16	32-bit used to store return addresses for breaks	-
0:31	R17	If MicroBlaze is configured to support hardware exceptions, this register is loaded with HW exception return address; if not it is a general purpose register	-
0:31	R18 through R31	R18 through R31 are 32-bit general purpose registers.	-

Σχήμα 3.3: Καταχωρητές Γενικού Σκοπού

3.1.3.2 Καταχωρητές Ειδικού Σκοπού

Υπάρχουν, επίσης, πέντε 32-bit καταχωρητές ειδικού σκοπού και οι οποίοι περιγράφονται αμέσως παρακάτω. Ο PC (Program Counter) περιέχει την 32-bit διεύθυνση της εντολής που εκτελείται. Μπορεί να διαβαστεί με μία MFS (Move From Special) εντολή. Δεν μπορεί να γραφτεί με μία MTS (Move To Special) εντολή.

Ο MSR (Machine Status Register) περιέχει bits σχετικά με τον έλεγχο και την κατάσταση του επεξεργαστή. Μπορεί να διαβαστεί με μία MFS εντολή. Με την ανάγνωση του MSR το bit 29 αντιγράφεται στο bit 0 σαν το αντίγραφο του προσήμου. Ο MSR μπορεί να γραφτεί με τη χρήση μίας εντολής MTS ή με τις εξειδικευμένες εντολές MSRSET και MSRCLR. Όταν γίνεται εγγραφή στον MSR, κάποια από τα bits αλλάζουν αμέσως και τα υπόλοιπα αλλάζουν ένα κύκλο ρολογιού αργότερα. Κάθε τιμή που γράφεται στο bit 0 απορρίπτεται. Από το λιγότερο σημαντικό προς τα περισσότερο σημαντικά τα bits έχουν την εξής σημασία:

1. bit 31, BE. Όταν έχει την τιμή 0, τότε το κλείδωμα διαδρόμου είναι απενεργοποιημένο στο On-chip Peripheral Bus της πλευράς δεδομένων. Όταν έχει την τιμή 1, τότε το κλείδωμα διαδρόμου είναι ενεργοποιημένο στο On-chip Peripheral Bus της πλευράς δεδομένων.
2. bit 30, IE. Όταν έχει την τιμή 0, τότε οι διακοπές είναι απενεργοποιημένες. Όταν έχει την τιμή 1, τότε οι διακοπές είναι ενεργοποιημένες.
3. bit 29, C. Όταν έχει την τιμή 0, τότε δεν υπάρχει κρατούμενο ή υπάρχει δανειζόμενο. Όταν έχει την τιμή 1, τότε υπάρχει κρατούμενο ή δεν υπάρχει δανειζόμενο.
4. bit 28, BIP. Όταν έχει την τιμή 0, τότε δεν υπάρχει break σε εξέλιξη. Όταν έχει την τιμή 1, τότε υπάρχει break σε εξέλιξη.
5. bit 27, FSL. Όταν έχει την τιμή 0, τότε η FSL get/put δεν είχε σφάλμα. Όταν έχει την τιμή 1, τότε η FSL get/put είχε σφάλμα μη-ταιριάσματος του τύπου εντολής με τον τύπο τιμής.
6. bit 26, ICE. Όταν έχει την τιμή 0, τότε η cache εντολών είναι απενεργοποιημένη. Όταν έχει την τιμή 1, τότε η cache εντολών είναι ενεργοποιημένη.
7. bit 25, DZ. Όταν έχει την τιμή 0, τότε δεν έχει γίνει ακέραιη διαίρεση με το μηδέν. Όταν έχει την τιμή 1, τότε έχει γίνει ακέραιη διαίρεση με το μηδέν.
8. bit 24, DCE. Όταν έχει την τιμή 0, τότε η cache δεδομένων είναι απενεργοποιημένη. Όταν έχει την τιμή 1, τότε η cache δεδομένων είναι ενεργοποιημένη.
9. bit 23, EE. Όταν έχει την τιμή 0, τότε οι εξαιρέσεις υλικού είναι απενεργοποιημένες. Όταν έχει

την τιμή 1, τότε οι εξαιρέσεις υλικού είναι ενεργοποιημένες.

10. bit 22, EIP. Όταν έχει την τιμή 0, τότε δεν υπάρχει εξαίρεση υλικού σε εξέλιξη. Όταν έχει την τιμή 1, τότε υπάρχει εξαίρεση υλικού σε εξέλιξη.
11. bits 1 – 21. Δεσμευμένα.
12. bit 0, CC. Πρόκειται για αντίγραφο του bit 29, του C.

Ο EAR (Exception Address Register) αποθηκεύει τη διεύθυνση φόρτωσης/αποθήκευσης που προκάλεσε την εξαίρεση. Για μία εξαίρεση μη-ευθυγραμμισμένης πρόσβασης αυτό σημαίνει τη μη ευθυγραμμισμένη διεύθυνση πρόσβασης και για μία εξαίρεση του On-chip Peripheral Bus δεδομένων τη διεύθυνση πρόσβασης στο On-chip Peripheral Bus δεδομένων που αποτυγχάνει. Τα περιεχόμενα του καταχωρητή αυτού είναι ακαθόριστα για όλες τις άλλες εξαιρέσεις.

Ο ESR (Exception Status Register) περιέχει bits κατάστασης του επεξεργαστή που αφορούν στις εξαιρέσεις. Από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά τα bits έχουν την εξής σημασία:

1. bits 27 – 31, EC. Πρόκειται για το αίτιο της εξαίρεσης. Συγκεκριμένα το πεδίο λαμβάνει τις εξής τιμές:

    1. 00001. Εξαίρεση μη-ευθυγραμμισμένης πρόσβασης δεδομένων.
    2. 00010. Εξαίρεση παράνομου κώδικα εντολής.
    3. 00011. Εξαίρεση σφάλματος διαδρόμου εντολών.
    4. 00100. Εξαίρεση σφάλματος διαδρόμου δεδομένων.
    5. 00101. Εξαίρεση διαίρεσης με το μηδέν.
    6. 00110. Εξαίρεση μονάδας κινητής υποδιαστολής.

2. bits 20 – 26, ESS. Πρόκειται για την ειδική κατάσταση της εξαίρεσης. Για την περίπτωση μη-ευθυγραμμισμένης πρόσβασης δεδομένων το bit 20, W, έχει την τιμή 0, όταν πρόκειται για μη-ευθυγραμμισμένη πρόσβαση μισής λέξης, και 1, όταν πρόκειται για μη-ευθυγραμμισμένη πρόσβαση λέξης. Το bit 21, S, έχει την τιμή 0, όταν πρόκειται για μη-ευθυγραμμισμένη πρόσβαση φόρτωσης, και 1, όταν πρόκειται για μη-ευθυγραμμισμένη πρόσβαση αποθήκευσης. Τα bits 22 – 26 δείχνουν τον καταχωρητή πηγή για την περίπτωση αποθήκευσης ή τον καταχωρητή προορισμού για την περίπτωση φόρτωσης.

3. bits 0 – 19. Δεσμευμένα.

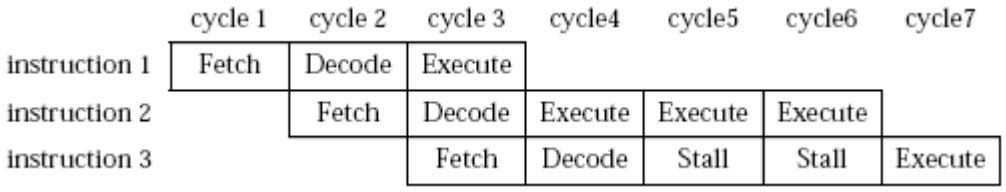
Ο FSR (Floating Point Status Register) περιέχει bits κατάστασης για τη μονάδα κινητής υποδιαστολής (Floating Point Unit). Μπορεί να διαβαστεί με μία εντολή MFS και να γραφτεί με μία εντολή MTS. Από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά τα bits έχουν την εξής σημασία:

1. bit 31, DO. Έχει την τιμή 1, όταν υπάρχει σφάλμα μη-κανονικοποιημένου τελεστέου. Διαφορετικά έχει την τιμή 0.
2. bit 30, UF. Έχει την τιμή 1, όταν υπάρχει σφάλμα υποχείλισης. Διαφορετικά έχει την τιμή 0.
3. bit 29, OF. Έχει την τιμή 1, όταν υπάρχει σφάλμα υπερχείλισης. Διαφορετικά έχει την τιμή 0.
4. bit 28, DZ. Έχει την τιμή 1, όταν υπάρχει σφάλμα διαίρεσης με το μηδέν. Διαφορετικά έχει την τιμή 0.
5. bit 27, IO. Έχει την τιμή 1, όταν υπάρχει σφάλμα άκυρης πράξης. Διαφορετικά έχει την τιμή 0.
6. bits 0 – 26. Δεσμευμένα.

3.1.4 Αρχιτεκτονική Σωλήνωσης

Ο MicroBlaze διαθέτει σωλήνωση (pipeline) τριών σταδίων για την εκτέλεση εντολών. Τα τρία στάδια είναι τα εξής: Ανάκληση (Fetch), Αποκωδικοποίηση (Decode) και Εκτέλεση (Execute).

Για τις περισσότερες εντολές κάθε στάδιο θέλει ένα κύκλο ρολογιού για να ολοκληρωθεί. Συνεπώς, χρειάζεται τρεις κύκλους ρολογιού μία συγκεκριμένη εντολή για να ολοκληρωθεί ενώ μία εντολή ολοκληρώνεται σε κάθε κύκλο. Λίγες εντολές χρειάζονται πολλαπλούς κύκλους ρολογιού στο στάδιο εκτέλεσης για να ολοκληρωθούν. Αυτό επιτυγχάνεται με την καθυστέρηση (stalling) της σωλήνωσης, όπως φαίνεται στο Σχήμα 3.4.



Σχήμα 3.4: Καθυστέρηση της Σωλήνωσης

3.1.4.1 Διακλαδώσεις

Κανονικά οι εντολές στα στάδια ανάκλησης και αποκωδικοποίησης απορρίπτονται, όταν γίνεται μία διακλάδωση. Το στάδιο ανάκλησης της σωλήνωσης φορτώνεται, τότε, με μία νέα εντολή από την υπολογισμένη διεύθυνση διακλάδωσης. Μία διακλάδωση (taken branch) του MicroBlaze απαιτεί τρεις κύκλους ρολογιού για να εκτελεστεί, δύο από τους οποίους απαιτούνται για το γέμισμα της σωλήνωσης. Για την καταπολέμηση αυτής της καθυστέρησης, ο MicroBlaze υποστηρίζει διακλαδώσεις με σχισμές καθυστέρησης (delay slots).

Κατά την εκτέλεση μίας διακλάδωσης με σχισμή καθυστέρησης μόνο το στάδιο ανάκλησης απορρίπτεται από τη σωλήνωση. Επιτρέπεται στην εντολή που βρίσκεται στο στάδιο αποκωδικοποίησης να εκτελεστεί. Αυτή η τεχνική μειώνει την πρόσθετη καθυστέρηση για διακλάδωση από δύο κύκλους σε έναν. Οι εντολές διακλάδωσης με σχισμές καθυστέρησης έχουν ένα D προσαρτημένο στο μνημονικό όνομα της εντολής.

Μία σχισμή καθυστέρησης δεν πρέπει να περιέχει τις ακόλουθες εντολές: IMM, branch ή break. Ούτε μπορεί να περιέχει εντολές που προκαλούν εξαιρέσεις που μπορούν να διορθωθούν (recoverable exceptions), μη-ευθυγράμμιση (unalignement) λόγου χάρη, όταν είναι ενεργοποιημένες οι εξαιρέσεις υλικού. Οι διακοπές και τα εξωτερικά breaks υλικού αναβάλλονται μέχρι μετά την ολοκλήρωση της διακλάδωσης με τη σχισμή καθυστέρησης.

3.1.5 Αρχιτεκτονική Μνήμης

Ο MicroBlaze έχει Harvard Αρχιτεκτονική μνήμης (Harvard Architecture), που σημαίνει ότι οι προσβάσεις στις εντολές και στα δεδομένα γίνονται σε ξεχωριστούς χώρους διευθύνσεων. Κάθε χώρος διευθύνσεων έχει εύρος 32 bit, δηλαδή χειρίζεται μέχρι και 4 GB μνήμης εντολών και δεδομένων. Το εύρος της μνήμης εντολών και το εύρος της μνήμης δεδομένων μπορούν να αλληλοεπικαλύπτονται με την απεικόνιση και των δύο στην ίδια φυσική μνήμη. Αυτό είναι χρήσιμο για την αποσφαλμάτωση του λογισμικού.

Οι διαπροσωπείες εντολών και δεδομένων του MicroBlaze είναι και οι δύο 32 bit σε πλάτος και χρησιμοποιούν τον big-endian και bit-reversed τρόπο αναπαράστασης δεδομένων. Ο MicroBlaze υποστηρίζει word, halfword και byte προσβάσεις στη μνήμη δεδομένων. Οι προσβάσεις δεδομένων πρέπει να είναι ευθυγραμμισμένες, δηλαδή οι word προσβάσεις πρέπει να είναι σε όρια λέξεων και οι halfword προσβάσεις πρέπει να είναι σε όρια μισών λέξεων, αν ο επεξεργαστής δεν είναι ρυθμισμένος

να υποστηρίζει μη-ευθυγραμμισμένες εξαιρέσεις. Όλες οι προσβάσεις εντολών πρέπει να είναι ευθυγραμμισμένες σε λέξεις.

Ο MicroBlaze δεν ξεχωρίζει τις προσβάσεις δεδομένων από τις προσβάσεις E/E, δηλαδή χρησιμοποιεί απεικονισμένη στη μνήμη (memory mapped) E/E. Ο επεξεργαστής έχει μέχρι και τρεις διαπροσωπείες για προσβάσεις στη μνήμη, το Local Memory Bus (LMB), το On-chip Peripheral Bus (OPB) και το Xilinx Cache Link (XCL). Οι απεικονίσεις στη μνήμη αυτών των διαπροσωπειών είναι αμοιβαία αποκλειόμενες.

Ο MicroBlaze έχει καθυστέρηση δύο κύκλων ρολογιού για τις προσβάσεις στη μνήμη του Local Memory Bus της πλευράς δεδομένων. Η ίδια καθυστέρηση υπάρχει και για τις επιτυχημένες προσβάσεις (hits) στην cache δεδομένων. Η μνήμη του Local Memory Bus και η cache της πλευράς εντολών χρησιμοποιούν αλληλοεπικαλυπτόμενες φάσεις διεύθυνσης και δεδομένων, ώστε να επιτύχουν καθυστέρηση ενός κύκλου.

Ο MicroBlaze χρησιμοποιεί υποθετικές προσβάσεις, για να μειώσει την καθυστέρηση των πιο αργών διαπροσωπειών μνήμης. Αυτό σημαίνει ότι ο επεξεργαστής θα ξεκινήσει κάθε πρόσβαση μνήμης σε όλες τις διαθέσιμες διαπροσωπείες. Όταν έχει βρεθεί η σωστή διαπροσωπεία, δηλαδή έχει ταιριάξει με την απεικόνιση διευθύνσεων της διαπροσωπείας, στον ακόλουθο κύκλο, οι άλλες προσβάσεις εγκαταλείπονται.

3.1.6 Reset, Διακοπές, Εξαιρέσεις και Breaks

Ο MicroBlaze υποστηρίζει reset, διακοπή, εξαίρεση χρήστη, break και εξαίρεση υλικού. Το ακόλουθο τμήμα περιγράφει τη ροή εκτέλεσης που σχετίζεται με καθένα από αυτά τα γεγονότα.

Η σχετική προτεραιότητα ξεκινώντας από τη μεγαλύτερη είναι:

1. Reset
2. Εξαίρεση Υλικού
3. Non-maskable Break
4. Break
5. Διακοπή
6. Διάνυσμα Χρήστη (Εξαίρεση)

Στο Σχήμα 3.5 φαίνονται οι διευθύνσεις θέσεων μνήμης των συσχετισμένων διανυσμάτων και

οι επιβεβλημένες από το υλικό θέσεις στο αρχείο καταχωρητών για τη διεύθυνση επιστροφής. Κάθε διάνυσμα καταλαμβάνει δύο διευθύνσεις, ώστε να επιτρέπει διακλάδωση σε ολόκληρο το εύρος διυθύνσεων (απαιτεί μία εντολή IMM ακολουθούμενη από μία εντολή BRAI). Το εύρος διευθύνσεων 0x28 έως 0x4F είναι δεσμευμένο για μελλοντική υποστήριξη λογισμικού από τη Xilinx.

Event	Vector Address	Register File Return Address
Reset	0x00000000 - 0x00000004	-
User Vector (Exception)	0x00000008 - 0x0000000C	-
Interrupt	0x00000010 - 0x00000014	R14
Break: Non-maskable hardware	0x00000018 - 0x0000001C	R16
Break: Hardware		
Break: Software		
Hardware Exception	0x00000020 - 0x00000024	R17
Reserved by Xilinx for future use	0x00000028 - 0x0000004F	-

Σχήμα 3.5: Διανύσματα και Διευθύνσεις Επιστροφής

3.1.6.1 Reset

Όταν συμβεί reset ή debug_rst (reset ελεγχόμενο από το Xilinx Microprocessor Debugger μέσω του Microprocessor Debug Module), ο MicroBlaze θα αδειάσει τη σωλήνωση και θα αρχίσει να ανακαλεί εντολές από το reset διάνυσμα (διεύθυνση 0x0). Και τα δύο εξωτερικά σήματα reset είναι ενεργά ψηλά (active high) και πρέπει να τοποθετηθούν για τουλάχιστον 16 κύκλους. Όταν αυτό συμβεί, τότε μηδενίζονται οι καταχωρητές Program Counter, Machine Status Register, Exception Address Register και Exception Status Register.

3.1.6.2 Εξαιρέσεις Υλικού

Ο MicroBlaze μπορεί να ρυθμιστεί, ώστε να συλλαμβάνει τις ακόλουθες συνθήκες εσωτερικού σφάλματος: παράνομη εντολή, σφάλμα διαδρόμου εντολών και δεδομένων και μη-ευθυγραμμισμένη πρόσβαση. Η εξαίρεση διαίρεσης με το μηδέν μπορεί μόνο να ενεργοποιηθεί, αν ο επεξεργαστής είναι

ρυθμισμένος με διαιρέτη υλικού (hardware divider). Όταν είναι ρυθμισμένος με μονάδα κινητής υποδιαστολής υλικού, μπορεί, επίσης, να συλλάβει τις ακόλουθες εξειδικευμένες εξαιρέσεις κινητής υποδιαστολής: υποχείλιση, υπερχείλιση, διαίρεση με μηδέν κινητής υποδιαστολής, άκυρη πράξη και σφάλμα μη-κανονικοποιημένου τελεστέου.

Μία εξαίρεση υλικού θα προκαλέσει στον MicroBlaze να αδειάσει τη σωλήνωση και να κάνει διακλάδωση στο διάνυσμα εξαιρέσεων υλικού (διεύθυνση 0x20). Η εξαίρεση θα φορτώσει, επίσης, την τιμή του μετρητή προγράμματος της εντολής του σταδίου της αποκωδικοποίησης στον καταχωρητή γενικού σκοπού R17. Η εντολή στο στάδιο εκτέλεσης στον κύκλο εξαίρεσης δεν εκτελείται. Τα bits EE και EIP του Machine Status Register αυτόματα επανέρχονται με την εκτέλεση της εντολής (RTED). Μία εξαίρεση στη σχισμή καθυστέρησης μίας εντολής διακλάδωσης δεν μπορεί να διορθωθεί (είναι non-recoverable). Για αυτό το λόγο ο μεταγλωττιστής του MicroBlaze δε θα τοποθετήσει ποτέ μία εντολή φόρτωσης/αποθήκευσης ή μία εντολή κινητής υποδιαστολής σε μία σχισμή καθυστέρησης, επειδή αυτές απαιτούν εξαιρέσεις που μπορούν να διορθωθούν (μη-ευθυγραμμισμένη πρόσβαση και υποχείλιση λόγου χάρη).

Αίτια Εξαιρέσεων

Τα αίτια εξαιρέσεων είναι τα εξής:

1. Εξαίρεση Διαδρόμου Εντολών. Η εξαίρεση του On-chip Peripheral Bus εντολών προκαλείται από ένα σήμα error acknowledge ή timeout error στο On-chip Peripheral Bus εντολών.
2. Εξαίρεση Παράνομου Κώδικα Εντολής. Η εξαίρεση παράνομου κώδικα εντολής προκαλείται από μία εντολή με άκυρο κύριο κώδικα εντολής (bits 0 – 5 της εντολής). Το υπόλοιπο της εντολής δεν ελέγχεται. Εντολές συσχετισμένες με επιλογές του επεξεργαστή ανιχνεύονται ως παράνομες, αν δεν είναι ενεργοποιημένη η επιλογή.
3. Εξαίρεση Διαδρόμου Δεδομένων. Η εξαίρεση του On-chip Peripheral Bus δεδομένων προκαλείται από ένα σήμα error acknowledge ή timeout error στο On-chip Peripheral Bus δεδομένων.
4. Μη-ευθυγραμμισμένη Εξαίρεση. Η μη-ευθυγραμμισμένη εξαίρεση προκαλείται από μία πρόσβαση λέξης με τη διεύθυνση στο διάδρομο δεδομένων να έχει τα bits 30 ή 31 με την τιμή 1 ή με μία πρόσβαση μισής λέξης με το bit 31 να έχει την τιμή 1.
5. Εξαίρεση Διαίρεσης με το Μηδέν. Η εξαίρεση διαίρεσης με το μηδέν προκαλείται από μία

ακέραιη διαίρεση (IDIV ή IDIVU) με το διαιρέτη να είναι μηδέν.

6. Εξαίρεση Μονάδας Κινητής Υποδιαστολής. Μία εξαίρεση μονάδας κινητής υποδιαστολής προκαλείται από υποχείλιση, υπερχείλιση, διαίρεση με το μηδέν, παράνομη πράξη ή μη-κανονικοποιημένο τελεστέο σε μία εντολή κινητής υποδιαστολής. Η υποχείλιση σημαίνει μη-κανονικοποιημένο αποτέλεσμα, η υπερχείλιση σημαίνει αποτέλεσμα που δεν είναι αριθμός (Not-a-Number αποτέλεσμα), η εξαίρεση διαίρεσης με το μηδέν κινητής υποδιαστολής προκαλείται όταν ο τελεστέος RA στην εντολή FDIV είναι μηδέν και ο RB δεν είναι άπειρος και η εξαίρεση παράνομης πράξης προκαλείται από ένα τελεστέο που δεν είναι αριθμός ή από συνδυασμούς παράνομου απείρου ή μηδενός.

Όταν συμβεί εξαίρεση, ο Program Counter θα φορτωθεί στον καταχωρητή R17 και η τιμή 0x20 θα φορτωθεί στον Program Counter. Στον Machine Status Register το bit Exception Enable θα γίνει 0 και το bit Exception In Progress θα γίνει 1. Στον Exception Status Register τα τμήματα Exception Cause και Exception Specific Status θα ενημερωθούν με τις συγκεκριμένες τιμές της εξαίρεσης, όπως και οι καταχωρητές Exception Address Register και Floating Point Status Register.

3.1.6.3 Breaks

Υπάρχουν δύο ειδών breaks, τα breaks υλικού (εξωτερικά) και τα breaks λογισμικού (εσωτερικά). Τα breaks υλικού συμβαίνουν με την τοποθέτηση των εξωτερικών σημάτων break. Σε ένα break η εντολή στο στάδιο εκτέλεσης θα ολοκληρωθεί ενώ η εντολή στο στάδιο αποκωδικοποίησης θα αντικατασταθεί με μία διακλάδωση στο διάνυσμα του break (διεύθυνση 0x18). Η διεύθυνση επιστροφής του break (ο Program Counter που συσχετίζεται με την εντολή στο στάδιο αποκωδικοποίησης τη στιγμή του break) αυτόματα φορτώνεται στον καταχωρητή γενικού σκοπού R16. Ο MicroBlaze θέτει, επίσης, τη σημαία Break In Progress στο Machine Status Register. Ένα κανονικό break υλικού θα εξυπηρετηθεί μόνο όταν δεν υπάρχει break σε εξέλιξη. Η σημαία Break In Progress απενεργοποιεί τις διακοπές. Ένα non-maskable break θα εξυπηρετηθεί πάντα αμέσως. Το bit BIP στον Machine Status Register αυτόματα καθαρίζεται όταν εκτελείται η εντολή RTBD. Για την εκτέλεση ενός break λογισμικού χρησιμοποιούνται οι εντολές brk και brki. Ο χρόνος που χρειάζεται ο MicroBlaze για να μπει στη ρουτίνα εξυπηρέτησης του break από την ώρα που συμβαίνει το break εξαρτάται από την εντολή που βρίσκεται την τρέχουσα στιγμή στο στάδιο εκτέλεσης.

3.1.6.4 Διακοπή

Ο MicroBlaze υποστηρίζει μία εξωτερική πηγή διακοπής. Ο επεξεργαστής θα αντιδράσει μόνο σε διακοπές, όταν το bit Interrupt Enable του Machine Status Register έχει την τιμή 1. Σε μία διακοπή η εντολή στο στάδιο εκτέλεσης θα ολοκληρωθεί ενώ η εντολή στο στάδιο αποκωδικοποίησης θα αντικατασταθεί από μία διακλάδωση στο διάνυσμα διακοπών (διεύθυνση 0x10). Η διεύθυνση επιστροφής από τη διακοπή (ο Program Counter που συσχετίζεται με την εντολή στο στάδιο αποκωδικοποίησης τη στιγμή της διακοπής) αυτόματα φορτώνεται στον καταχωρητή γενικού σκοπού R14. Επιπροσθέτως, ο επεξεργαστής απενεργοποιεί μελλοντικές διακοπές καθαρίζοντας το bit Interrupt Enable στον Machine Status Register. Το bit IE τίθεται αυτόματα ξανά όταν εκτελείται η εντολή RTID. Οι διακοπές αγνοούνται από τον επεξεργαστή αν το bit Break In Progress στον Machine Status Register είναι 1. Ο χρόνος που θα χρειαστεί ο MicroBlaze για να μπει στη ρουτίνα εξυπηρέτησης διακοπής από τη στιγμή που συμβαίνει η διακοπή εξαρτάται από τη ρύθμιση του επεξεργαστή. Αν ο MicroBlaze έχει ρυθμιστεί να έχει διαιρέτη υλικού, η μεγαλύτερη καθυστέρηση θα συμβεί όταν μία διακοπή συμβαίνει κατά την εκτέλεση μίας εντολής διαίρεσης.

3.1.6.5 Διάνυσμα Χρήστη (Εξαίρεση)

Το διάνυσμα εξαιρέσεων χρήστη βρίσκεται στη διεύθυνση 0x8. Μία εξαίρεση χρήστη προκαλείται με την εισαγωγή μίας εντολής “BRALID Rx, 0x8” στη ροή του προγράμματος. Παρόλο που ο Rx μπορεί να είναι οποιοσδήποτε καταχωρητής γενικού σκοπού, η Xilinx προτείνει τη χρήση του R15 για την αποθήκευση της διεύθυνσης επιστροφής της εξαίρεσης χρήστη και τη χρήση της εντολής RTSD για την επιστροφή από τη ρουτίνα εξυπηρέτησης εξαίρεσης χρήστη. Όταν συμβεί εξαίρεση χρήστη, ο Program Counter φορτώνεται στον καταχωρητή Rx και η τιμή 0x8 φορτώνεται στον Program Counter.

3.1.7 Μονάδα Κινητής Υποδιαστολής

Η μονάδα κινητής υποδιαστολής του MicroBlaze βασίζεται στο πρότυπο IEEE 754. Χρησιμοποιεί ένα τρόπο αναπαράστασης απλής ακριβείας, που περιέχει ορισμούς για το άπειρο για το μη-αριθμό (NaN) και το μηδέν. Υποστηρίζει εντολές για πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση και σύγκριση. Υλοποιεί τον τρόπο λειτουργίας στρογγυλοποίησης-στον-κοντινότερο (round-

to-nearest). Παράγει bits κατάστασης για υποχείλιση, υπερχείλιση και άκυρη πράξη. Για βελτιωμένη απόδοση έχουν γίνει οι ακόλουθες μη-προτυποποιημένες απλοποιήσεις:

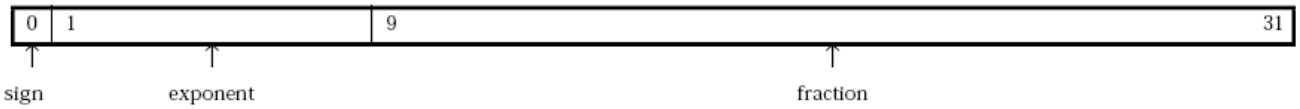
1. Μη-κανονικοποιημένοι τελεστέοι δεν υποστηρίζονται. Μία πράξη μονάδας κινητής υποδιαστολής υλικού πάνω σε ένα μη-κανονικοποιημένο αριθμό θα επιστρέψει ένα NaN και θα θέσει το bit σφάλματος μη-κανονικοποιημένου τελεστέου στον Floating Point Status Register.
2. Ένα μη-κανονικοποιημένο αποτέλεσμα αποθηκεύεται ως ένα προσημασμένο 0 με το bit υποχείλισης να έχει τεθεί στον FSR. Αυτή η μέθοδος συχνά αναφέρεται ως άδειασμα-στο-μηδέν (flush-to-zero).
3. Μία πράξη πάνω σε ένα NaN θα επιστρέψει το προκαθορισμένο NaN 0xFFC00000 αντί για έναν από τους NaN τελεστέους.
4. Η υπερχείλιση ως αποτέλεσμα μίας πράξης κινητής υποδιαστολής θα επιστρέψει πάντα προσημασμένο άπειρο, ακόμα και όταν συλλαμβάνεται η εξαίρεση.

3.1.7.1 Τρόπος Αναπαράστασης

Ένας IEEE 754 αριθμός κινητής υποδιαστολής απλής ακριβείας συντίθεται από τα ακόλουθα τρία πεδία:

1. 1-bit πρόσημο
2. 8-bit πολωμένος εκθέτης
3. 23-bit συντελεστής

Τα πεδία αποθηκεύονται σε μία 32-bit λέξη, όπως φαίνεται στο Σχήμα 3.6.



Σχήμα 3.6: IEEE 754 Τρόπος Αναπαράστασης Απλής Ακριβείας

Η τιμή ενός αριθμού κινητής υποδιαστολής v στο MicroBlaze έχει την ακόλουθη ερμηνεία:

1. Αν ο εκθέτης έχει την τιμή 255 και ο συντελεστής δεν είναι 0, τότε v = NaN, ανεξάρτητα από το bit προσήμου.
2. Αν ο εκθέτης έχει την τιμή 255 και ο συντελεστής είναι 0, τότε $v = (-1)^{sign} \cdot \infty$.
3. Αν 0 < εκθέτης < 255, τότε $v = (-1)^{sign} \cdot 2^{(\text{εκθέτης} - 127)} \cdot (1.\text{συντελεστής})$.
4. Αν ο εκθέτης είναι 0 και ο συντελεστής δεν είναι 0, τότε $v = (-1)^{sign} \cdot 2^{-126} \cdot (0.\text{συντελεστής})$.
5. Αν ο εκθέτης είναι 0 και ο συντελεστής είναι 0, τότε $v = (-1)^{sign} \cdot 0$.

3.1.7.2 Στρογγυλοποίηση

Η μονάδα κινητής υποδιαστολής του MicroBlaze υλοποιεί μόνο τον προεπιλεγμένο τρόπο στρογγυλοποίησης, τη στρογγυλοποίηση-στον-κοντινότερο, όπως ορίζεται στο IEEE 754. Εξ ορισμού, το αποτέλεσμα κάθε πράξης κινητής υποδιαστολής πρέπει να επιστρέφει τον κοντινότερο στο απείρως ακριβές αποτέλεσμα αριθμό απλής ακριβείας. Αν δύο τιμές είναι το ίδιο κοντά, τότε επιστρέφεται εκείνη που έχει το λιγότερο σημαντικό bit της 0.

3.1.7.3 Πράξεις

Όλες οι πράξεις της μονάδας κινητής υποδιαστολής του επεξεργαστή MicroBlaze χρησιμοποιούν τους καταχωρητές γενικού σκοπού του επεξεργαστή και όχι ένα εξειδικευμένο αρχείο καταχωρητών κινητής υποδιαστολής. Οι αριθμητικές πράξεις της μονάδας κινητής υποδιαστολής είναι πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση. Οι πράξεις σύγκρισης που υλοποιεί η μονάδα κινητής υποδιαστολής είναι μικρότερο-από, ίσο, μικρότερο-ή-ίσο, μεγαλύτερο-από, όχι-ίσο, μεγαλύτερο-ή-ίσο και μη-διατεταγμένη σύγκριση για NaN.

3.1.7.4 Εξαιρέσεις

Η μονάδα κινητής υποδιαστολής χρησιμοποιεί τον κανονικό μηχανισμό εξαιρέσεων υλικού στο MicroBlaze. Όταν είναι ενεργοποιημένες, οι εξαιρέσεις συμβαίνουν για όλες τις σύμφωνες με το πρότυπο IEEE συνθήκες: υποχείλιση, υπερχείλιση, διαίρεση με το μηδέν και παράνομη πράξη, όπως και για την ειδική εξαίρεση του MicroBlaze: σφάλμα μη-κανονικοποιημένου τελεστέου. Μία εξαίρεση κινητής υποδιαστολής θα εμποδίσει την εγγραφή στον καταχωρητή προορισμού. Αυτό επιτρέπει στη ρουτίνα εξυπηρέτησης της εξαίρεσης κινητής υποδιαστολής να διερευνήσει το απείραχτο αρχείο

καταχωρητών.

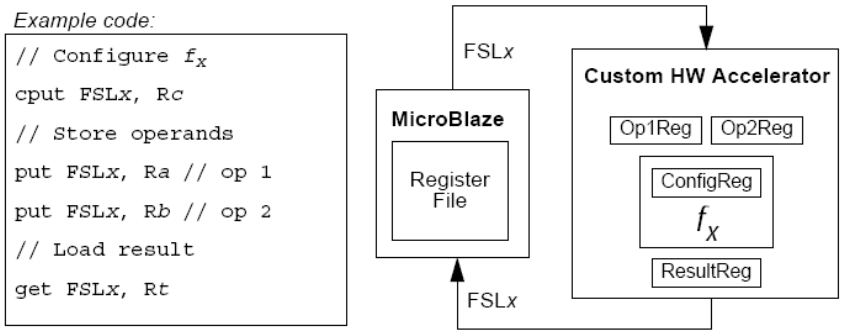
3.1.8 Fast Simplex Link

Ο MicroBlaze μπορεί να ρυθμιστεί να έχει έως και οκτώ διαπροσωπείες Fast Simplex Link (FSL), κάθε μία εκ των οποίων αποτελείται από μία θύρα εισόδου και μία θύρα εξόδου. Τα κανάλια του FSL είναι εξειδικευμένες διαπροσωπείες ρεύματος δεδομένων μονής κατεύθυνσης και σημείου προς σημείο.

Οι διαπροσωπείες FSL του MicroBlaze είναι πλάτους 32 bit. Ένα ξεχωριστό bit δείχνει το αν η λέξη που στέλνεται ή λαμβάνεται είναι τύπου ελέγχου ή τύπου δεδομένων. Η εντολή get του συνόλου εντολών του MicroBlaze χρησιμοποιείται για τη μεταφορά πληροφορίας από μία θύρα FSL σε έναν καταχωρητή γενικού σκοπού. Η εντολή put χρησιμοποιείται για τη μεταφορά κατά την αντίθετη κατεύθυνση. Και οι δύο εντολές έχουν από 4 εκδόσεις: μπλοκαριζόμενα δεδομένα, μη-μπλοκαριζόμενα δεδομένα, μπλοκαριζόμενος έλεγχος και μη-μπλοκαριζόμενος έλεγχος.

3.1.8.1 Επιτάχυνση Υλικού μέσω FSL

Κάθε FSL παρέχει μία εξειδικευμένη διαπροσωπεία μικρής καθυστέρηση στη σωλήνωση του επεξεργαστή. Είναι ιδανική, λοιπόν, η χρήση αυτών για την επέκταση της μονάδας εκτέλεσης του επεξεργαστή με ειδικούς επιταχυντές υλικού. Ένα απλό παράδειγμα φαίνεται στο Σχήμα 3.7.



Σχήμα 3.7: Χρήση του FSL

Αυτή η μέθοδος είναι παρόμοια με την επέκταση του συνόλου εντολών με ειδικές εντολές αλλά έχει το πλεονέκτημα ότι δεν κάνει τη συνολική ταχύτητα της σωλήνωσης του επεξεργαστή να εξαρτάται από

την ειδική λειτουργία. Επίσης, δεν υπάρχουν επιπρόσθετες απαιτήσεις στην αλυσίδα των εργαλείων λογισμικού που να συσχετίζονται με αυτόν τον τύπο της λειτουργικής επέκτασης.

3.1.9 Συμβάσεις Μεταγλωττιστή του MicroBlaze

Ο MicroBlaze GNU Compiler που χρησιμοποιείται για την ανάπτυξη λογισμικού ακολουθεί τις συμβάσεις που περιγράφονται σε αυτό το τμήμα. Αυτές τις συμβάσεις θα πρέπει να ακολουθούν οι προγραμματιστές που θα γράψουν κώδικα σε συμβολική γλώσσα, για να πετύχουν τη συμβατότητα μεταξύ του κώδικά τους και του παραγόμενου από το μεταγλωττιστή κώδικα. Επίσης, εξηγείται σύντομα σε αυτό το τμήμα ο χειρισμός διακοπών και εξαιρέσεων.

3.1.9.1 Τύποι Δεδομένων

Οι τύποι δεδομένων που χρησιμοποιούνται από τα προγράμματα του MicroBlaze σε συμβολική γλώσσα φαίνονται στο Σχήμα 3.8. Οι τύποι δεδομένων data8, data16 και data32 χρησιμοποιούνται στη θέση των συνήθων byte, halfword και word.

MicroBlaze data types (for assembly programs)	Corresponding ANSI C data types	Size (bytes)
data8	char	1
data16	short	2
data32	int	4
data32	long int	4
data32	float	4
data32	enum	4
data16/data32	pointer[a]	2/4

a.Pointers to small data areas, which can be accessed by global pointers are data16.

Σχήμα 3.8: Τύποι Δεδομένων στα Προγράμματα Συμβολικής Γλώσσας

3.1.9.2 Συμβάσεις Χρήσης Καταχωρητών

Στο Σχήμα 3.9 φαίνεται η σύμβαση για τη χρήση των καταχωρητών του επεξεργαστή

MicroBlaze.

Register	Type	Enforcement	Purpose
R0	Dedicated	HW	Value 0
R1	Dedicated	SW	Stack Pointer
R2	Dedicated	SW	Read-only small data area anchor
R3-R4	Volatile	SW	Return Values/Temporaries
R5-R10	Volatile	SW	Passing parameters/Temporaries
R11-R12	Volatile	SW	Temporaries
R13	Dedicated	SW	Read-write small data area anchor
R14	Dedicated	HW	Return address for Interrupt
R15	Dedicated	SW	Return address for Sub-routine
R16	Dedicated	HW	Return address for Trap (Debugger)
R17	Dedicated	HW, if configured to support HW exceptions, else SW	Return Address for Exceptions
R18	Dedicated	SW	Reserved for Assembler
R19-R31	Non-volatile	SW	Must be saved across function calls. Callee-save
RPC	Special	HW	Program counter
RMSR	Special	HW	Machine Status Register
REAR	Special	HW	Exception Address Register
RESR	Special	HW	Exception Status Register
RFSR	Special	HW	Floating Point Status Register

Σχήμα 3.9: Συμβάσεις Χρήσης Καταχωρητών

Η αρχιτεκτονική του MicroBlaze καθορίζει 32 καταχωρητές γενικού σκοπού. Αυτοί οι καταχωρητές κατηγοριοποιούνται ως volatile, non-volatile και εξειδικευμένοι.

Οι volatile (ή caller-save) καταχωρητές χρησιμοποιούνται σαν προσωρινοί καταχωρητές και δεν διατηρούν τις τιμές τους ανάμεσα σε κλήσεις συναρτήσεων. Οι καταχωρητές R3 μέχρι και R12 είναι volatile, εκ των οποίων οι R3 και R4 χρησιμοποιούνται για την επιστροφή τιμών στην καλούσα συνάρτηση, αν επιστρέφονται τιμές. Οι καταχωρητές R5 μέχρι και R10 χρησιμοποιούνται για το πέρασμα παραμέτρων ανάμεσα σε υπορουτίνες.

Οι καταχωρητές R19 μέχρι και R31 διατηρούν τα περιεχόμενά τους ανάμεσα σε κλήσεις συναρτήσεων και χαρακτηρίζονται ως non-volatile (ή callee-save) καταχωρητές. Η καλούμενη συνάρτηση αναμένεται να σώσει αυτούς τους non-volatile καταχωρητές, οι οποίοι χρησιμοποιούνται. Αυτοί τυπικά σώζονται στη στοίβα κατά τον πρόλογο και μετά ξαναφορτώνονται κατά τον επίλογο.

Ορισμένοι καταχωρητές χρησιμοποιούνται σαν εξειδικευμένοι καταχωρητές και οι προγραμματιστές δεν πρέπει να τους χρησιμοποιούν για κανένα άλλο σκοπό.

Οι καταχωρητές R14 μέχρι και R17 χρησιμοποιούνται για την αποθήκευση της διεύθυνσης επιστροφής από διακοπές, υπορουτίνες, traps και εξαιρέσεις με αυτή τη σειρά. Οι υπορουτίνες καλούνται χρησιμοποιώντας την εντολής διακλάδωσης και σύνδεσης (branch and link), η οποία σώζει τον τρέχοντα μετρητή προγράμματος στον καταχωρητή R15.

Οι δείκτες σε περιοχή μικρών δεδομένων χρησιμοποιούνται για την πρόσβαση σε συγκεκριμένες θέσεις μνήμης με άμεση τιμή 16-bit. Αυτές οι περιοχές συζητιούνται στο τμήμα μοντέλου μνήμης. Ο καταχωρητής R2 χρησιμοποιείται για την πρόσβαση στις σταθερές, όπως, για παράδειγμα, στις λεκτικές σταθερές, στην περιοχή μικρών δεδομένων ανάγνωσης μόνο. Ο καταχωρητής R13 χρησιμοποιείται για την πρόσβαση στις τιμές στο τμήμα μικρών δεδομένων ανάγνωσης/εγγραφής.

Ο καταχωρητής R1 αποθηκεύει την τιμή του δείκτη στοίβας και ενημερώνεται κατά την είσοδο και κατά την έξοδο από τις συναρτήσεις.

Ο καταχωρητής R18 χρησιμοποιείται σαν προσωρινός καταχωρητής για λειτουργίες του συμβολομεταφραστή.

Ο MicroBlaze περιλαμβάνει, επίσης, τους εξής καταχωρητές ειδικού σκοπού: Program Counter, Machine Status Register, Exception Status Register, Exception Address Register και Floating Point Status Register. Αυτοί οι καταχωρητές δεν απεικονίζονται απευθείας στο αρχείο καταχωρητών και, έτσι, η χρήση αυτών των καταχωρητών είναι διαφορετική από τη χρήση των καταχωρητών γενικού σκοπού. Η μεταφορά της τιμής από έναν καταχωρητή ειδικού σκοπού σε έναν καταχωρητή γενικού σκοπού και αντίστροφα μπορεί να γίνει αντίστοιχα με τη χρήση των εντολών mfs και mts.

3.1.9.3 Σύμβαση Στοίβας

Στο Σχήμα 3.10 φαίνονται οι συμβάσεις στοίβας που χρησιμοποιούνται από το MicroBlaze. Η σκιασμένη περιοχή αποτελεί ένα τμήμα της στοίβας της καλούσας συνάρτησης ενώ η μη-σκιασμένη περιοχή αποτελεί τη στοίβα της καλούμενης συνάρτησης. Οι συμβάσεις της στοίβας καθορίζουν το πρωτόκολλο για το πέρασμα παραμέτρων, προστατεύοντας τις τιμές των non-volatile καταχωρητών και δεσμεύοντας χώρο για τις τοπικές μεταβλητές σε μία συνάρτηση. Οι συναρτήσεις που περιέχουν κλήσεις σε άλλες υπορουτίνες ονομάζονται όχι-φύλλα συναρτήσεις. Αυτές οι όχι-φύλλα συναρτήσεις πρέπει να δημιουργήσουν ένα νέο τμήμα της στοίβας για ιδία χρήση. Όταν το πρόγραμμα αρχίσει να

εκτελείται, ο δείκτης στοίβας θα έχει τη μέγιστη τιμή. Όταν καλούνται συναρτήσεις, ο δείκτης στοίβας μειώνεται κατά τον αριθμό των λέξεων που απαιτούνται από κάθε συνάρτηση για το δικό της τμήμα στοίβας. Ο δείκτης στοίβας μίας καλούσας συνάρτησης θα έχει πάντα μεγαλύτερη τιμή συγκρινόμενος με το δείκτη στοίβας της καλούμενης συνάρτησης.

High Address	
	Function Parameters for called sub-routine (Arg n ..Arg1) (Optional: Maximum number of arguments required for any called procedure from the current procedure.)
Old Stack Pointer	Link Register (R15)
	Callee Saved Register (R31....R19) (Optional: Only those registers which are used by the current procedure are saved)
	Local Variables for Current Procedure (Optional: Present only if Locals defined in the procedure)
	Functional Parameters (Arg n .. Arg 1) (Optional: Maximum number of arguments required for any called procedure from the current procedure)
New Stack Pointer	Link Register
Low Address	

Σχήμα 3.10: Σύμβαση Στοίβας

Σύμβαση Κλήσης

Η καλούσα συνάρτηση περνάει παραμέτρους στην καλούμενη συνάρτηση με τη χρήση των καταχωρητών R5 – R10 ή πάνω στο δικό της τμήμα στοίβας. Η καλούμενη συνάρτηση χρησιμοποιεί το τμήμα της στοίβας της καλούσας συνάρτησης για την αποθήκευση των παραμέτρων που περάστηκαν στην καλούμενη συνάρτηση.

3.1.9.4 Μοντέλο Μνήμης

Το μοντέλο μνήμης για τον MicroBlaze κατηγοριοποιεί τα δεδομένα σε τέσσερα διαφορετικά τμήματα.

Περιοχή Μικρών Δεδομένων

Οι καθολικές αρχικοποιημένες μεταβλητές που είναι μικρές σε μέγεθος αποθηκεύονται σε αυτήν την περιοχή. Το κατώφλι για την απόφαση του μεγέθους της μεταβλητής που θα αποθηκευτεί στην περιοχή μικρών δεδομένων τίθεται στα 8 bytes στον MicroBlaze C Compiler αλλά αυτό μπορεί να αλλάξει δίνοντας μία επιλογή στη γραμμή εντολών στον μεταγλωττιστή. 64 KB μνήμης δεσμεύονται για τις περιοχές μικρών δεδομένων. Η πρόσβαση στην περιοχή μικρών δεδομένων γίνεται με τη χρήση του καταχωρητή R13, της βάσης της περιοχής μικρών δεδομένων ανάγνωσης/εγγραφής, και με μία 16-bit απόκλιση. Η δέσμευση μικρών μεταβλητών σε αυτήν την περιοχή μειώνει την απαίτηση της προσθήκης εντολών Imm στον κώδικα για την πρόσβαση σε καθολικές μεταβλητές. Η πρόσβαση σε κάθε μεταβλητή στην περιοχή μικρών δεδομένων μπορεί, επίσης, να γίνει με τη χρήση μία απόλυτης διεύθυνσης.

Περιοχή Δεδομένων

Συγκριτικά μεγάλες αρχικοποιημένες μεταβλητές δεσμεύονται στην περιοχή δεδομένων, η πρόσβαση στις οποίες μπορεί να γίνει με τη χρήση της βάσης της περιοχής μικρών δεδομένων ανάγνωσης/εγγραφής R13 ή με τη χρήση της απόλυτης διεύθυνσης, ανάλογα με την επιλογή της γραμμής εντολών που δίνεται στον μεταγλωττιστή.

Κοινή Μη-αρχικοποιημένη Περιοχή

Μη-αρχικοποιημένες καθολικές μεταβλητές δεσμεύονται στην κοινή περιοχή και η πρόσβαση σε αυτές μπορεί να γίνει με τη χρήση της απόλυτης διεύθυνσης ή με τη χρήση της βάσης της περιοχής μικρών δεδομένων ανάγνωσης/εγγραφής R13.

Σταθερές ή Λεκτικές Σταθερές

Οι σταθερές τοποθετούνται στην περιοχή μικρών δεδομένων ανάγνωσης μόνο και η πρόσβαση σε αυτές γίνεται με τη χρήση της βάσης της περιοχής μικρών δεδομένων ανάγνωσης μόνο R2.

Ο μεταγλωττιστής παράγει τους κατάλληλους καθολικούς δείκτες που συμπεριφέρονται σαν

δείκτες βάσης. Οι πραγματικές τιμές των βάσεων των περιοχών μικρών δεδομένων καθορίζονται από τον συνδέτη στα τελικά στάδια της σύνδεσης. Ο μεταγλωττιστής παράγει τις κατάλληλες περιοχές ανάλογα με τις επιλογές στη γραμμή εντολών.

3.1.9.5 Χειρισμός Διακοπών και Εξαιρέσεων

Ο MicroBlaze θεωρεί συγκεκριμένες διευθύνσεις μνήμης για το χειρισμό διακοπών και εξαιρέσεων, όπως φαίνεται στο Σχήμα 3.11. Σε αυτές τις θέσεις γράφεται κώδικας για το άλμα στις αντίστοιχες ρουτίνες εξυπηρέτησης.

On	Hardware jumps to	Software Labels
Start / Reset	0x0	_start
User exception	0x8	_exception_handler
Interrupt	0x10	_interrupt_handler
Break (HW/SW)	0x18	-
Hardware exception	0x20	_hw_exception_handler
Reserved by Xilinx for future use	0x28 - 0x4F	-

Σχήμα 3.11: Χειρισμός Διακοπών και Εξαιρέσεων

Ο κώδικας που αναμένεται σε αυτές τις θέσεις φαίνεται στο Σχήμα 3.12. Για τα προγράμματα που μεταγλωττίζονται χωρίς την επιλογή -xl-mode-xmdstub του μεταγλωττιστή το αρχείο αρχικοποίησης crt0.o περνάει από τον μεταγλωττιστή στον συνδέτη για σύνδεση. Αυτό το αρχείο θέτει τις κατάλληλες διευθύνσεις των χειριστών. Για τα προγράμματα που μεταγλωττίζονται με την επιλογή -xl-mode-xmdstub του μεταγλωττιστή το αρχείο αρχικοποίησης crt1.o συνδέεται με το πρόγραμμα εξόδου. Αυτό το πρόγραμμα πρέπει να τρέξει με το xmdstub ήδη φορτωμένο στη μνήμη στη θέση μνήμης 0x0. Στο χρόνο εκτέλεσης, λοιπόν, ο κώδικας αρχικοποίησης στο crt1.o γράφει τις κατάλληλες εντολές στις θέσεις 0x8 έως 0x14 ανάλογα με τις διευθύνσεις των χειριστών εξαιρέσεων και διακοπών.

```
0x00:   bri     _start1
0x04:   nop
0x08:   imm     high bits of address (user exception handler)
0x0c:   bri     _exception_handler
0x10:   imm     high bits of address (interrupt handler)
0x14:   bri     _interrupt_handler
0x20:   imm     high bits of address (HW exception handler)
0x24:   bri     _hw_exception_handler
```

Σχήμα 3.12: Κώδικας για το Πέρασμα Ελέγχου στους Χειριστές Εξαιρέσεων και Διακοπών

Ο MicroBlaze επιτρέπει στις ρουτίνες χειρισμού εξαιρέσεων και διακοπών να βρίσκονται σε οποιαδήποτε θέση μνήμης που εντοπίζεται με τη χρήση 32 bits. Ο κώδικας για το χειριστή της εξαίρεσης χρήστη ξεκινάει με την ετικέτα _exception_handler, για το χειριστή της εξαίρεσης υλικού ξεκινάει με την ετικέτα _hw_exception_handler ενώ για το χειριστή διακοπής ξεκινάει με την ετικέτα _interrupt_handler.

Στο τρέχον σύστημα του MicroBlaze υπάρχουν dummy ρουτίνες για το χειρισμό διακοπών και εξαιρέσεων, οι οποίες μπορούν να αλλάξουν. Για την αλλαγή αυτών των ρουτινών και τη σύνδεση νέων χειριστών διακοπών και εξαιρέσεων πρέπει να καθοριστεί ο κώδικας των χειριστών με ένα attribute. Για το χειρισμό διακοπής, για παράδειγμα, θα πρέπει να δηλωθεί με το attribute interrupt_handler. Έστω ότι η ρουτίνα είναι η interruptServiceRoutine. Θα πρέπει να δηλωθεί το εξής:

void interruptServiceRoutine() __attribute__((interrupt_handler));

3.2 OPB Interrupt Controller

Ο ελεγκτής διακοπών OPB Interrupt Controller είναι ένας απλός και παραμετροποιήσιμος ελεγκτής διακοπών. Διαθέτει διαπροσωπεία για σύνδεση σε διάδρομο και μπορεί να συνδεθεί στο On-chip Peripheral Bus, με αποτέλεσμα να μπορεί να χρησιμοποιηθεί σε ενσωματωμένα συστήματα βασισμένα στον επεξεργαστή MicroBlaze. Η προτεραιότητα μεταξύ των αιτήσεων διακοπών καθορίζεται από τη θέση τους στο διάνυσμα διακοπών (vector position). Το λιγότερο σημαντικό bit έχει την μεγαλύτερη προτεραιότητα. Το πλήθος των εισόδων διακοπών είναι ρυθμιζόμενο μέχρι και το πλάτος του διαδρόμου δεδομένων. Υπάρχει η δυνατότητα χρήσης πολλαπλών στιγμιοτύπων του ελεγκτή για την κάλυψη πρόσθετων εισόδων διακοπών. Επίσης, υπάρχει η δυνατότητα απενεργοποίησης συγκεκριμένων εισόδων διακοπών από τον Interrupt Enable Register αλλά και της

εξόδου αίτησης διακοπής από τον Master Enable Register. Κάθε είσοδος μπορεί να ρυθμιστεί, ώστε να προκαλεί διακοπή σε θετική ή αρνητική ακμή ή σε υψηλή ή χαμηλή στάθμη. Το ίδιο συμβαίνει και με την έξοδο αίτησης διακοπής. Διαθέτει κύκλωμα αυτόματου συγχρονισμού της εισόδου διακοπής, όταν αυτή έχει ρυθμιστεί να προκαλεί διακοπή με ακμή. Ορισμένοι καταχωρητές του ελεγκτή διακοπών μπορούν να παραμετροποιηθούν εκτός της υλοποίησης με σκοπό τη μείωση των χρησιμοποιούμενων από το FPGA πόρων. Περισσότεροι καταχωρητές, όπως και περισσότερες είσοδοι διακοπών, μεταφράζονται σε περισσότερους χρησιμοποιούμενους πόρους του FPGA.

3.2.1 Λειτουργική Περιγραφή

Οι ελεγκτές διακοπών χρησιμοποιούνται για να επεκτείνουν τον αριθμό των εισόδων διακοπών που διαθέτει ένα υπολογιστικό σύστημα και που απευθύνονται σε έναν επεξεργαστή. Παρέχουν, ενδεχομένως, και ένα σχήμα κωδικοποίησης και προγραμματισμού της προτεραιότητας των διακοπών. Οι μοντέρνοι επεξεργαστές διαθέτουν μία ή περισσότερες εισόδους αίτησης διακοπής που επιτρέπουν στις εξωτερικές συσκευές να ζητήσουν εξυπηρέτηση από τον επεξεργαστή.

Υπάρχουν δύο κύριοι μηχανισμοί αίτησης διακοπής που χρησιμοποιούνται από τους επεξεργαστές. Τα auto vectoring σχήματα διακοπής παρέχουν ένα σήμα αίτησης διακοπής στον επεξεργαστή και κατά την αναγνώριση της διακοπής ο ελεγκτής διακοπών παρέχει ολόκληρη ή τμήμα της διεύθυνσης της ρουτίνας εξυπηρέτησης διακοπής στον επεξεργαστή. Τα hard vector σχήματα διακοπής παρέχουν μία ή περισσότερες καθορισμένες θέσεις στη μνήμη, μία για κάθε είσοδο αίτησης διακοπής ή μία θέση για όλες τις εισόδους διακοπής. Σε κάθε περίπτωση, μερικοί ελεγκτές διακοπών επιτρέπουν τον προγραμματισμό της πολικότητας των εισόδων διακοπών και του αν είναι ευαίσθητες σε στάθμη ή ακμή.

Ο OPB Interrupt Controller υποστηρίζει το σχήμα διακοπής hard vector. Υπάρχει ακριβώς μία θέση στη μνήμη για όλες τις διακοπές. Επίσης, ρυθμίζεται στον ελεγκτή διακοπών η προτεραιότητα κάθε διακοπής. Από τα παραπάνω συνάγεται ότι είναι θέμα του λογισμικού να εξακριβώσει την πηγή της διακοπής. Και ο ελεγκτής διακοπών προσφέρει τη δυνατότητα αυτή με την παροχή του αριθμού της διακοπής στο διάνυσμα διακοπών (vector number).

3.2.2 Οργάνωση Ελεγκτή Διακοπών

Ο ελεγκτής διακοπών οργανώνεται στις επόμενες τρεις λειτουργικές μονάδες:

1. Ανίχνευση διακοπής και παραγωγή αίτησης.
2. Καταχωρητές προγραμματιστή.
3. Διαπροσωπεία διαδρόμου.

3.2.2.1 Ανίχνευση Διακοπής

Η ανίχνευση διακοπής μπορεί να ρυθμιστεί είτε για στάθμη είτε για ακμή και ξεχωριστά για κάθε είσοδο διακοπής. Αν επιλεχθεί ανίχνευση ακμής, τότε συμπεριλαμβάνονται και καταχωρητές συγχρονισμού. Η παραγωγή αίτησης διακοπής ρυθμίζεται, επίσης, είτε σαν παλμός εξόδου για αίτηση ευαίσθητη στην ακμή είτε σαν στάθμη εξόδου που καθαρίζεται όταν αναγνωριστεί η διακοπή.

3.2.2.2 Καταχωρητές Προγραμματιστή

Ο ελεγκτής διακοπών διαθέτει τους ακόλουθους καταχωρητές που είναι προσβάσιμοι από τον προγραμματιστή:

1. Interrupt Status Register (ISR). Αυτός είναι ένας καταχωρητής ανάγνωσης/εγγραφής που, όταν διαβαστεί, δείχνει ποιες είσοδοι διακοπής είναι ενεργές (active). Εγγραφή στον ISR επιτρέπει στο λογισμικό να παράγει διακοπές μέχρι να ενεργοποιηθεί το HIE bit του Master Enable Register.
2. Interrupt Pending Register (IPR). Αυτός είναι ένας καταχωρητής ανάγνωσης μόνο που παρέχει μία ένδειξη των διακοπών που είναι ενεργές (active) και ενεργοποιημένες (enabled). Πρόκειται για έναν προαιρετικό καταχωρητή που μπορεί να παραμετροποιηθεί ώστε να μην υπάρχει στην υλοποίηση με σκοπό τη μείωση των πόρων που απαιτεί ο ελεγκτής διακοπών από το FPGA.
3. Interrupt Enable Register (IER). Αυτός είναι ένας καταχωρητής ανάγνωσης/εγγραφής, τα περιεχόμενα του οποίου χρησιμοποιούνται για την ενεργοποίηση επιλεγμένων διακοπών.
4. Interrupt Acknowledge Register (IAR). Δεν είναι πραγματικά ένας καταχωρητής. Πρόκειται για μία θέση εγγραφής μόνο που χρησιμοποιείται για τον καθαρισμό αιτήσεων διακοπών.
5. Set Interrupt Enables (SIE). Δεν είναι πραγματικά ένας καταχωρητής. Πρόκειται για μία θέση εγγραφής μόνο που παρέχει τη δυνατότητα ενεργοποίησης συγκεκριμένων bits του IER σε μία ατομική πράξη, χωρίς την απαίτηση μίας ακολουθίας ανάγνωσης/τροποποίησης/εγγραφής. Μπορεί να παραμετροποιηθεί ώστε να μην υπάρχει στην υλοποίηση με σκοπό τη μείωση των

πόρων που απαιτεί ο ελεγκτής διακοπών από το FPGA.

6. Clear Interrupt Enables (CIE). Δεν είναι πραγματικά ένας καταχωρητής. Πρόκειται για μία θέση εγγραφής μόνο που παρέχει τη δυνατότητα απενεργοποίησης συγκεκριμένων bits του IER σε μία ατομική πράξη. Μπορεί να παραμετροποιηθεί ώστε να μην υπάρχει στην υλοποίηση με σκοπό τη μείωση των πόρων που απαιτεί ο ελεγκτής διακοπών από το FPGA.
7. Interrupt Vector Register (IVR). Είναι ένας καταχωρητής ανάγνωσης μόνο που περιέχει την τάξη (ordinal value) της διακοπής με την υψηλότερη προτεραιότητα που είναι ενεργή και ενεργοποιημένη. Πρόκειται για έναν προαιρετικό καταχωρητή που μπορεί να παραμετροποιηθεί ώστε να μην υπάρχει στην υλοποίηση με σκοπό τη μείωση των πόρων που απαιτεί ο ελεγκτής διακοπών από το FPGA.
8. Master Enable Register (MER). Είναι ένας 2-bit καταχωρητής ανάγνωσης/εγγραφής που χρησιμοποιείται για την ενεργοποίηση ή απενεργοποίηση της εξόδου αίτησης διακοπής και την ενεργοποίηση των διακοπών υλικού.

3.2.2.3 Διαπροσωπεία Διαδρόμου

Η διαπροσωπεία του On-chip Peripheral Bus παρέχει μία διαπροσωπεία σκλάβου (slave interface) πάνω στο OPB για τη μεταφορά δεδομένων ανάμεσα στον OPB Interrupt Controller και τον επεξεργαστή. Οι καταχωρητές του ελεγκτή διακοπών είναι απεικονισμένοι στη μνήμη (memory mapped) μέσα στο διάστημα διευθύνσεων του OPB. Οι διευθύνσεις των καταχωρητών είναι τοποθετημένες σε όρια μίας λέξης και οι καταχωρητές, καθώς και οι μεταφορές δεδομένων από και προς αυτούς, έχουν το πλάτος του διαδρόμου δεδομένων. Ο αριθμός των εισόδων διακοπής ρυθμίζεται μέχρι και το πλάτος του διαδρόμου δεδομένων, που, επίσης, καθορίζεται από μία ρυθμιζόμενη παράμετρο, όπως και η διεύθυνση βάσης για τους καταχωρητές. Ανεξάρτητα, ωστόσο, από το πλάτος του διαδρόμου, αρκετά στιγμιότυπα του ελεγκτή διακοπών μπορούν να παραταχθούν για την παροχή οποιουδήποτε αριθμού εισόδων διακοπής.

3.2.3 Τύποι Δεδομένων Καταχωρητών και Οργάνωση

Η πρόσβαση σε όλους τους καταχωρητές του ελεγκτή διακοπών γίνεται μέσω της διαπροσωπείας του On-chip Peripheral Bus. Η διεύθυνση βάσης αυτών των καταχωρητών παρέχεται από μία ρυθμιζόμενη παράμετρο. Για έναν OPB Interrupt Controller η πρόσβαση σε κάθε καταχωρητή

γίνεται με απόκλιση σε όρια μίας λέξης από τη διεύθυνση βάσης, ανεξάρτητα από το πλάτος των καταχωρητών, με σκοπό την υπακοή στη σύμβαση θέσης καταχωρητών OPB-IPIF.

Επειδή οι διευθύνσεις του OPB είναι διευθύνσεις byte, οι καταχωρητές του ελεγκτή διακοπών βρίσκονται με απόκλιση κάποιο ακέραιο πολλαπλάσιο του 4 από τη διεύθυνση βάσης ο καθένας. Στο Σχήμα 3.13 φαίνονται οι καταχωρητές και οι αποκλίσεις τους από τη διεύθυνση βάσης για έναν OPB Interrupt Controller.

Register Name	Abbreviation	OPB Offset
Interrupt Status Register	ISR	0 (00h)
Interrupt Pending Register	IPR	4 (04h)
Interrupt Enable Register	IER	8 (08h)
Interrupt Acknowledge Register	IAR	12 (0Ch)
Set Interrupt Enable Bits	SIE	16 (10h)
Clear Interrupt Enable Bits	CIE	20 (14h)
Interrupt Vector Register	IVR	24 (18h)
Master Enable Register	MER	28 (1Ch)

Σχήμα 3.13: Καταχωρητές και Αποκλίσεις

Οι καταχωρητές του ελεγκτή διακοπών διαβάζονται σαν big-endian δεδομένα. Οι ετικέτες των bits και των bytes για τα big-endian δεδομένα φαίνονται στο Σχήμα 3.14 για τον τύπο word.

Byte address	n	n+1	n+2	n+3	
Byte label	0	1	2	3	Word
Byte significance	MS Byte			LS Byte	
Bit label	0			31	
Bit significance	MS Bit			LS Byte	

Σχήμα 3.14: Ο Τύπος Δεδομένων word

3.2.4 Καταχωρητές

Οι οκτώ καταχωρητές που είναι ορατοί στον προγραμματιστή φαίνονται στο Σχήμα 3.13. Σε γκρίζο φόντο είναι οι παρόντες σε κάθε υλοποίηση καταχωρητές ενώ σε λευκό φόντο είναι οι

προαιρετικοί καταχωρητές. Η μεγαλύτερης προτεραιότητας διακοπή αντιστοιχίζεται στο λιγότερο σημαντικό bit των πρώτων έξι καταχωρητών ενώ η προτεραιότητα φθίνει πηγαίνοντας προς τα αριστερότερα bits στα οποία αντιστοιχίζονται διαδοχικά οι υπόλοιπες διακοπές. Αν δεν ειπωθεί κάτι διαφορετικό, bits που δεν αντιστοιχούν σε εισόδους επιστρέφουν 0 κατά την ανάγνωση και δεν κάνουν τίποτα κατά την εγγραφή.

3.2.4.1 Interrupt Status Register

Όταν διαβαστούν, τα περιεχόμενα του καταχωρητή αυτού δείχνουν την παρουσία ή την απουσία ενός ενεργού (active) σήματος διακοπής ανεξάρτητα από την κατάσταση του IER. Κάθε bit στον καταχωρητή που είναι 1 δείχνει ένα ενεργό σήμα διακοπής στην αντίστοιχη είσοδο διακοπής. Τα bits που είναι 0 δεν είναι ενεργά. Ο ISR είναι εγγράψιμος από το λογισμικό μέχρι το Hardware Interrupt Enable (HIE) bit στον MER να τεθεί. Μόλις τεθεί αυτό το bit, το λογισμικό δεν μπορεί πλέον να γράψει στον ISR. Με δεδομένους αυτούς τους περιορισμούς, όταν αυτός ο καταχωρητής γράφεται, όσα bits λαμβάνουν την τιμή 1 θα καταστήσουν ενεργή την αντίστοιχη διακοπή, σαν να γινόταν ενεργή η είσοδος υλικού. Τα bits που έχουν την τιμή 0 δεν έχουν κανένα αποτέλεσμα. Αυτό επιτρέπει στο λογισμικό να παράγει διακοπές κατά τη διαδικασία αποσφαλμάτωσης μέχρι να τεθεί το HIE bit. Μόλις τεθεί το bit αυτό, τότε είναι δυνατόν να προκληθούν διακοπές από το υλικό και η εγγραφή σε αυτόν τον καταχωρητή δεν κάνει τίποτα.

3.2.4.2 Interrupt Pending Register

Αυτός είναι ένας προαιρετικός καταχωρητής του OPB Interrupt Controller. Η ανάγνωση των περιεχομένων αυτού του καταχωρητή δείχνει την παρουσία ή την απουσία ενός ενεργού σήματος διακοπής που είναι, επίσης, ενεργοποιημένο. Κάθε bit στον καταχωρητή αυτόν είναι το λογικό AND των bits των ISR και IER.

3.2.4.3 Interrupt Enable Register

Αυτός είναι ένας καταχωρητής ανάγνωσης/εγγραφής. Η εγγραφή ενός 1 σε ένα bit αυτού του καταχωρητή ενεργοποιεί το αντίστοιχο σήμα εισόδου διακοπής. Η εγγραφή ενός 0 σε ένα bit αυτού του καταχωρητή απενεργοποιεί το αντίστοιχο σήμα εισόδου διακοπής. Η απενεργοποίηση μίας εισόδου

διακοπής δεν έχει, βέβαια, την ίδια σημασία με τον καθαρισμό ή την αναγνώριση της διακοπής. Η απενεργοποίηση μίας ενεργής[1] διακοπής εμποδίζει αυτή τη διακοπή να φτάσει την έξοδο αίτησης διακοπής αλλά, μόλις ξαναενεργοποιηθεί η διακοπή, θα παράγει αμέσως μία αίτηση στην έξοδο αίτησης διακοπής. Μία διακοπή πρέπει να καθαριστεί με την εγγραφή στον IAR, όπως περιγράφεται παρακάτω. Η ανάγνωση του IER δείχνει ποιες είσοδοι διακοπών είναι ενεργοποιημένες, όπου το 1 σημαίνει ότι η είσοδος είναι ενεργοποιημένη ενώ το 0 σημαίνει ότι η είσοδος είναι απενεργοποιημένη.

3.2.4.4 Interrupt Acknowledge Register

Πρόκειται για μία θέση εγγραφής μόνο που καθαρίζει τις αιτήσεις διακοπών που σχετίζονται με τις επιλεγμένες εισόδους διακοπών. Η εγγραφή ενός 1 σε μία θέση bit στον IAR θα καθαρίσει την αίτηση διακοπής που προκλήθηκε από την αντίστοιχη είσοδο διακοπής. Μία διακοπή που είναι ενεργή και απενεργοποιημένη από την εγγραφή ενός 0 στο αντίστοιχο bit του IER θα παραμείνει ενεργή μέχρι να καθαριστεί με την αναγνώρισή της. Η ενεργοποίηση μίας ενεργής διακοπής θα προκαλέσει την παραγωγή μίας εξόδου αίτησης διακοπής, αν το bit ME στον MER έχει τεθεί.

3.2.4.5 Set Interrupt Enables

Πρόκειται για μία θέση που χρησιμοποιείται για να θέτει bits του IER σε μία ατομική πράξη. Η εγγραφή ενός 1 σε μία θέση bit του SIE θα θέσει το αντίστοιχο bit στον IER. Μπορεί να παραμετροποιηθεί εκτός της υλοποίησης.

3.2.4.6 Clear Interrupt Enables

Πρόκειται για μία θέση που χρησιμοποιείται για να καθαρίζει bits του IER σε μία ατομική πράξη, χωρίς τη χρήση μίας ακολουθίας ανάγνωσης/τροποποίησης/εγγραφής. Η εγγραφή ενός 1 σε μία θέση bit στον CIE καθαρίζει το αντίστοιχο bit στον IER. Ο CIE είναι επίσης προαιρετικός στον OPB Interrupt Controller.

3.2.4.7 Interrupt Vector Register

1 Για αποφυγή σύγχυσης υπενθυμίζεται η μετάφραση του active ως ενεργού και του enabled ως ενεργοποιημένου. Ομοίως προκύπτουν τα αντίθετα των προηγουμένων, δηλαδή το inactive μεταφράζεται ως ανενεργός και το disabled ως απενεργοποιημένος. Χρησιμοποιούνται, επίσης, ο όρος masked αντί του disabled και ο όρος unmasked αντί του enabled.

Ο IVR είναι ένας καταχωρητής ανάγνωσης μόνο που περιέχει την τάξη της ενεργής και ενεργοποιημένης εισόδου διακοπής με τη μεγαλύτερη προτεραιότητα. Αν δεν υπάρχει τέτοια είσοδος διακοπής, τότε ο IVR θα περιέχει όλο 1. Μπορεί να παραμετροποιηθεί εκτός της υλοποίησης για την κατανάλωση λιγότερων πόρων του FPGA.

3.2.4.8 Master Enable Register

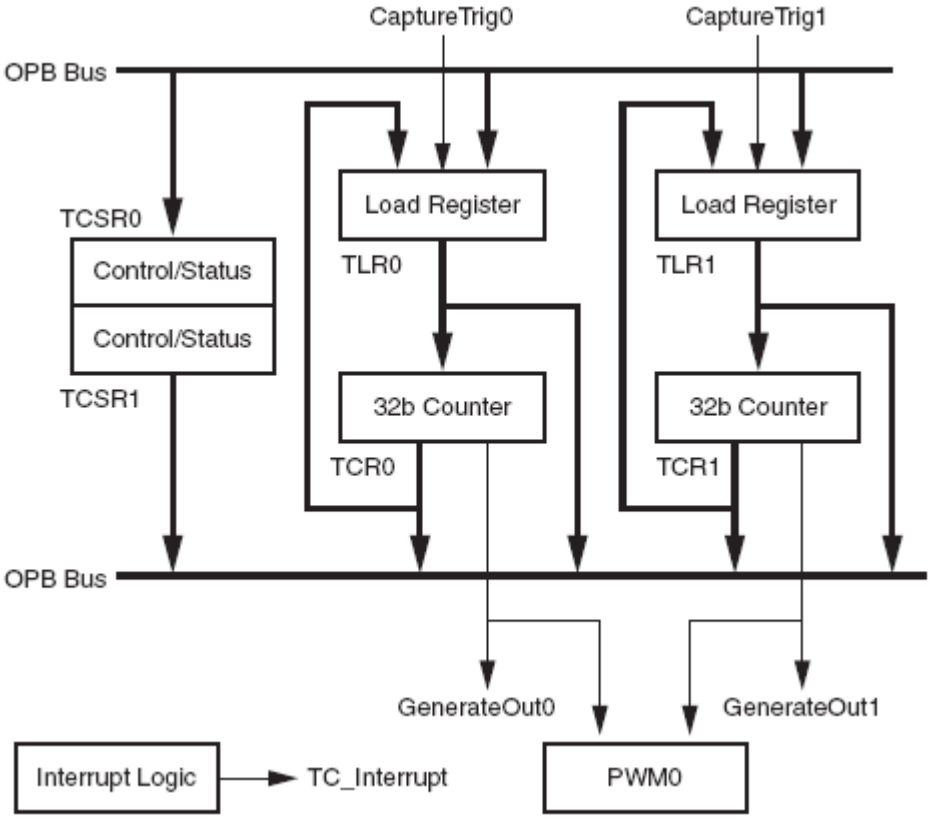
Είναι ένας 2-bit καταχωρητής ανάγνωσης/εγγραφής. Τα 2 bits απεικονίζονται στα 2 λιγότερο σημαντικά ψηφία του. Το λιγότερο σημαντικό bit περιέχει το Master Enable bit και το επόμενο bit περιέχει το Hardware Interrupt Enable bit. Η εγγραφή ενός 1 στο bit ME ενεργοποιεί το σήμα εξόδου αίτησης διακοπής. Η εγγραφή ενός 0 στο ίδιο bit απενεργοποιεί την έξοδο αίτησης διακοπής, εμποδίζοντας, συνεπώς, όλες τις εισόδους διακοπών να φτάσουν στην έξοδο. Το HIE bit είναι ένα bit εγγραφής μόνο μίας φοράς. Κατά το reset αυτό το bit γίνεται 0, επιτρέποντας στο λογισμικό να γράφει στον ISR, ώστε να παράγει διακοπές για την αποσφαλμάτωση του συστήματος, και απενεργοποιώντας τις εισόδους διακοπών υλικού. Η εγγραφή ενός 1 σε αυτό το bit αφενός εμποδίζει περαιτέρω αλλαγές στο bit και αφετέρου ενεργοποιεί τις εισόδους διακοπών υλικού ενώ απενεργοποιεί τις διακοπές από το λογισμικό.

3.3 OPB Timer/Counter

Ο OPB Timer/Counter είναι μία μονάδα μετρητή 32-bit που διαθέτει σύνθετες λειτουργίες μέτρησης και παραγωγής παλμών και συνδέεται στο On-chip Peripheral Bus του MicroBlaze. Υποστηρίζει μετρήσεις μέχρι και 32-bit από δύο παραμετροποιήσιμους ως προς το εύρος μέτρησης προγραμματιζόμενους μετρητές με δυνατότητες διακοπής, παραγωγής και σύλληψης γεγονότων. Επιπλέον, διαθέτει μία έξοδο διαμόρφωσης εύρους παλμών (Pulse Width Modulation). Αποσκοπώντας στη μικρότερη απαραίτητη δέσμευση πόρων του FPGA, ο OPB Timer/Counter ρυθμίζεται έτσι ώστε να υποστηρίζει μόνο εκείνα τα χαρακτηριστικά που είναι απαραίτητα στην εκάστοτε εφαρμογή. Μεγαλύτερο εύρος μέτρησης συνεπάγεται και δέσμευση περισσότερων πόρων, όπως το ίδιο συνεπάγεται και η παρουσία και των δύο μετρητών σε αντίθεση με την παρουσία μόνο του ενός.

3.3.1 Λειτουργική Περιγραφή

Ο OPB Timer/Counter αποτελείται από δύο ίδιους μετρητές, όπως φαίνεται στο Σχήμα 3.15. Κάθε μετρητής έχει ένα συσχετισμένο με αυτόν καταχωρητή (τον Load Register) που χρησιμοποιείται για να κρατάει είτε την αρχική τιμή του μετρητή για παραγωγή γεγονότων είτε την τιμή σύλληψης, ανάλογα με τον τρόπο λειτουργίας του μετρητή. Η τιμή παραγωγής χρησιμοποιείται για την παραγωγή μίας διακοπής κατά τη λήξη ενός διαστήματος ή μία συνεχή ακολουθία διακοπών με ένα προγραμματιζόμενο ενδιάμεσο διάστημα. Η τιμή σύλληψης είναι η τιμή του μετρητή κατά την ανίχνευση ενός εξωτερικού γεγονότος. Η συχνότητα ρολογιού των μετρητών είναι αυτή του ρολογιού του On-chip Peripheral Bus. Όλες οι διακοπές του OPB Timer/Counter είναι λογικά συνδεδεμένες με τη λογική συνάρτηση OR, ώστε να παράγουν ένα μόνο εξωτερικό σήμα διακοπής. Η ρουτίνα εξυπηρέτησης διακοπής διαβάζει τους καταχωρητές ελέγχου/κατάστασης (Control/Status Registers), ώστε να καθορίσει την πηγή της διακοπής.



Σχήμα 3.15: Διάγραμμα του OPB Timer/Counter

3.3.2 Τρόποι Λειτουργίας

Υπάρχει η δυνατότητα χρήσης των τρόπων λειτουργίας παραγωγής, σύλληψης και

διαμόρφωσης εύρους παλμών με τους δύο μετρητές του OPB Timer/Counter. Παρακάτω περιγράφονται οι τρόποι λειτουργίας και τα χαρακτηριστικά τους.

3.3.2.1 Λειτουργία Παραγωγής

Στον τρόπο λειτουργίας παραγωγής, η τιμή του Load Register φορτώνεται στο μετρητή και ο μετρητής αρχίζει να μετράει πάνω ή κάτω, ανάλογα με το bit UDT στον Control/Status Register, όταν ενεργοποιηθεί ο μετρητής. Κατά τις μεταβάσεις της τιμής του μετρητή από όλα 1 σε όλα 0, όταν μετράει προς τα πάνω, και από όλα 0 σε όλα 1, όταν μετράει προς τα κάτω, ο μετρητής σταματάει ή αυτόματα ξαναφορτώνει την τιμή παραγωγής από τον Load Register και συνεχίζει να μετράει, ανάλογα με το bit ARHT στον Control/Status Register. Το bit TINT τίθεται στον Control/Status Register και, αν είναι ενεργοποιημένο, το εξωτερικό σήμα GenerateOut πηγαίνει στο 1 για έναν κύκλο ρολογιού. Επίσης, αν είναι ενεργοποιημένο, το σήμα διακοπής για τον μετρητή πηγαίνει στο 1 για έναν κύκλο ρολογιού. Αυτός ο τρόπος λειτουργίας είναι χρήσιμος για την παραγωγή επαναληπτικών διακοπών ή εξωτερικών σημάτων με ένα καθορισμένο ενδιάμεσο διάστημα.

Ο τρόπος λειτουργίας παραγωγής έχει τα ακόλουθα χαρακτηριστικά:

1. Η τιμή που φορτώνεται στο Load Register ονομάζεται τιμή παραγωγής.
2. Κατά την εκκίνηση η τιμή παραγωγής στον Load Register πρέπει να φορτωθεί στον μετρητή θέτοντας το Load bit στον Control/Status Register. Αυτό ισχύει είτε ο μετρητής είναι ρυθμισμένος να ξαναφορτώνει αυτόματα είτε να σταματάει, όταν έχει λήξει το χρονικό διάστημα. Κάνοντας 1 το Load bit φορτώνεται ο μετρητής με την τιμή του Load Register. Για τη σωστή λειτουργία του μετρητή, το Load bit πρέπει να καθαριστεί, πριν ο μετρητής ενεργοποιηθεί.
3. Όταν το ARHT bit (Auto Reload/Hold) είναι 1 και ο μετρητής περνάει από όλα 1 σε όλα 0 ή από όλα 0 σε όλα 1, η τιμή παραγωγής στο Load Register θα ξαναφορτωθεί αυτόματα στο μετρητή και ο μετρητής θα συνεχίσει να μετράει. Αν το σήμα GenerateOut είναι ενεργοποιημένο (bit GENT στον Control/Status Register), ένας παλμός εξόδου ενός κύκλου ρολογιού σε πλάτος θα παραχθεί. Αυτό είναι χρήσιμο για την παραγωγή μίας επαναληπτικής ακολουθίας παλμών με μία καθορισμένη περίοδο.
4. Όταν το ARHT bit είναι 0 και ο μετρητής περνάει από όλα 1 σε όλα 0 ή από όλα 0 σε όλα 1, ο μετρητής θα σταματήσει στην τρέχουσα τιμή και δε θα ξαναφορτώσει την τιμή παραγωγής. Αν

είναι ενεργοποιημένο το σήμα GenerateOut (bit GENT στον Control/Status Register), θα παραχθεί ένας παλμός εξόδου με πλάτος μίας περιόδου ενός κύκλου ρολογιού. Αυτό είναι χρήσιμο για έναν μοναδικό παλμό που είναι να παραχθεί μετά από ένα συγκεκριμένο χρονικό διάστημα.

5. Ο μετρητής μπορεί να ρυθμιστεί να μετράει είτε προς τα πάνω είτε προς τα κάτω (bit UDT στον Control/Status Register). Όταν ο μετρητής είναι ρυθμισμένος να μετράει προς τα κάτω, ισχύει:

   TIMING_INTERVAL = (TLR + 2) · OPB_CLOCK_PERIOD

   Όταν ο μετρητής είναι ρυθμισμένος να μετράει προς τα πάνω, ισχύει:

   TIMING_INTERVAL = (MAX_COUNT – TLR + 2) · OPB_CLOCK_PERIOD

   όπου MAX_COUNT είναι η μέγιστη τιμή μέτρησης του μετρητή, όπως, για παράδειγμα, η τιμή 0xFFFFFFFF για ένα μετρητή 32-bit.
6. Τα σήματα GenerateOut μπορούν να ρυθμιστούν σαν high-true ή low-true.

3.3.2.2 Λειτουργία Σύλληψης

Στον τρόπο λειτουργίας σύλληψης η τιμή του μετρητή αποθηκεύεται στον Load Register, όταν υπάρξει εξωτερικό σήμα σύλληψης. Το bit TINT τίθεται, επίσης, στον Control/Status Register με την ανίχνευση του γεγονότος σύλληψης. Ο μετρητής μπορεί να ρυθμιστεί να μετράει προς τα πάνω ή προς τα κάτω για αυτόν τον τρόπο λειτουργίας, ανάλογα με το UDT bit στον Control/Status Register. Το ARHT bit ελέγχει το αν η τιμή σύλληψης γράφεται από πάνω με μία νέα τιμή σύλληψης, πριν η προηγούμενη TINT σημαία καθαριστεί. Αυτός ο τρόπος λειτουργίας είναι χρήσιμος για την καταγραφή εξωτερικών γεγονότων παράγοντας ταυτόχρονα μία διακοπή.

Ο τρόπος λειτουργίας σύλληψης έχει τα ακόλουθα χαρακτηριστικά:

1. Το σήμα σύλληψης μπορεί να ρυθμιστεί ως low-true ή high-true.
2. Το σήμα σύλληψης δειγματοληπτείται με βάση το ρολόι του OPB. Το γεγονός σύλληψης ορίζεται ως η μετάβαση του σήματος που έχει υποστεί δειγματοληψία στην επιθυμητή κατάσταση. Για παράδειγμα, αν το σήμα σύλληψης έχει οριστεί ως high-true, τότε το γεγονός σύλληψης συμβαίνει όταν το σήμα που έχει υποστεί δειγματοληψία, συγχρονισμένο με το ρολόι του OPB, μεταβαίνει από το 0 στο 1.
3. Όταν συμβαίνει το γεγονός σύλληψης, τότε η τιμή του μετρητή γράφεται στον Load Register.

Αυτή η τιμή ονομάζεται τιμή σύλληψης.

4. Όταν το bit ARHT (Auto Reload/Hold) είναι 0 και συμβαίνει το γεγονός σύλληψης, η τιμή σύλληψης γράφεται στο Load Register που την κρατάει μέχρι να διαβαστεί ο Load Register. Αν ο καταχωρητής δε διαβαστεί, επόμενα γεγονότα σύλληψης δε θα ενημερώσουν το Load Register και θα χαθούν.
5. Όταν το bit ARHT είναι 1 και συμβαίνει το γεγονός σύλληψης, η τιμή σύλληψης γράφεται πάντα στο Load Register. Επόμενα γεγονότα σύλληψης θα ενημερώσουν το Load Register και θα γράψουν από πάνω την προηγούμενη τιμή, ανεξάρτητα από το αν έχει διαβαστεί ή όχι.
6. Ο μετρητής μπορεί να ρυθμιστεί να μετράει είτε προς τα πάνω είτε προς τα κάτω, ανάλογα με την επιλογή του bit UDT στον Control/Status Register.

3.3.2.3 Λειτουργία Διαμόρφωσης Εύρους Παλμών

Στη λειτουργία Διαμόρφωσης Εύρους Παλμών (Pulse Width Modulation ή PWM) δύο μετρητές χρησιμοποιούνται ως ζευγάρι για την παραγωγή ενός εξωτερικού σήματος, του PWM0, με μία καθορισμένη συχνότητα και ένα καθορισμένο duty factor. Στον Timer0 τίθεται η περίοδος και στον Timer1 τίθεται ο χρόνος σε μία περίοδο που η PWM0 έξοδος είναι στο 1 (high time).

Η λειτουργία PWM έχει τα εξής χαρακτηριστικά:

1. Ο τρόπος λειτουργίας τόσο για τον Timer0 όσο και για τον Timer1 πρέπει να ρυθμιστεί στο Generate Mode, δηλαδή το bit MDT του Control/Status Register πρέπει να τεθεί στο 0.
2. Το bit PWMA0 στον TCSR0 και το bit PWMB0 στον TCSR1 πρέπει να τεθούν στο 1 για την ενεργοποίηση του τρόπου λειτουργίας PWM.
3. Τα σήματα GenerateOut πρέπει να ενεργοποιηθούν στους Control/Status Registers (το bit GENT να είναι 1). Το σήμα PWM0 παράγεται από τα σήματα GenerateOut των Timer0 και Timer1 και, συνεπώς, τα σήματα αυτά πρέπει να είναι ενεργοποιημένα και στους δύο μετρητές.
4. Η στάθμη των σημάτων GenerateOut και για τους δύο μετρητές στο ζευγάρι πρέπει να είναι 1.
5. Οι μετρητές μπορούν να ρυθμιστούν να μετρούν πάνω ή κάτω.
6. Η περίοδος του PWM καθορίζεται από την τιμή παραγωγής στο Load Register του Timer0. Ο high time του PWM καθορίζεται από την τιμή παραγωγής στο Load Register του Timer1. Η περίοδος και ο high time στην περίπτωση που οι μετρητές μετρούν προς τα πάνω (UDT bit είναι 0) υπολογίζονται ως εξής:

PWM_PERIOD = (MAX_COUNT – TLR0 + 2) · OPB_CLOCK_PERIOD

PWM_HIGH_TIME = (MAX_COUNT – TLR1 + 2) · OPB_CLOCK_PERIOD

Στην περίπτωση που μετρούν προς τα κάτω (UDT bit είναι 1) υπολογίζονται ως εξής:

PWM_PERIOD = (TLR0 + 2) · OPB_CLOCK_PERIOD

PWM_HIGH_TIME = (TLR1 + 2) · OPB_CLOCK_PERIOD

3.3.3 Διακοπές

Τα σήματα διακοπών του OPB Timer/Counter μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν με το ENIT bit του Control/Status Register. Το bit κατάστασης διακοπής (TINT) στον Control/Status Register δεν μπορεί να απενεργοποιηθεί και πάντα δείχνει την τρέχουσα κατάσταση της διακοπής του μετρητή. Σε λειτουργία παραγωγής μία διακοπή μετρητή προκαλείται από το πέρασμα του μετρητή από όλα 0 σε όλα 1 ή από το πέρασμα από όλα 1 σε όλα 0. Σε λειτουργία σύλληψης το γεγονός διακοπής είναι το γεγονός σύλληψης. Τα χαρακτηριστικά των διακοπών είναι:

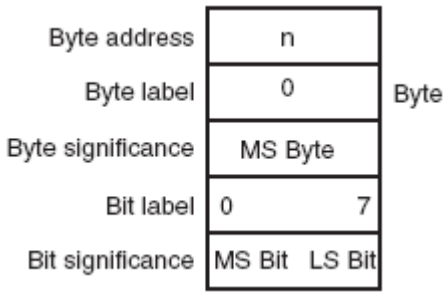
1. Τα γεγονότα διακοπών μπορούν να συμβούν μόνο όταν ο μετρητής είναι ενεργοποιημένος. Σε λειτουργία σύλληψης αυτό εμποδίζει τις διακοπές να συμβούν πριν την ενεργοποίηση του μετρητή.
2. Το σήμα διακοπής πηγαίνει ψηλά όταν ικανοποιείται η συνθήκη διακοπής και η διακοπή είναι ενεργοποιημένη στον Control/Status Register. Η διακοπή διεκδικείται όταν το σήμα διακοπής είναι ψηλά.
3. Ένα μόνο σήμα διακοπής παρέχεται. Το σήμα διακοπής είναι η λογική συνάρτηση OR των διακοπών από τους δύο μετρητές. Η ρουτίνα εξυπηρέτησης διακοπής θα πρέπει να διαβάσει τους Control/Status Registers, ώστε να καθορίσει την πηγή ή τις πηγές της διακοπής.
4. Το bit κατάστασης της διακοπής (TINT στον Control/Status Register) μπορεί μόνο να καθαριστεί με την εγγραφή ενός 1 σε αυτό. Η εγγραφή ενός 0 σε αυτό δεν έχει καμία συνέπεια στο bit. Αφού η συνθήκη διακοπής είναι μία ακμή (το πέρασμα του μετρητή ή το γεγονός σύλληψης) μπορεί να καθαριστεί οποιαδήποτε στιγμή και δε θα σημαίνει κάποια συνθήκη διακοπής μέχρι το επόμενο γεγονός διακοπής.

3.3.4 Τύποι Δεδομένων Καταχωρητών και Οργάνωση

Η πρόσβαση στους καταχωρητές του OPB Timer/Counter μπορεί να γίνει σε τρεις τύπους: byte (8 bits), halfword (2 bytes) και word (4 bytes). Όλες οι προσβάσεις στους καταχωρητές γίνονται σε όρια λέξεων (word boundaries), ώστε να υπακούν στη σύμβαση θέσης καταχωρητών OPB-IPIF. Η διεύθυνση κάθε καταχωρητή προκύπτει από την πρόσθεση της αντίστοιχης απόκλισης (offset) στη διεύθυνση βάσης του OPB Timer/Counter. Η απεικόνιση διευθύνσεων των καταχωρητών, καθώς και πληροφορίες για τους καταχωρητές (απόκλιση, μέγεθος, τύπος, περιγραφή), φαίνονται στο Σχήμα 3.16. Οι καταχωρητές του OPB Timer/Counter οργανώνονται σαν big-endian δεδομένα. Στο Σχήμα 3.17 φαίνεται ο τύπος byte.

Register	Address (Hex)	Size	Type	Description
TCSR0	0x00	Word	R/W	Control/Status Register 0
TLR0	0x04	Word	R/W	Load Register 0
TCR0	0x08	Word	R	Timer/Counter Register 0
TCSR1	0x10	Word	Read/Write	Control/Status Register 1
TLR1	0x14	Word	Read/Write	Load Register 1
TCR1	0x18	Word	Read	Timer/Counter Register 1

Σχήμα 3.16: Απεικόνιση Διευθύνσεων



Σχήμα 3.17: Ο Τύπος byte

3.3.5 Περιγραφή Καταχωρητών

3.3.5.1 Load Register (TLR0 – TLR1)

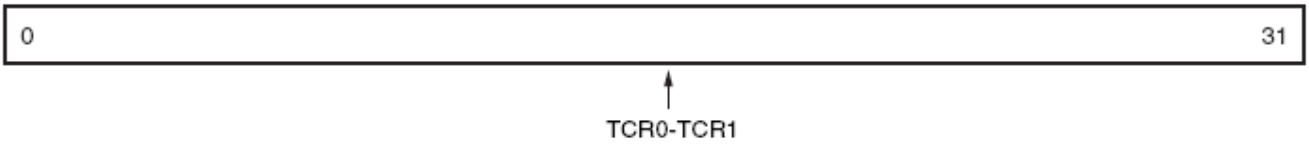
Όταν το πλάτος του μετρητή έχει ρυθμιστεί σε λιγότερα από 32 bits, η τιμή του Load Register στοιχίζεται στα δεξιά στους TLR0, TLR1. Το λιγότερο σημαντικό bit του μετρητή απεικονίζεται πάντα

στο bit 31 του Load Register. Ο Load Register φαίνεται στο Σχήμα 3.18.

Σχήμα 3.18: Timer Load Register

3.3.5.2 Timer/Counter Register (TCR0 – TCR1)

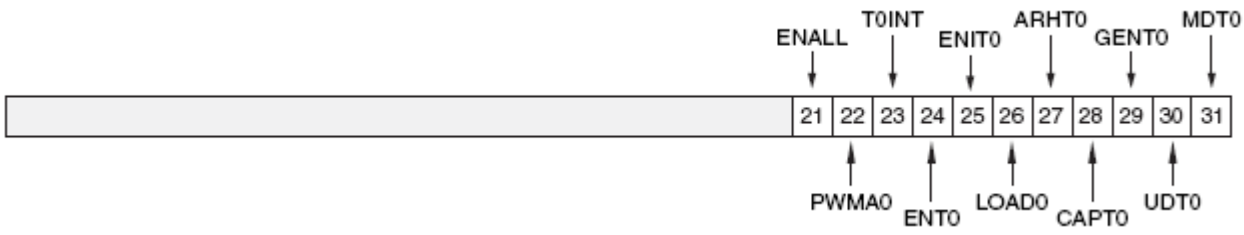
Όταν το πλάτος του μετρητή έχει ρυθμιστεί σε λιγότερα από 32 bits, η τιμή μέτρησης στοιχίζεται στα δεξιά στους TCR0, TCR1. Το λιγότερο σημαντικό bit του μετρητή πάντοτε απεικονίζεται στο bit 31 του Timer/Counter Register. Ο Timer/Counter Register φαίνεται στο Σχήμα 3.19.



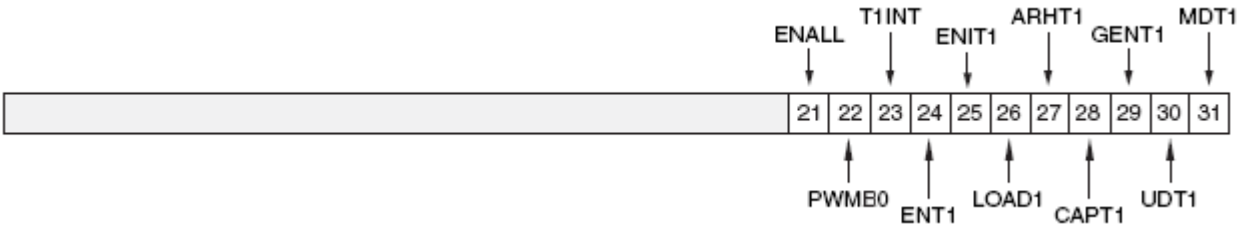
Σχήμα 3.19: Timer/Counter Register

3.3.5.3 Control/Status Register (TCSR0 – TCSR1)

Ο Control/Status Register 0 περιέχει τα bits ελέγχου και κατάστασης της μονάδας μετρητή 0 και ο Control/Status Register 1 περιέχει τα bits ελέγχου και κατάστασης της μονάδας μετρητή 1. Ο Control/Status Register 0 φαίνεται στο Σχήμα 3.20 ενώ ο Control/Status Register 1 φαίνεται στο Σχήμα 3.21.



Σχήμα 3.20: Timer Control/Status Register 0



Σχήμα 3.21: Timer Control/Status Register 1

Από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά, τα bits έχουν την εξής σημασία:

1. bit 31, MDT. Όταν έχει την τιμή 0, τότε ο μετρητής είναι στη λειτουργία παραγωγής. Όταν έχει την τιμή 1, τότε ο μετρητής είναι στη λειτουργία σύλληψης.
2. bit 30, UDT. Όταν έχει την τιμή 0, τότε ο μετρητής μετράει προς τα πάνω. Όταν έχει την τιμή 1, τότε ο μετρητής μετράει προς τα κάτω.
3. bit 29, GENT. Όταν έχει την τιμή 0, τότε απενεργοποιείται το εξωτερικό σήμα παραγωγής. Όταν έχει την τιμή 1, τότε ενεργοποιείται το εξωτερικό σήμα παραγωγής.
4. bit 28, CAPT. Όταν έχει την τιμή 0, τότε απενεργοποιείται το εξωτερικό σήμα σύλληψης. Όταν έχει την τιμή 1, τότε ενεργοποιείται το εξωτερικό σήμα σύλληψης.
5. bit 27, ARHT. Όταν έχει την τιμή 0, τότε ο μετρητής σταματά κατά το πέρασμα στη λειτουργία παραγωγής ενώ κρατάει την προηγούμενη τιμή σύλληψης στη λειτουργία σύλληψης. Όταν έχει την τιμή 1, τότε ο μετρητής κατά το πέρασμα ξαναφορτώνει την τιμή παραγωγής στη λειτουργία παραγωγής ενώ γράφει από πάνω την προηγούμενη τιμή σύλληψης στη λειτουργία σύλληψης.
6. bit 26, LOAD. Όταν έχει την τιμή 0, δε φορτώνει το μετρητή. Όταν έχει την τιμή 1, φορτώνει το μετρητή με την τιμή στον TLR.
7. bit 25, ENIT. Όταν έχει την τιμή 0, απενεργοποιεί το σήμα διακοπής. Όταν έχει την τιμή 1, ενεργοποιεί το σήμα διακοπής.
8. bit 24, ENT. Όταν έχει την τιμή 0, απενεργοποιεί το μετρητή, δηλαδή ο μετρητής σταματά. Όταν έχει την τιμή 1, ενεργοποιεί το μετρητή, δηλαδή ο μετρητής τρέχει.
9. bit 23, TINT. Δείχνει αν έχει συμβεί η συνθήκη για διακοπή στο μετρητή. Αν ο μετρητής είναι σε λειτουργία σύλληψης και είναι ενεργοποιημένος, αυτό το bit δείχνει αν έχει συμβεί σύλληψη. Αν ο μετρητής είναι σε λειτουργία παραγωγής, αυτό το bit δείχνει αν ο μετρητής έχει

περάσει από όλα 0 σε όλα 1 ή από όλα 1 σε όλα 0. Πρέπει να καθαριστεί με την εγγραφή ενός 1. Κατά την ανάγνωση η τιμή 0 σημαίνει ότι δεν έχει συμβεί διακοπή ενώ η τιμή 1 σημαίνει ότι έχει συμβεί διακοπή. Κατά την εγγραφή η τιμή 0 δεν αλλάζει την κατάσταση του bit ενώ η τιμή 1 καθαρίζει, δηλαδή μηδενίζει, το bit.

10. bit 22, PWM. Όταν έχει την τιμή 0, απενεργοποιεί τη λειτουργία Pulse Width Modulation. Όταν έχει την τιμή 1, ενεργοποιεί τη λειτουργία Pulse Width Modulation.
11. bit 21, ENALL. Αυτό το bit καθρεφτίζεται και στους δύο Control/Status Registers και χρησιμοποιείται για την ταυτόχρονη ενεργοποίηση και των δύο μετρητών. Η εγγραφή ενός 1 σε αυτό το bit θέτει τα ENALL, ENT0 και ENT1. Η εγγραφή ενός 0 καθαρίζει το ENALL αλλά δεν επηρεάζει τα ENT0 και ENT1.
12. bits 0 – 20. Δεσμευμένα.

3.4 OPB UART Lite

Το OPB UART Lite είναι μία μονάδα που συνδέεται στο On-chip Peripheral Bus και έχει τα εξής χαρακτηριστικά ενός πυρήνα UART (Universal Asynchronous Receiver Transmitter):

1. Ένα κανάλι για μετάδοση δεδομένων και ένα για λήψη δεδομένων (full duplex).
2. FIFO μετάδοσης 16 χαρακτήρων και FIFO λήψης 16 χαρακτήρων.
3. Ρυθμιζόμενο αριθμό από bits δεδομένων σε ένα χαρακτήρα (5 – 8).
4. Ισοτιμία που μπορεί να ρυθμιστεί ως περιττή ή άρτια.
5. Ρυθμιζόμενο baud rate.

Το OPB UART Lite είναι παραμετροποιήσιμο σε μεγάλο βαθμό, με σκοπό τη χρησιμοποίηση όσο το δυνατόν λιγότερων πόρων του FPGA, ανάλογα με τις απαιτήσεις της εφαρμογής. Περισσότερα data bits σημαίνουν περισσότερους χρησιμοποιούμενους πόρους, όπως το ίδιο σημαίνει και η χρήση ισοτιμίας.

3.4.1 Τύποι Δεδομένων Καταχωρητών και Οργάνωση

Η πρόσβαση στους καταχωρητές του UART Lite γίνεται σε τρεις τύπους: byte (8 bits), halfword (2 bytes) και word (4 bytes). Όλες οι προσβάσεις στους καταχωρητές γίνονται σε όρια

λέξεων (word boundaries), ώστε να υπακούν στη σύμβαση θέσης καταχωρητών OPB-IPIF. Οι διευθύνσεις των καταχωρητών του UART Lite παρέχονται στο τμήμα της Απεικόνισης Διευθύνσεων (Address Map).

Οι καταχωρητές του UART Lite οργανώνονται ως big-endian bit-reversed δεδομένα. Τόσο τα bytes, δηλαδή, όσο και τα bits ξεκινούν από το περισσότερο σημαντικό και πηγαίνουν προς το λιγότερο σημαντικό με την αρίθμηση αντίστοιχα, τόσο για τα bytes όσο και για τα bits, να ξεκινάει από το 0 και να φτάνει ως και τη μέγιστη τιμή της, την οποία καθορίζει ο εκάστοτε τύπος δεδομένων. Στο Σχήμα 3.22 φαίνεται ο τύπος δεδομένων word.

Byte address	n	n+1	n+2	n+3	
Byte label	0	1	2	3	Word
Byte significance	MS Byte			LS Byte	
Bit label	0			31	
Bit significance	MS Bit			LS Bit	

Σχήμα 3.22: Ο Τύπος Δεδομένων word

3.4.2 Καταχωρητές του UART Lite

Σε αυτό το τμήμα γίνεται η περιγραφή των καταχωρητών που χρησιμοποιούνται στον προγραμματισμό σε συμβολική γλώσσα. Οι καταχωρητές αυτοί φαίνονται στο Σχήμα 3.23.

Receive FIFO	Read Character from Receive FIFO
Transmit FIFO	Write Character into Transmit FIFO
Status	Read from Status Register
Control	Write to Control Register

Σχήμα 3.23: Το Σύνολο Καταχωρητών του OPB UART Lite

3.4.2.1 Status Register

Ο καταχωρητής κατάστασης (status register) περιέχει την κατάσταση των FIFOs λήψης και

αποστολής, αν οι διακοπές είναι ενεργοποιημένες και αν υπάρχουν σφάλματα. Από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά bits, τα bits του καταχωρητή κατάστασης έχουν την εξής σημασία:

1. bit 31, RX_FIFO_VALID_DATA. Αν η FIFO λήψης έχει έγκυρα δεδομένα, τότε έχει την τιμή 1. Αν η FIFO λήψης είναι άδεια, τότε έχει την τιμή 0.
2. bit 30, RX_FIFO_FULL. Αν η FIFO λήψης είναι γεμάτη, έχει την τιμή 1. Αν δεν είναι γεμάτη, έχει την τιμή 0.
3. bit 29, TX_FIFO_EMPTY. Αν η FIFO αποστολής είναι άδεια, έχει την τιμή 1. Αν η FIFO αποστολής δεν είναι άδεια, έχει την τιμή 0.
4. bit 28, TX_FIFO_FULL. Αν η FIFO αποστολής είναι γεμάτη, έχει την τιμή 1. Αν δεν είναι γεμάτη, έχει την τιμή 0.
5. bit 27, INTR_ENABLED. Αν η διακοπή είναι ενεργοποιημένη, έχει την τιμή 1. Αν η διακοπή δεν είναι ενεργοποιημένη, έχει την τιμή 0.
6. bit 26, OVERRUN_ERROR. Αν έχει συμβεί υπερχείλιση, δηλαδή έγινε η λήψη χαρακτήρα ενώ η FIFO λήψης ήταν γεμάτη με αποτέλεσμα την απόρριψη του χαρακτήρα, τότε έχει την τιμή 1. Διαφορετικά, αν δηλαδή δεν έχει συμβεί υπερχείλιση, έχει την τιμή 0. Το bit καθαρίζεται με την ανάγνωση του status register.
7. bit 25, FRAME_ERROR. Αν έχει ανιχνευτεί stop bit με την τιμή 0, τότε συμβαίνει απόρριψη του χαρακτήρα και το bit αυτό παίρνει την τιμή 1. Διαφορετικά έχει την τιμή 0. Το bit αυτό καθαρίζεται με την ανάγνωση του status register.
8. bit 24, PAR_ERROR. Αν έχει υπάρξει σφάλμα ισοτιμίας, τότε το bit έχει την τιμή 1. Αν δεν έχει συμβεί σφάλμα ισοτιμίας, τότε έχει την τιμή 0. Το bit καθαρίζεται με την ανάγνωση του status register. Αν η UART έχει ρυθμιστεί να μην έχει έλεγχο ισοτιμίας, τότε το bit αυτό είναι πάντοτε 0.
9. bits 0 – 23. Δεσμευμένα.

3.4.2.2 Control Register

Ο καταχωρητής ελέγχου (Control Register) ελέγχει τη UART Lite. Από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά, τα bits του καταχωρητή ελέγχου έχουν την εξής σημασία:

1. bit 31, RST_TX_FIFO. Με την εγγραφή της τιμής 1 στο bit αυτό καθαρίζεται η FIFO αποστολής. Η εγγραφή της τιμής 0 δεν έχει κανένα αποτέλεσμα.
2. bit 30, RST_RX_FIFO. Η εγγραφή της τιμής 1 στο bit καθαρίζει τη FIFO λήψης. Η εγγραφή της τιμής 0 δεν έχει κανένα αποτέλεσμα.
3. bits 28 – 29. Δεσμευμένα.
4. bit 27, ENABLE_INTR. Η εγγραφή της τιμής 1 στο bit αυτό ενεργοποιεί το σήμα διακοπής ενώ η εγγραφή της τιμής 0 απενεργοποιεί το σήμα διακοπής.
5. bits 0 – 26. Δεσμευμένα.

3.4.3 Απεικόνιση Διευθύνσεων

Κατά τη δημιουργία του συστήματος, όπως ισχύει και για όλα τα περιφερειακά και τις μνήμες που χρησιμοποιούνται στο σύστημα, δίνεται η διεύθυνση βάσης του UART, η UART_BASE_ADDRESS, και η οποία χρησιμοποιείται για την πρόσβαση στους καταχωρητές του UART και τις FIFOs αποστολής και λήψης. Με απόκλιση 0 από τη UART_BASE_ADDRESS βρίσκεται η διεύθυνση για ανάγνωση από τη FIFO λήψης. Με απόκλιση 4 από τη UART_BASE_ADDRESS βρίσκεται η διεύθυνση για εγγραφή στη FIFO αποστολής. Με απόκλιση 8 από τη UART_BASE_ADDRESS βρίσκεται η διεύθυνση για ανάγνωση από το Status Register. Με απόκλιση 12 από τη UART_BASE_ADDRESS βρίσκεται η διεύθυνση για εγγραφή στον Control Register. Αυτά συνοψίζονται στο Σχήμα 3.24.

UART_BASE_ADDRESS + 0: Read from Receive FIFO

UART_BASE_ADDRESS + 4: Write to transmit FIFO

UART_BASE_ADDRESS + 8: Read from Status Register

UART_BASE_ADDRESS + 12: Write to Control Register

Σχήμα 3.24: Απεικόνιση Διευθύνσεων

3.4.4 Διακοπές

Αν οι διακοπές είναι ενεργοποιημένες, μία διακοπή παράγεται, όταν μία από τις παρακάτω συνθήκες είναι αληθής:

1. Όταν υπάρχει έγκυρος χαρακτήρας στη FIFO λήψης, η διακοπή μένει ενεργή μέχρι η FIFO λήψης να είναι άδεια.
2. Όταν η FIFO αποστολής μεταπίπτει από την κατάσταση κατά την οποία έχει δεδομένα στην κατάσταση κατά την οποία είναι άδεια, όπως συμβαίνει όταν αποστέλλεται ο τελευταίος χαρακτήρας από την FIFO αποστολής, η διακοπή είναι ενεργή για έναν κύκλο ρολογιού.

Κεφάλαιο 4

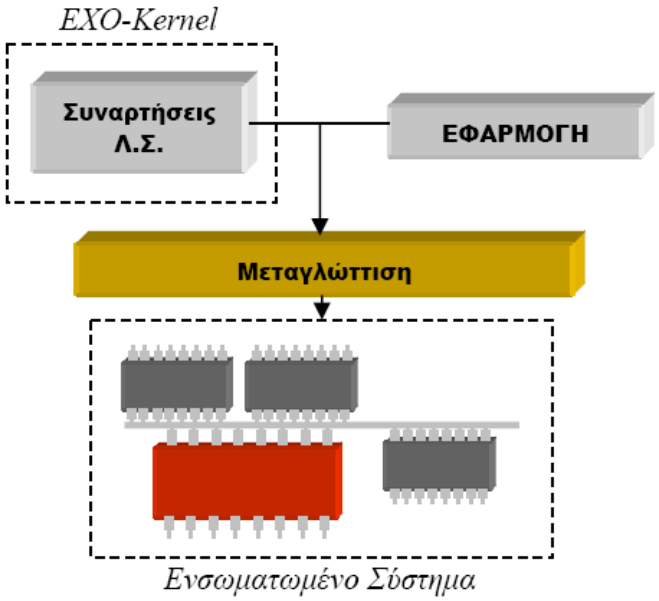
Υλοποίηση του MicroEmpix/FPGA

Σε αυτό το κεφάλαιο θα αναλυθεί πλήρως το λειτουργικό σύστημα MicroEmpix/FPGA, που έχει τροποποιηθεί στο πλαίσιο της διπλωματικής εργασίας και προορίζεται για χρήση σε ενσωματωμένα συστήματα της εταιρείας Xilinx, στα οποία γίνεται χρήση και του επεξεργαστή MicroBlaze. Θα επιχειρηθεί μία top-down προσέγγιση, που σημαίνει ότι η ανάλυση θα ξεκινήσει από τα υψηλότερα επίπεδα της υλοποίησης και θα οδηγηθεί στα χαμηλότερα. Σκοπός αυτής της προσέγγισης είναι η πλήρης κατανόηση από τον αναγνώστη του κάθε σταδίου της λειτουργίας του MicroEmpix/FPGA, πριν η ανάλυση προχωρήσει σε λεπτομέρεια. Με το πέρας αυτού του κεφαλαίου ο αναγνώστης θα μπορεί να μελετήσει με ευκολία τον κώδικα του MicroEmpix/FPGA και θα είναι ικανός, ενδεχομένως, να τον τροποποιήσει με βάση τις δικές του ανάγκες.

4.1 Φιλοσοφία Σχεδίασης του MicroEmpix/FPGA

Η φιλοσοφία σχεδίασης του MicroEmpix/FPGA βασίζεται στην πρακτική των EXO-Kernels και φαίνεται στο Σχήμα 4.1. Η πρακτική αυτή ορίζει τη δημιουργία ενός συνόλου λειτουργιών των λειτουργικών συστημάτων σε μορφή βιβλιοθήκης, οι οποίες μπορούν να χρησιμοποιούνται από τον προγραμματιστή για τη σχεδίαση της όλης εφαρμογής που το ενσωματωμένο σύστημα καλείται να επιτελέσει. Η βιβλιοθήκη αυτή αποκαλείται στη συνέχεια MicroEmpix/FPGA Application Programming Interface.

Με τη χρήση των EXO-Kernels αφενός δίνεται η δυνατότητα στον προγραμματιστή να έχει πλήρη έλεγχο του πυρήνα του λειτουργικού συστήματος κατά την υλοποίηση της ζητούμενης εφαρμογής και αφετέρου μεγιστοποιείται η απόδοση της τελικής υλοποίησης, αφού η εφαρμογή θα εκτελείται σε επίπεδο πυρήνα. Η δυνατότητα του προγραμματιστή να βρίσκεται σε επίπεδο πυρήνα, αν και επισφαλής από πλευράς σταθερότητας για το λειτουργικού σύστημα, δεν αποτελεί σοβαρό μειονέκτημα στα ενσωματωμένα συστήματα, λόγω του γεγονότος ότι ο τελικός χρήστης σπάνια έχει την ευκαιρία επαναπρογραμματισμού τους.



Σχήμα 4.1: Φιλοσοφία EXO-Kernel

4.2 Κώδικας του MicroEmpix/FPGA

Το MicroEmpix/FPGA αποτελείται από τρία αρχεία, το micro.h, το micro.c και το microasm.s. Το αρχείο micro.h περιέχει τις σταθερές του λειτουργικού συστήματος και τις δηλώσεις των συναρτήσεων που μπορεί να χρησιμοποιηθούν στις εφαρμογές, δηλαδή το Application Programming Interface (API) που παρέχει το MicroEmpix/FPGA στον προγραμματιστή του λειτουργικού. Το συγκεκριμένο header file συμπεριλαμβάνεται από το αρχείο micro.c και πρέπει να συμπεριλαμβάνεται και από τα αρχεία εφαρμογών. Το micro.c συμπεριλαμβάνει, επίσης, το αρχείο xparameters.h, το οποίο περιέχει ως σταθερές τις παραμέτρους του ενσωματωμένου συστήματος. Ανάμεσα σε αυτές τις σταθερές βρίσκονται και οι διευθύνσεις βάσης των περιφερειακών, οι οποίες χρειάζονται για τις memory mapped λειτουργίες πρόσβασης σε αυτά. Το αρχείο microasm.s περιέχει ρουτίνες γραμμένες πλήρως σε συμβολική γλώσσα του επεξεργαστή MicroBlaze που χρησιμοποιούνται για λειτουργίες χαμηλού επιπέδου. Αυτές οι ρουτίνες καλούνται αποκλειστικά και μόνο από άλλες συναρτήσεις του λειτουργικού που βρίσκονται στο αρχείο micro.c. Το micro.c περιέχει, τέλος, τόσο συναρτήσεις του MicroEmpix/FPGA που δεν είναι ορατές στον προγραμματιστή, όσο και την υλοποίηση του API που διατίθεται στον προγραμματιστή από το MicroEmpix/FPGA. Είναι γραμμένο εξ ολοκλήρου σε γλώσσα C. Έχει ακολουθηθεί η σύμβαση ότι μόνο οι συναρτήσεις που μπορούν να χρησιμοποιηθούν στις εφαρμογές (δηλαδή οι συναρτήσεις που ανήκουν στο API του MicroEmpix/FPGA) ξεκινούν με κεφαλαίο γράμμα, ενώ όλες οι υπόλοιπες συναρτήσεις, που ξεκινούν με πεζό γράμμα, είναι

συναρτήσεις του λειτουργικού συστήματος αποκλειστικά.

4.3 MicroEmpix/FPGA Application Programming Interface

Στο τέλος του micro.h δηλώνονται οι συναρτήσεις που μπορεί να χρησιμοποιήσει ο προγραμματιστής του MicroEmpix/FPGA στις εφαρμογές του. Υπενθυμίζεται ότι όλες και μόνο αυτές οι συναρτήσεις ξεκινούν με κεφαλαίο γράμμα και αποτελούν το API που προσφέρει το MicroEmpix/FPGA στον προγραμματιστή. Το αρχείο micro.h πρέπει να συμπεριλαμβάνεται στα αρχεία εφαρμογών. Οι συναρτήσεις του API είναι οι εξής:

1. *void P(int s)*. Πρόκειται για τη λειτουργία P πάνω στο σηματοφορέα *s*. Αν η τιμή του σηματοφορέα είναι θετική, τότε απλά μειώνεται κατά ένα και η τρέχουσα διεργασία συνεχίζει να εκτελείται. Διαφορετικά, η διεργασία μπλοκάρεται, μπαίνει στην ουρά αναμονής του σηματοφορέα και καλείται η συνάρτηση που είναι υπεύθυνη για τη χρονοδρομολόγηση διεργασιών, ώστε να συνεχίσει την εκτέλεσή της μία νέα διεργασία. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 167.
2. *void V(int s)*. Πρόκειται για τη λειτουργία V πάνω στο σηματοφορέα *s*. Αν καμία διεργασία δε βρίσκεται στην ουρά αναμονής του σηματοφορέα, τότε η τιμή του αυξάνεται κατά ένα. Διαφορετικά, η πρώτη διεργασία που βρίσκεται στην ουρά αναμονής του σηματοφορέα γίνεται έτοιμη για εκτέλεση, δηλαδή ξεμπλοκάρεται, και βγαίνει από την ουρά. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 168.
3. *int SER_Read(char *buf, int n)*. Όταν ο τρόπος λειτουργίας διακοπής είναι απενεργοποιημένος για τη σειριακή θύρα, τότε μπορεί να χρησιμοποιηθεί αυτή η συνάρτηση για το διάβασμα χαρακτήρων από τη σειριακή θύρα. Διαβάζει το πολύ μέχρι *n* χαρακτήρες τους οποίους αποθηκεύει στον πίνακα χαρακτήρων *buf*. Επιστρέφει το πλήθος των χαρακτήρων που διάβασε συνολικά, όταν αδειάσει η FIFO λήψης της σειριακής θύρας ή όταν έχουν ήδη διαβαστεί *n* χαρακτήρες. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 170.
4. *int SER_Write(char *buf, int n)*. Όταν ο τρόπος λειτουργίας διακοπής είναι απενεργοποιημένος για τη σειριακή θύρα, τότε μπορεί να χρησιμοποιηθεί αυτή η συνάρτηση για το γράψιμο χαρακτήρων στη σειριακή θύρα. Γράφει το πολύ μέχρι *n* χαρακτήρες τους οποίους φορτώνει από τον πίνακα χαρακτήρων *buf*. Επιστρέφει το πλήθος των χαρακτήρων που έγραψε συνολικά, όταν γεμίσει η FIFO αποστολής της σειριακής θύρας ή όταν έχουν ήδη γραφτεί *n* χαρακτήρες.

Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 171.

5. *int QSR_ReadChar(char *c)*. Όταν ο τρόπος λειτουργίας διακοπής είναι ενεργοποιημένος για τη σειριακή θύρα, τότε μπορεί να χρησιμοποιηθεί αυτή η συνάρτηση για το διάβασμα ενός χαρακτήρα από τη σειριακή θύρα, ο οποίος αποθηκεύεται στη διεύθυνση μνήμης *c*. Αν δεν υπάρχει διαθέσιμος χαρακτήρας για ανάγνωση, η συνάρτηση μπλοκάρει την τρέχουσα διεργασία μέχρι να υπάρξει διαθέσιμος χαρακτήρας για ανάγνωση. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 172.
6. *int QSR_WriteChar(char *c)*. Όταν ο τρόπος λειτουργίας διακοπής είναι ενεργοποιημένος για τη σειριακή θύρα, τότε μπορεί να χρησιμοποιηθεί αυτή η συνάρτηση για το γράψιμο ενός χαρακτήρα στη σειριακή θύρα, ο οποίος φορτώνεται από τη διεύθυνση μνήμης *c*. Αν η FIFO αποστολής της σειριακής θύρας δεν έχει αδειάσει, η συνάρτηση μπλοκάρει την τρέχουσα διεργασία μέχρι να αδειάσει η FIFO αποστολής και να μπορεί, συνεπώς, να γραφτεί ένας χαρακτήρας. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 172.
7. *void PrintStr(char *str)*. Γράφει στη σειριακή θύρα τη συμβολοσειρά *str* που παίρνει ως παράμετρο. Μπορεί να χρησιμοποιηθεί με οποιονδήποτε τρόπο λειτουργίας της σειριακής θύρας. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 172.
8. *void PrintInt(int n)*. Γράφει στη σειριακή θύρα τον ακέραιο *n* που παίρνει ως παράμετρο. Μπορεί να χρησιμοποιηθεί με οποιονδήποτε τρόπο λειτουργίας της σειριακής θύρας. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 173.
9. *int GetUartError()*. Αν επιστρέψει μηδενική τιμή, τότε δεν έχει συμβεί κάποιο σφάλμα λήψης της σειριακής θύρας από την τελευταία κλήση της συνάρτησης ή από την αρχή της εκτέλεσης του MicroEmpix/FPGA, αν η συνάρτηση δεν έχει κληθεί ποτέ από τότε. Διαφορετικά, έχει συμβεί κάποιο σφάλμα λήψης. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 174.
10. *void StartCounter(int count)*. Η συνάρτηση αυτή κάνει το μετρητή χρήστη να αρχίσει να μετράει από την τιμή *count* προς τα πάνω. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 174.
11. *int StopCounter()*. Η συνάρτηση αυτή κάνει το μετρητή χρήστη να σταματήσει τη μέτρηση και επιστρέφει την τρέχουσα τιμή του μετρητή. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 175.
12. *int FSL0_ReadInt(int *value, int *status, int control)*. Η συνάρτηση αυτή χρησιμοποιείται για την ανάγνωση ενός ακεραίου από τη διαπροσωπεία FSL0. Αν η ανάγνωση ήταν επιτυχημένη, τότε επιστρέφει 1 και ο ακέραιος βρίσκεται στη διεύθυνση *value*. Διαφορετικά επιστρέφει 0.

Στη διεύθυνση *status* βρίσκεται ένας ακέραιος του οποίου το λιγότερο σημαντικό bit έχει την τιμή 1, αν η ανάγνωση ήταν αποτυχημένη, διαφορετικά έχει την τιμή 0. Το αμέσως σημαντικότερο bit του ακεραίου αυτού έχει την τιμή 1, αν το control bit της διαπροσωπείας FSL0 βρεθεί να έχει διαφορετική τιμή από την τιμή *control*, διαφορετικά έχει την τιμή 0. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 169.

13. *int FSL0_WriteInt(int *value, int *status, int control)*. Η συνάρτηση αυτή χρησιμοποιείται για την εγγραφή του ακεραίου που βρίσκεται στη διεύθυνση *value* στη διαπροσωπεία FSL0. Αν η εγγραφή ήταν επιτυχημένη, τότε επιστρέφει 1, διαφορετικά επιστρέφει 0. Στη διεύθυνση *status* βρίσκεται ένας ακέραιος του οποίου το λιγότερο σημαντικό bit έχει την τιμή 1, αν η εγγραφή ήταν αποτυχημένη, διαφορετικά έχει την τιμή 0. Το control bit της διαπροσωπείας FSL0 λαμβάνει την τιμή *control*. Η υλοποίηση της συνάρτησης βρίσκεται στη σελίδα 169.

4.4 Σταθερές του MicroEmpix/FPGA

Στην αρχή του micro.h ορίζονται οι σταθερές του λειτουργικού συστήματος MicroEmpix/FPGA. Οι σταθερές αυτές χρησιμοποιούνται στο αρχείο micro.c, το οποίο συμπεριλαμβάνει το micro.h, και είναι οι εξής:

1. *QUANTUM*. Πρόκειται για το πλήθος των διακοπών χρονιστή που πρέπει να υποστεί μία διεργασία, ώστε να σταματήσει η εκτέλεσή της και να αρχίσει να εκτελείται μία νέα διεργασία, η οποία θα επιλεχθεί από τον αλγόριθμο χρονοδρομολόγησης. Με τη σταθερά αυτή αρχικοποιείται η μεταβλητή *preempt*, η οποία μειώνεται κατά 1 με κάθε διακοπή χρονιστή, και με αυτή ανανεώνεται η μεταβλητή, όταν κληθεί η συνάρτηση χρονοδρομολόγησης και επιλέξει μία νέα διεργασία για εκτέλεση. Έχει προκαθορισμένη τιμή 2. Η συγκεκριμένη τιμή εξασφαλίζει ότι μία διεργασία θα λάβει από 50% έως 100% του μέγιστου χρόνου εκτέλεσης που αντιστοιχεί σε οποιαδήποτε διεργασία. Η χειρότερη περίπτωση είναι, λόγου χάρη, η περίπτωση μία διεργασία να κολλάει σε κάποιον σηματοφορέα με την κλήση της συνάρτησης *P* και την ώρα της εκτέλεσης του κρίσιμου τμήματος της συνάρτησης, που περιλαμβάνει την κλήση της συνάρτησης χρονοδρομολόγησης (και, συνεπώς, την ανανέωση της τιμής της μεταβλητής *preempt*), να συμβαίνει διακοπή χρονιστή. Αυτή η διακοπή θα εξυπηρετηθεί με τη συνέχεια της νέας διεργασίας, που θα επιλέξει ο αλγόριθμος χρονοδρομολόγησης, και θα καταναλώσει αμέσως τη μία από τις δύο διακοπές χρονιστή που δικαιούται η νέα διεργασία. Αν

η σταθερά είχε την τιμή 1, στη χειρότερη αυτή περίπτωση η νέα διεργασία δε θα εκτελούνταν καθόλου, καθώς στη ρουτίνα εξυπηρέτησης διακοπής θα γινόταν αμέσως κλήση της συνάρτησης χρονοδρομολόγησης. Μεγαλύτερες τιμές της σταθεράς (με ενδεχόμενη ταυτόχρονη μείωση της περιόδου των διακοπών χρονιστή) θα εξασφαλίσουν περισσότερη δικαιοσύνη στην κατανομή του χρόνου μεταξύ των διεργασιών αλλά η ρουτίνα εξυπηρέτησης διακοπής θα εκτελείται περισσότερες φορές με αποτέλεσμα μεγαλύτερο κόστος εκτέλεσης.

2. *MAXPROC*. Πρόκειται για το μέγιστο πλήθος των διεργασιών που μπορούν να υπάρξουν ταυτόχρονα στο MicroEmpix/FPGA. Σε αυτό το πλήθος συμπεριλαμβάνεται και η διεργασία αρχικοποίησης του συστήματος που καταλήγει να είναι η system idle process.
3. *MAXSEM*. Πρόκειται για το μέγιστο πλήθος σηματοφορέων που μπορούν να χρησιμοποιηθούν από τον προγραμματιστή του MicroEmpix/FPGA και από το ίδιο το λειτουργικό. Στην περίπτωση που η σειριακή θύρα δε δουλεύει με τον τρόπο λειτουργίας διακοπών, ο προγραμματιστής μπορεί να χρησιμοποιήσει όλους τους σηματοφορείς. Στην περίπτωση που η σειριακή θύρα δουλεύει με τον τρόπο λειτουργίας διακοπών, ο προγραμματιστής δεν μπορεί να χρησιμοποιήσει τους δύο τελευταίους σηματοφορείς. Αυτοί χρησιμοποιούνται από το MicroEmpix/FPGA για τον τρόπο λειτουργίας διακοπών της σειριακής θύρας.
4. *STACKSIZE*. Πρόκειται για το μέγεθος σε ακεραίους της στοίβας που δίνεται σε κάθε διεργασία του MicroEmpix/FPGA. Εξαιρείται η αρχική διεργασία συστήματος, καθώς αυτή εκτελείται στο τμήμα στοίβας του προγράμματος.
5. *SYSTEM_PROCESS*. Πρόκειται για μία σταθερά που δείχνει, στο πεδίο *ptype* της δομής *pcb* που αντιστοιχεί σε μία διεργασία, ότι η διεργασία αυτή είναι διεργασία συστήματος.
6. *USER_PROCESS*. Πρόκειται για μία σταθερά που δείχνει, στο πεδίο *ptype* της δομής *pcb* που αντιστοιχεί σε μία διεργασία, ότι η διεργασία αυτή είναι διεργασία χρήστη.
7. *RUNNING*. Πρόκειται για μία σταθερά που δείχνει, στο πεδίο *pstate* της δομής *pcb* που αντιστοιχεί σε μία διεργασία, ότι η διεργασία αυτή εκτελείται σε κάποια χρονική στιγμή.
8. *BLOCKED*. Πρόκειται για μία σταθερά που δείχνει, στο πεδίο *pstate* της δομής *pcb* που αντιστοιχεί σε μία διεργασία, ότι η διεργασία αυτή είναι μπλοκαρισμένη σε κάποια χρονική στιγμή.
9. *READY*. Πρόκειται για μία σταθερά που δείχνει, στο πεδίο *pstate* της δομής *pcb* που αντιστοιχεί σε μία διεργασία, ότι η διεργασία αυτή είναι έτοιμη προς εκτέλεση σε κάποια χρονική στιγμή.
10. *PNMLEN*. Πρόκειται για το μέγιστο πλήθος των χαρακτήρων που απαρτίζουν το όνομα μίας

διεργασίας, συμπεριλαμβανομένου του χαρακτήρα τερματισμού μίας συμβολοσειράς.

11. *PNREGS*. Πρόκειται για το πλήθος των καταχωρητών που αποθηκεύονται και φορτώνονται για κάθε διεργασία κατά τη μεταγωγή περιεχομένου. Έχει την τιμή 37, καθώς ο MicroBlaze έχει 32 καταχωρητές γενικού σκοπού και 5 καταχωρητές ειδικού σκοπού.
12. *OSEM*. Πρόκειται για το σηματοφορέα που χρησιμοποιείται για τη λειτουργία εξόδου της σειριακής θύρας κατά τον τρόπο λειτουργίας διακοπής. Είναι ο τελευταίος σηματοφορέας όλων.
13. *ISEM*. Πρόκειται για το σηματοφορέα που χρησιμοποιείται για τη λειτουργία εισόδου της σειριακής θύρας κατά τον τρόπο λειτουργίας διακοπής. Είναι ο προτελευταίος σηματοφορέας όλων.
14. *BUFLEN*. Πρόκειται για το μέγιστο πλήθος χαρακτήρων που αποθηκεύονται από το MicroEmpix/FPGA για τη λειτουργία εισόδου της σειριακής θύρας κατά τον τρόπο λειτουργίας διακοπής.
15. *UART_INT_MODE_OFF*. Σταθερά που αντιστοιχεί στον απενεργοποιημένο τρόπο λειτουργίας διακοπής της σειριακής θύρας.
16. *UART_INT_MODE_ON*. Σταθερά που αντιστοιχεί στον ενεργοποιημένο τρόπο λειτουργίας διακοπής της σειριακής θύρας.

4.5 Μεταβλητές του MicroEmpix/FPGA

Στην αρχή του αρχείου micro.c δηλώνονται οι καθολικές μεταβλητές (global variables) που θα χρησιμοποιηθούν από τις συναρτήσεις του ίδιου αρχείου και απαιτούνται για τη λειτουργία του MicroEmpix/FPGA. Οι μεταβλητές είναι, λοιπόν, οι εξής:

1. *int uartError*. Στη μεταβλητή αυτή αποθηκεύονται τα σφάλματα που ενδέχεται να συμβούν κατά τη λήψη χαρακτήρων από τη σειριακή θύρα. Χρησιμοποιούνται τα τέσσερα λιγότερο σημαντικά bits της μεταβλητής (τα υπόλοιπα bits είναι πάντα μηδενικά ενώ και η μεταβλητή αρχικοποιείται στο μηδέν) και, από το λιγότερο σημαντικό bit προς τα περισσότερο σημαντικά, τα bits έχουν την εξής σημασία:

    1. bit 31, OVERRUN_ERROR. Αν το bit αυτό έχει την τιμή 1, τότε ελήφθη ένας χαρακτήρας από τη σειριακή θύρα ενώ η FIFO λήψης της ήταν γεμάτη. Αυτό είχε ως αποτέλεσμα την

απόρριψη του χαρακτήρα που ελήφθη. Αν έχει την τιμή 0, τότε δεν έχει συμβεί τέτοιο γεγονός και δεν έχει απορριφθεί χαρακτήρας από τη σειριακή θύρα.

2. bit 30, FRAME_ERROR. Αν το bit αυτό έχει την τιμή 1, τότε ανιχνεύτηκε από τη σειριακή θύρα stop bit με την τιμή 0. Αυτό είχε ως αποτέλεσμα την απόρριψη του χαρακτήρα που ελήφθη. Αν έχει την τιμή 0, τότε δεν έχει συμβεί τέτοιο γεγονός και δεν έχει απορριφθεί χαρακτήρας από τη σειριακή θύρα.
3. bit 29, PAR_ERROR. Αν το bit αυτό έχει την τιμή 1, τότε έχει συμβεί σφάλμα ισοτιμίας κατά τη λήψη ενός χαρακτήρα από τη σειριακή θύρα. Ο χαρακτήρας θα γραφτεί, ωστόσο, στη FIFO λήψης της σειριακής θύρας. Αν δεν έχει συμβεί σφάλμα ισοτιμίας ή αν η σειριακή θύρα έχει ρυθμιστεί ώστε να μην κάνει έλεγχο για ισοτιμία, το bit θα έχει την τιμή 0.
4. bit 28, OVERFLOW_ERROR. Αν το bit αυτό έχει την τιμή 1, τότε έχει συμβεί υπερχείλιση του buffer εισόδου της σειριακής θύρας. Ενώ, δηλαδή, υπάρχει χαρακτήρας στη FIFO λήψης της σειριακής θύρας, δεν υπάρχει χώρος στο buffer εισόδου για την τοποθέτηση του χαρακτήρα. Αυτό έχει ως αποτέλεσμα την απόρριψη του χαρακτήρα. Αν δεν έχει συμβεί υπερχείλιση του buffer ή αν η σειριακή θύρα δε δουλεύει με τον τρόπο λειτουργίας διακοπής, το bit αυτό έχει την τιμή 0.

Η τιμή της μεταβλητής *uartError* επιστρέφεται και η μεταβλητή καθαρίζεται από τη συνάρτηση *int GetUartError()*. Χρησιμοποιείται για την αποθήκευση των σφαλμάτων λήψης της σειριακής θύρας χωρίς τη διακοπή των λειτουργιών ανάγνωσης και εγγραφής είτε η σειριακή θύρα δουλεύει με τον τρόπο λειτουργίας διακοπής είτε όχι.

2. *int uartIntMode*. Στη μεταβλητή αυτή αποθηκεύεται είτε η τιμή της σταθεράς *UART_INT_MODE_OFF* είτε η τιμή της σταθεράς *UART_INT_MODE_ON*. Αυτές οι σταθερές δείχνουν αντίστοιχα ότι η σειριακή θύρα δε δουλεύει με τον τρόπο λειτουργίας διακοπής και ότι η σειριακή θύρα δουλεύει με τον τρόπο λειτουργίας διακοπής.
3. *int uartWritten*. Αυτή η μεταβλητή γίνεται 1 κάθε φορά που γίνεται εγγραφή ενός χαρακτήρα στη FIFO αποστολής της σειριακής θύρας και γίνεται 0 κάθε φορά που μέσα στη ρουτίνα εξυπηρέτησης διακοπής βρεθεί να έχει την τιμή 1 και η FIFO αποστολής βρεθεί άδεια. Επίσης, αρχικοποιείται στο μηδέν. Χρησιμοποιείται για την ανίχνευση του γεγονότος ότι η FIFO αποστολής είχε κάποιο χαρακτήρα και άδειασε κατά το τρόπο λειτουργίας διακοπής της σειριακής θύρας.

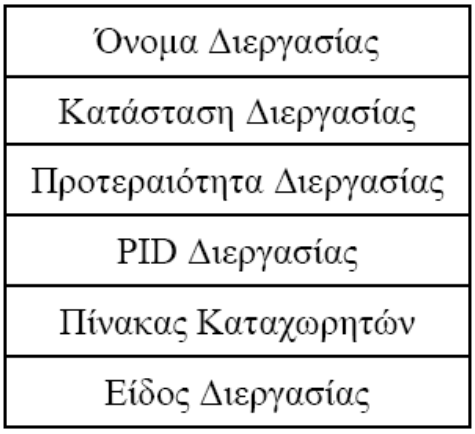
4. *char SER_InBuf[BUFLEN]*. Πρόκειται για τον buffer εισόδου της σειριακής θύρας που χρησιμοποιείται κατά τον τρόπο λειτουργίας διακοπής της σειριακής θύρας. Σε αυτόν τοποθετούνται οι χαρακτήρες που βρίσκονται στη FIFO λήψης της σειριακής θύρας μέχρι να ζητηθούν από κάποια διεργασία.
5. *char *SER_InBufPtrRd*. Πρόκειται για τον δείκτη ανάγνωσης από τον buffer εισόδου της σειριακής θύρας. Αρχικοποιείται στη διεύθυνση του buffer εισόδου.
6. *char *SER_InBufPtrWr*. Πρόκειται για τον δείκτη εγγραφής στον buffer εισόδου της σειριακής θύρας. Αρχικοποιείται στη διεύθυνση του buffer εισόδου.
7. *struct pcb {char pstate; int pid; int pprio; int pregs[PNREGS]; char pname[PNMLEN]; int ptype;} proctab[MAXPROC]*. Πρόκειται για τον πίνακα που κρατάει τις δομές *pcb* όλων των διεργασιών (συστήματος και χρήστη). Ένα Process Control Block κρατάει για τη διεργασία στην οποία αντιστοιχεί την κατάστασή της, την ταυτότητά της, την προτεραιότητά της, τους καταχωρητές της, το όνομά της και τον τύπο της.
8. *int current*. Πρόκειται για τη θέση της τρέχουσας διεργασίας που εκτελείται στον πίνακα *proctab*, που ταυτίζεται με την ταυτότητα της διεργασίας. Πρόκειται, δηλαδή, για την τρέχουσα διεργασία που εκτελείται.
9. *int numproc*. Πρόκειται για το πλήθος των διεργασιών (συστήματος και χρήστη) που βρίσκονται στο σύστημα τη συγκεκριμένη χρονική στιγμή.
10. *int stacks[MAXPROC – 1][STACKSIZE]*. Πρόκειται για τον πίνακα που περιέχει τις στοίβες των διεργασιών χρήστη. Η αρχική διεργασία συστήματος εκτελείται στο τμήμα στοίβας του προγράμματος και για αυτό το λόγο δεν της αντιστοιχεί κάποια θέση στο συγκεκριμένο πίνακα.
11. *int preempt*. Πρόκειται για τη μεταβλητή που μειώνεται κατά 1 σε κάθε διακοπή του χρονιστή συστήματος. Όταν μηδενιστεί, διακόπτεται η εκτέλεση της τρέχουσας διεργασίας που εκτελείται και επιλέγεται μία νέα διεργασία για να συνεχίσει την εκτέλεσή της. Αρχικοποιείται και ανανεώνεται με την τιμή της σταθεράς *QUANTUM* κάθε φορά που μία νέα διεργασία συνεχίζει την εκτέλεσή της.
12. *int semaphores[MAXSEM][MAXPROC]*. Πρόκειται για τον πίνακα που κρατάει τις ουρές αναμονής των σηματοφορέων. Ένα στοιχείο *semaphores[i][j]* μπορεί να έχει είτε μηδενική είτε θετική τιμή. Η μηδενική τιμή σημαίνει ότι η διεργασία *j* δεν αναμένει το σηματοφορέα *i*. Η θετική τιμή σημαίνει ότι η διεργασία *j* βρίσκεται στην αντίστοιχη θέση της ουράς αναμονής του σηματοφορέα *i*.
13. *int semavalue[MAXSEM]*. Πρόκειται για τον πίνακα που κρατάει τις τιμές των σηματοφορέων.

Μία θετική ή μηδενική τιμή σημαίνει ότι καμία διεργασία δεν αναμένει το συγκεκριμένο σηματοφορέα. Μία αρνητική τιμή σημαίνει ότι το αντίθετό της είναι το πλήθος των διεργασιών που αναμένουν το συγκεκριμένο σηματοφορέα.

4.6 Λειτουργίες του MicroEmpix/FPGA

4.6.1 Διαχείριση Διεργασιών

Το σημαντικότερο χαρακτηριστικό που πρέπει να διαθέτει ένα λειτουργικό σύστημα είναι η δυνατότητα υποστήριξης πολλών διεργασιών ταυτόχρονα (multitasking). Πρέπει, δηλαδή, να υποστηρίζει τον πολυπρογραμματισμό. Και, επειδή μία διεργασία δεν είναι απλά το πρόγραμμα που αυτή εκτελεί, χρειάζεται ειδική πρόνοια για την υποστήριξη αυτής της λειτουργίας. Η βασικότερη δομή σε ένα λειτουργικό σύστημα που υποστηρίζει τον πολυπρογραμματισμό είναι το σύνολο ελέγχου διεργασίας (process control block). Σε κάθε διεργασία στο MicroEmpix/FPGA αντιστοιχεί μία δομή *pcb*, η οποία κρατάει πληροφορίες για τη διεργασία. Όταν μία διεργασία πρέπει να διακοπεί, ώστε να συνεχίσει την εκτέλεσή της κάποια άλλη, θα πρέπει κάποιες πληροφορίες να αποθηκευτούν από το λειτουργικό σύστημα, ώστε η διεργασία που διακόπτεται να μπορεί να συνεχίσει την εκτέλεσή της αργότερα. Στο MicroEmpix/FPGA η δομή *pcb* κρατάει αυτές τις απαραίτητες πληροφορίες, οι οποίες είναι το όνομα της διεργασίας, η κατάστασή της, η προτεραιότητά της, το process identification number της, ένας πίνακας με τις τιμές των καταχωρητών της και το είδος της διεργασίας. Τα παραπάνω φαίνονται γραφικά στο Σχήμα 4.2.



Σχήμα 4.2: Process Control Block

Το MicroEmpix/FPGA διατηρεί μία δομή *pcb* για καθεμία διεργασία που υπάρχει στο σύστημα. Όλες μαζί οι δομές αυτές βρίσκονται στον πίνακα *proctab*. Επίσης, το MicroEmpix/FPGA διαθέτει μία ξεχωριστή στοίβα για κάθε διεργασία. Η στοίβες των διεργασιών βρίσκονται στον πίνακα *stacks*. Μοναδική εξαίρεση αποτελεί η διεργασία που καταλαμβάνει τη μηδενική θέση στον πίνακα *proctab*. Πρόκειται για τη διεργασία συστήματος που αρχικοποιεί ολόκληρο το λειτουργικό σύστημα και στη συνέχεια εξελίσσεται στη system idle process. Η διεργασία αυτή εκτελείται στο τμήμα στοίβας του προγράμματος. Για το λόγο αυτό ο πίνακας *stacks* έχει μέγεθος μικρότερο κατά 1 από το μέγεθος του πίνακα *proctab*. Τα μεγέθη αυτά των δύο πινάκων είναι σταθερά και γνωστά κατά τη μεταγλώττιση. Αυτό σημαίνει ότι το πλήθος των διεργασιών που μπορούν να υπάρξουν ταυτόχρονα στο MicroEmpix/FPGA περιορίζεται από μία σταθερά, τη σταθερά *MAXPROC*.

4.6.1.1 Δημιουργία Διεργασιών

Η δημιουργία των διεργασιών στο MicroEmpix/FPGA γίνεται μέσα στη συνάρτηση *int main()* με την κλήση της συνάρτησης *void create(void (*procaddr)(), int priority, char *name, int type)*. Η *create* παίρνει ως πρώτη παράμετρο τη διεύθυνση της συνάρτησης που θα εκτελέσει η διεργασία, η οποία συνάρτηση δεν παίρνει παραμέτρους και δεν επιστρέφει τιμή. Ως δεύτερη παράμετρο η *create* παίρνει την προτεραιότητα της διεργασίας, ως τρίτη παίρνει το όνομα της διεργασίας και ως τέταρτη και τελευταία παράμετρο παίρνει τον τύπο της διεργασίας.

Αν το πλήθος των διεργασιών στο σύστημα είναι ίσο με το μέγιστο επιτρεπόμενο, τότε η συνάρτηση *create* επιστρέφει αμέσως, χωρίς να δημιουργήσει τη διεργασία. Διαφορετικά, η *create* αντιστοιχίζει αρχικά στη νέα διεργασία που δημιουργείται μία δομή *pcb* στον πίνακα *proctab*, η οποία θα κρατάει τις πληροφορίες της διεργασίας. Το PID της διεργασίας γίνεται ίσο με με τη θέση του συγκεκριμένου *pcb* στον πίνακα *proctab*. Από εκεί και πέρα αποθηκεύονται στο *pcb* η κατάσταση της διεργασίας, η οποία γίνεται έτοιμη προς εκτέλεση, το όνομά της, η προτεραιότητα και ο τύπος της. Στη συνέχεια γίνονται οι απαραίτητες αρχικοποιήσεις των καταχωρητών της διεργασίας. Ο stack pointer της διεργασίας, που περιέχεται στον καταχωρητή R1, τοποθετείται αμέσως πριν την αρχή της στοίβας που αντιστοιχεί στη συγκεκριμένη διεργασία, δηλαδή στο τέλος της στοίβας που αντιστοιχεί στην επόμενη διεργασία στον πίνακα *stacks*. Υπενθυμίζεται ότι η στοίβα γεμίζει από τις μεγαλύτερες προς τις μικρότερες διευθύνσεις. Ο καταχωρητής R14 αρχικοποιείται με τη διεύθυνση της συνάρτησης που θα εκτελεί η διεργασία ενώ ο καταχωρητής R15 αρχικοποιείται με τη διεύθυνση της συνάρτησης *void intro()* μειωμένη κατά 8. Η συνάρτηση *intro* ενεργοποιεί τις διακοπές και κάνει άλμα στη διεύθυνση

που περιέχεται στον καταχωρητή R14. Ο λόγος που ο καταχωρητής R15 αρχικοποιείται με την παραπάνω τιμή βρίσκεται στο τέλος της συνάρτησης *void ctxsw(int *prevpregs, int *currpregs)*, η οποία εκτελεί τη μεταγωγή περιεχομένου. Κατά την επιστροφή της *ctxsw* γίνεται άλμα στη διεύθυνση που περιέχεται στον καταχωρητή R15 αυξημένη κατά 8. Το παραπάνω τέχνασμα χρησιμοποιείται για την πρώτη φορά που εκτελείται η εν λόγω διεργασία και είναι απαραίτητο για την ενεργοποίηση των διακοπών. Αρχικοποιούνται, τέλος, με την τιμή 0 όλοι οι καταχωρητές ειδικού σκοπού της διεργασίας εκτός από τον Program Counter, ο οποίος δεν αρχικοποιείται, καθώς δεν ενδιαφέρει η τιμή του. Το ίδιο συμβαίνει και με τους υπόλοιπους καταχωρητές που δεν αρχικοποιούνται.

4.6.1.2 Χρονοδρομολόγηση Διεργασιών

Ο πολυπρογραμματισμός σε ένα λειτουργικό σύστημα προϋποθέτει την ύπαρξη ενός μηχανισμού εναλλαγής των διεργασιών που εκτελούνται. Αυτός ο μηχανισμός ονομάζεται χρονοδρομολόγηση διεργασιών (process scheduling). Τη λειτουργία αυτή της εναλλαγής των διεργασιών στο MicroEmpix/FPGA την επιτελεί η συνάρτηση *void reschedule()*. Η *reschedule* καλείται σε δύο μόνο περιπτώσεις:

1. Στην περίπτωση που η τρέχουσα διεργασία αναγκαστεί να μπει στην ουρά αναμονής ενός σηματοφορέα με την εκτέλεση της συνάρτησης *P* πάνω στο σηματοφορέα. Η διεργασία τότε, αφού μπλοκαριστεί, καλεί τη συνάρτηση *reschedule*, αφού δεν μπορεί πλέον να συνεχίσει την εκτέλεσή της. Για να συνεχίσει την εκτέλεσή της κάποια στιγμή, θα πρέπει να εκτελεστεί από άλλες διεργασίες κάποιο πλήθος κλήσεων της συνάρτησης *V* πάνω στον ίδιο σηματοφορέα.
2. Στην περίπτωση που κατά τη διάρκεια της συνεχούς εκτέλεσης μίας διεργασίας, δηλαδή εκτέλεσης κατά την οποία δεν έχει κληθεί καμία φορά η *reschedule*, μηδενιστεί η μεταβλητή *preempt* λόγω διακοπών χρονιστή.

Η *reschedule* αρχικά ανανεώνει τη μεταβλητή *preempt*, η οποία μειώνεται κατά 1 κάθε φορά που συμβαίνει διακοπή χρονιστή, και την κάνει ίση με τη σταθερά *QUANTUM*. Στη συνέχεια αναζητά στον πίνακα διεργασιών *proctab* την επόμενη διεργασία που θα εκτελεστεί. Η αναζήτηση γίνεται κυκλικά στον πίνακα *proctab*, δηλαδή όταν φτάσει στο τέλος του πίνακα γίνεται επιστροφή στην αρχή. Αναζητείται η πρώτη διεργασία που δεν είναι μπλοκαρισμένη και αυτή γίνεται τρέχουσα. Αν η προηγούμενη διεργασία ήταν σε κατάσταση εκτέλεσης και διακόπηκε από το χρονιστή, δηλαδή δεν

μπλοκαρίστηκε σε κάποιο σηματοφορέα, τότε γίνεται έτοιμη προς εκτέλεση. Στη συνέχεια η τρέχουσα διεργασία μπαίνει σε κατάσταση εκτέλεσης. Η σειρά αυτή της τροποποίησης των καταστάσεων των δύο διεργασιών είναι πολύ σημαντική, καθώς υπάρχει το ενδεχόμενο οι δύο διεργασίες να ταυτίζονται (στην περίπτωση, βέβαια, που η *reschedule* κλήθηκε μέσα στη ρουτίνα εξυπηρέτησης διακοπής λόγω διακοπής χρονιστή, αφού μία μπλοκαρισμένη διεργασία δε θα γινόταν τρέχουσα). Εκτελείται κατόπιν η μεταγωγή περιεχομένου με την κλήση της συνάρτησης *void ctxsw(int *prevpregs, int *currpregs)* με παραμέτρους τους πίνακες καταχωρητών της προηγούμενης και της τρέχουσας διεργασίας, που ολοκληρώνει την εναλλαγή της εκτέλεσης της προηγούμενης και της τρέχουσας διεργασίας. Από τα παραπάνω προκύπτει ότι εφαρμόζεται ο αλγόριθμος χρονοδρομολόγησης round-robin στο MicroEmpix/FPGA, δηλαδή της κυκλικής εναλλαγής των διεργασιών. Πρέπει, τέλος, να σημειωθεί ότι όλες οι λειτουργίες της *reschedule* αποτελούν συνολικά ένα κρίσιμο τμήμα και πρέπει, συνεπώς, να προστατεύονται από τις διακοπές, όπως και γίνεται άλλωστε.

4.6.1.3 Μεταγωγή Περιεχομένου

Τη μεταγωγή περιεχομένου (context switching) εκτελεί η συνάρτηση *void ctxsw(int *prevpregs, int *currpregs)* που είναι γραμμένη στη συμβολική γλώσσα του MicroBlaze. Η πρώτη παράμετρος που παίρνει είναι η διεύθυνση των καταχωρητών της διεργασίας της οποίας διακόπτεται η εκτέλεση. Οι καταχωρητές βρίσκονται στο αντίστοιχο πεδίο του *pcb* της διεργασίας. Στην αρχή της *ctxsw* αποθηκεύονται στις κατάλληλες θέσεις της περιοχής της μνήμης που ξεκινάει από την παραπάνω διεύθυνση οι καταχωρητές R1 (stack pointer), R14 (return address for interrupt), R15 (return address for subroutine), R17 (return address for exceptions), R19 – R31 (non-volatile ή callee-save), καθώς και οι καταχωρητές ειδικού σκοπού MSR, EAR, ESR και FSR. Οι τιμές του καταχωρητή ειδικού σκοπού PC και των καταχωρητών R3 – R12 (volatile ή caller-save) δεν ενδιαφέρουν ενώ οι τιμές των υπολοίπων καταχωρητών είναι κοινές για όλες τις διεργασίες. Γι' αυτό και οι τελευταίες τιμές δεν αποθηκεύονται. Η δεύτερη παράμετρος που παίρνει η συνάρτηση *ctxsw* είναι η διεύθυνση των καταχωρητών της διεργασίας της οποίας συνεχίζεται η εκτέλεση. Στο τέλος της *ctxsw* φορτώνονται από τις κατάλληλες θέσεις της περιοχής της μνήμης που ξεκινάει από την παραπάνω διεύθυνση οι καταχωρητές R1 (stack pointer), R14 (return address for interrupt), R15 (return address for subroutine), R17 (return address for exceptions), R19 – R31 (non-volatile ή callee-save), καθώς και οι καταχωρητές ειδικού σκοπού MSR, EAR, ESR και FSR. Οι τιμές του καταχωρητή ειδικού σκοπού PC και των καταχωρητών R3 – R12 (volatile ή caller-save) δεν ενδιαφέρουν ενώ οι τιμές των υπολοίπων

καταχωρητών είναι κοινές για όλες τις διεργασίες. Γι' αυτό και οι τελευταίες τιμές δεν φορτώνονται.

Η συνάρτηση *ctxsw* καλείται μόνο μέσα από τη συνάρτηση *reschedule* και αποτελεί κρίσιμο τμήμα. Συνεπώς, οι διακοπές πρέπει να είναι απενεργοποιημένες κατά την εκτέλεσή της. Ιδιαίτερη προσοχή πρέπει να δοθεί στο τι γίνεται κατά την επιστροφή της *ctxsw*. Αν η διεργασία που έχει επιλεχθεί να συνεχίσει την εκτέλεσή της δεν έχει εκτελεστεί ποτέ μέχρι εκείνη τη χρονική στιγμή, τότε η επιστροφή της *ctxsw* θα γίνει στην αρχή της συνάρτησης *intro*, από αυτή θα ενεργοποιηθούν οι διακοπές και στη συνέχεια θα αρχίσει η εκτέλεση της συνάρτησης που έχει αναλάβει να εκτελέσει η διεργασία. Αν η διεργασία που έχει επιλεχθεί να συνεχίσει την εκτέλεσή της έχει εκτελεστεί τουλάχιστον μία φορά, αυτό σημαίνει ότι την τελευταία φορά που εκτελέστηκε η διεργασία κλήθηκε η *reschedule*, αφού μόνο αυτή καλεί την *ctxsw*, που εκτελεί τη μεταγωγή περιεχομένου. Σε αυτήν την περίπτωση η επιστροφή της *ctxsw* θα γίνει στο σημείο αμέσως μετά την κλήση της *ctxsw* της διεργασίας που έχει επιλεχθεί να συνεχίσει την εκτέλεσή της, μέσα στη συνάρτηση *reschedule*.

4.6.1.4 Συγχρονισμός Διεργασιών

Ο συγχρονισμός των διεργασιών στο MicroEmpix/FPGA γίνεται με τη χρήση σηματοφορέων. Το πλήθος των σηματοφορέων στο σύστημα ορίζεται από τη σταθερά *MAXSEM*. Δύο είναι οι δομές που σχετίζονται με την υλοποίηση των σηματοφορέων και πρόκειται για δύο πίνακες. Ο πρώτος είναι ο πίνακας *int semavalue[MAXSEM]* και χρησιμοποιείται για να κρατάει την τιμή του κάθε σηματοφορέα. Όταν η τιμή αυτή είναι θετική ή μηδέν για κάποιο σηματοφορέα, τότε καμία διεργασία δεν περιμένει τον συγκεκριμένο σηματοφορέα. Όταν η τιμή αυτή είναι αρνητική, τότε το αντίθετο αυτής δίνει το πλήθος των διεργασιών που περιμένουν το σηματοφορέα. Για παράδειγμα, αν το *semavalue[0]* είναι 42, τότε καμία διεργασία δεν περιμένει τον σηματοφορέα 0. Αν το *semavalue[2]* είναι -3, τότε 3 διεργασίες περιμένουν τον σηματοφορέα 2.

Ο δεύτερος πίνακας είναι ο πίνακας *int semaphores[MAXSEM][MAXPROC]* και χρησιμοποιείται για να κρατάει την ουρά κάθε σηματοφορέα. Αν τιμή ενός σηματοφορέα είναι θετική ή μηδέν, τότε η αντίστοιχη γραμμή του πίνακα θα περιέχει μόνο μηδενικά. Αν είναι αρνητική, τότε η αντίστοιχη γραμμή του πίνακα *semaphores* θα περιέχει όλους τους φυσικούς αριθμούς από το 1 μέχρι και το αντίθετο της τιμής του σηματοφορέα, ακριβώς μία φορά τον κάθε αριθμό, με τα υπόλοιπα στοιχεία της γραμμής να είναι μηδενικά. Στο παραπάνω παράδειγμα με το *semavalue[2]* να είναι -3, θα μπορούσε να ισχύει το σενάριο το *semaphores[2][5]* να είναι 3, το *semaphores[2][6]* να είναι 1 και το *semaphores[2][7]* να είναι 2. Όλα τα άλλα στοιχεία της γραμμής 2 θα είναι μηδενικά. Το παραπάνω θα

σήμαινε ότι στην ουρά αναμονής του σηματοφορέα 2 πρώτη είναι η διεργασία 6, δεύτερη είναι η διεργασία 7 και τρίτη είναι η διεργασία 5.

Δύο, επίσης, είναι και οι συναρτήσεις που σχετίζονται με τους σηματοφορείς και τη χρήση τους στις εφαρμογές του MicroEmpix/FPGA. Πρόκειται για τις υλοποιήσεις των κλασικών λειτουργιών P και V του Edsger Dijkstra. Η πρώτη λειτουργία υλοποιείται με τη συνάρτηση *void P(int s)* που παίρνει ως παράμετρο τον σηματοφορέα. Η συνάρτηση αυτή εκτελείται αδιαίρετα, δηλαδή χωρίς να μπορεί να διακοπεί, στο κρίσιμο τμήμα της και μέσα σε αυτό αρχικά μειώνει κατά 1 την τιμή του σηματοφορέα. Αν η νέα τιμή του σηματοφορέα είναι θετική ή μηδέν, τότε γίνεται έξοδος από το κρίσιμο τμήμα και από τη συνάρτηση. Αν η νέα τιμή του σηματοφορέα είναι αρνητική, τότε η τρέχουσα διεργασία μπλοκάρεται και μπαίνει στην ουρά του σηματοφορέα στη θέση την αντίθετη από τη νέα τιμή του σηματοφορέα. Στο παραπάνω παράδειγμα, αν η διεργασία 1 καλέσει τη συνάρτηση *P(2)*, το *semavalue[2]* θα γίνει -4 από -3 και το *semaphores[2][1]* θα γίνει 4 από 0. Αυτό σημαίνει ότι η διεργασία 1 μπαίνει στην τέταρτη θέση της ουράς αναμονής του σηματοφορέα 2. Αμέσως μετά καλείται η συνάρτηση *reschedule*. Όταν (και αν) κάποια στιγμή μελλοντικά ξεμπλοκαριστούν με τη σειρά οι διεργασίες 6, 7, 5 και 1, η διεργασία 1, όταν έρθει η σειρά της να εκτελεστεί, θα βγει από το κρίσιμο τμήμα και από τη συνάρτηση.

Η δεύτερη λειτουργία υλοποιείται με τη συνάρτηση *void V(int s)* που παίρνει ως παράμετρο τον σηματοφορέα. Η συνάρτηση αυτή εκτελείται αδιαίρετα στο κρίσιμο τμήμα της και μέσα σε αυτό αρχικά αυξάνει κατά 1 την τιμή του σηματοφορέα. Αν η νέα τιμή του σηματοφορέα είναι θετική, τότε γίνεται έξοδος από το κρίσιμο τμήμα και από τη συνάρτηση. Αν η νέα τιμή του σηματοφορέα είναι αρνητική ή μηδέν, που σημαίνει ότι ήταν αρνητική προηγουμένως, τότε υπάρχει τουλάχιστον μία διεργασία που αναμένει τον σηματοφορέα. Η πρώτη διεργασία στην ουρά του σηματοφορέα βγαίνει από την ουρά και γίνεται έτοιμη προς εκτέλεση. Με αναφορά στο παραπάνω παράδειγμα, έστω ότι η διεργασία 3 καλεί τη συνάρτηση *V(2)*. Τότε το *semavalue[2]* θα γίνει -3 από -4 που ήταν προηγουμένως. Στη συνέχεια θα γίνει αναζήτηση στη γραμμή 2 του πίνακα *semaphores* για τη διεργασία που είναι πρώτη στην ουρά. Θα βρεθεί ότι το *semaphores[2][6]* έχει την τιμή 1. Η διεργασία 6 θα γίνει τότε έτοιμη προς εκτέλεση. Επίσης, όλες οι θετικές τιμές της γραμμής 2 του πίνακα *semaphores* θα μειωθούν κατά 1 με αποτέλεσμα αφενός η διεργασία 6 να βγει από την ουρά και αφετέρου οι διεργασίες 7, 5 και 1 να προχωρήσουν μία θέση μπροστά. Η διεργασία 3 θα βγει, τέλος, από το κρίσιμο τμήμα και από τη συνάρτηση.

Κατά την αρχικοποίηση του συστήματος με την κλήση της συνάρτησης *initialize* που γίνεται μέσα στη συνάρτηση *main* μηδενίζονται οι πίνακες *semaphores* και *semavalue*. Συνεπώς, αρχικά οι

ουρές των σηματοφορέων είναι άδειες και οι σηματοφορείς έχουν μηδενική τιμή.

4.6.2 Οδηγοί Σειριακής Θύρας

Για τη χρήση της σειριακής θύρας μέσα από εφαρμογές το MicroEmpix/FPGA παρέχει δύο τρόπους λειτουργίας της σειριακής θύρας. Ο πρώτος είναι ο τρόπος λειτουργίας με polling και ο δεύτερος είναι ο τρόπος λειτουργίας με διακοπές.

4.6.2.1 Τρόπος Λειτουργίας με Polling

Η φιλοσοφία αυτού του τρόπου λειτουργίας είναι ότι μία διεργασία με τη χρήση των κατάλληλων συναρτήσεων του MicroEmpix/FPGA έχει άμεση πρόσβαση στις FIFOs λήψης και αποστολής της UART. Όταν επιθυμεί να λάβει κάποιο πλήθος χαρακτήρων, θα τους διαβάσει απευθείας από τη FIFO λήψης, αν υπάρχουν διαθέσιμοι χαρακτήρες στη FIFO λήψης. Όταν επιθυμεί να στείλει κάποιο πλήθος χαρακτήρων, θα τους γράψει απευθείας στη FIFO αποστολής, αν δεν είναι γεμάτη η FIFO αποστολής. Υπάρχουν δύο συναρτήσεις του MicroEmpix/FPGA για τον τρόπο λειτουργίας με polling, η *SER_Read* και η *SER_Write*.

Η συνάρτηση *int SER_Read(char *buf, int n)* χρησιμοποιείται για την απευθείας ανάγνωση χαρακτήρων από τη FIFO λήψης. Παίρνει δύο παραμέτρους και επιστρέφει μία ακέραιη τιμή. Η πρώτη παράμετρος είναι ένας πίνακας χαρακτήρων και αποτελεί τον πίνακα στον οποίο θα αποθηκευτούν οι χαρακτήρες που θα διαβαστούν από τη FIFO λήψης. Η δεύτερη παράμετρος είναι ένας ακέραιος αριθμός και αποτελεί το μέγιστο πλήθος των χαρακτήρων που θα επιχειρηθεί να διαβαστούν από τη FIFO λήψης. Η συνάρτηση *SER_Read* επιστρέφει το πλήθος των χαρακτήρων που τελικά διαβάστηκαν και αυτό συμβαίνει όταν διαβαστούν *n* χαρακτήρες ή όταν αδειάσει η FIFO λήψης. Αρχικά η *SER_Read* διαβάζει τον καταχωρητή κατάστασης της σειριακής θύρας και ενημερώνει τη μεταβλητή *uartError* για ενδεχόμενα σφάλματα λήψης. Στη συνέχεια, αν υπάρχει χαρακτήρας στη FIFO λήψης, τον διαβάζει και τον αποθηκεύει στον πίνακα *buf*. Αν δεν υπάρχει χαρακτήρας στη FIFO λήψης, η συνάρτηση επιστρέφει. Πριν την ανάγνωση του καταχωρητή κατάστασης η συνάρτηση θα είχε ήδη επιστρέψει, αν είχαν συμπληρωθεί οι *n* χαρακτήρες που έπρεπε να διαβαστούν. Κρίσιμο είναι το τμήμα που περιλαμβάνει την ανάγνωση του καταχωρητή κατάστασης και την ενδεχόμενη, ανάλογα με την ένδειξη του καταχωρητή, ανάγνωση από τη FIFO λήψης, καθώς η κατάσταση της UART δεν πρέπει να αλλάξει στο ενδιάμεσο χρονικό διάστημα, και για αυτό το λόγο προστατεύεται από διακοπές.

Η συνάρτηση *int SER_Write(char *buf, int n)* χρησιμοποιείται για την απευθείας εγγραφή χαρακτήρων στη FIFO αποστολής. Παίρνει δύο παραμέτρους και επιστρέφει μία ακέραιη τιμή. Η πρώτη παράμετρος είναι ένας πίνακας χαρακτήρων και αποτελεί τον πίνακα από τον οποίο θα φορτωθούν οι χαρακτήρες που θα γραφτούν στη FIFO αποστολής. Η δεύτερη παράμετρος είναι ένας ακέραιος αριθμός και αποτελεί το μέγιστο πλήθος των χαρακτήρων που θα επιχειρηθεί να γραφτούν στη FIFO αποστολής. Η συνάρτηση *SER_Write* επιστρέφει το πλήθος των χαρακτήρων που τελικά γράφτηκαν και αυτό συμβαίνει όταν γραφτούν *n* χαρακτήρες ή όταν γεμίσει η FIFO λήψης. Αρχικά η *SER_Write* διαβάζει τον καταχωρητή κατάστασης της σειριακής θύρας και ενημερώνει τη μεταβλητή *uartError* για ενδεχόμενα σφάλματα λήψης. Στη συνέχεια, αν δεν είναι γεμάτη η FIFO αποστολής, φορτώνει ένα χαρακτήρα από τον πίνακα *buf* και τον γράφει στη FIFO αποστολής. Ενημερώνεται, επίσης, η μεταβλητή *uartWritten* ότι έγινε εγγραφή στη FIFO αποστολής. Αν είναι γεμάτη η FIFO αποστολής, η συνάρτηση επιστρέφει. Πριν την ανάγνωση του καταχωρητή κατάστασης η συνάρτηση θα είχε ήδη επιστρέψει, αν είχαν συμπληρωθεί οι *n* χαρακτήρες που έπρεπε να γραφτούν. Κρίσιμο είναι το τμήμα που περιλαμβάνει την ανάγνωση του καταχωρητή κατάστασης και την ενδεχόμενη, ανάλογα με την ένδειξη του καταχωρητή, εγγραφή στη FIFO αποστολής, καθώς η κατάσταση της UART δεν πρέπει να αλλάξει στο ενδιάμεσο χρονικό διάστημα, και για αυτό το λόγο προστατεύεται από διακοπές.

4.6.2.2 Τρόπος Λειτουργίας με Διακοπές

Η φιλοσοφία αυτού του τρόπου λειτουργίας είναι ότι αφενός μία διεργασία δε χρειάζεται να προσπαθεί να διαβάσει ένα χαρακτήρα από την άδεια FIFO λήψης της σειριακής θύρας, αν το λειτουργικό μπορεί να την ειδοποιήσει ότι ένας χαρακτήρας είναι διαθέσιμος, και αφετέρου μία διεργασία δε χρειάζεται να προσπαθεί να γράψει ένα χαρακτήρα στη γεμάτη FIFO αποστολής της σειριακής θύρας, αν το λειτουργικό μπορεί να την ειδοποιήσει ότι η FIFO αποστολής είναι άδεια. Επίσης, αν καμία διεργασία δε διαβάσει για κάποιο χρονικό διάστημα από τη FIFO λήψης της σειριακής θύρας και για το ίδιο χρονικό διάστημα έρχονται συνεχώς νέοι χαρακτήρες, από ένα σημείο και μετά θα αρχίσουν να χάνονται χαρακτήρες, καθώς η FIFO λήψης έχει πεπερασμένο μέγεθος. Στον τρόπο λειτουργίας διακοπής της σειριακής θύρας χρησιμοποιούνται ένας σηματοφορέας εισόδου, ένας σηματοφορέας εξόδου, ένας buffer εισόδου, ένας δείκτης ανάγνωσης από το buffer εισόδου και ένας δείκτης εγγραφής στο buffer εισόδου. Υπάρχουν δύο συναρτήσεις του MicroEmpix/FPGA για τον τρόπο λειτουργίας με διακοπές, η *QSR_ReadChar* και η *QSR_WriteChar*.

Η συνάρτηση *int QSR_ReadChar(char *c)* παίρνει ως παράμετρο τη διεύθυνση της μνήμης στην οποία θα αποθηκευτεί ο χαρακτήρας που θα διαβαστεί από το buffer εισόδου της σειριακής θύρας. Στην αρχή της *QSR_ReadChar* καλείται η συνάρτηση *P* πάνω στο σηματοφορέα εισόδου της σειριακής θύρας. Η διεργασία κολλάει, ενδεχομένως, μέχρι να υπάρξει διαθέσιμος χαρακτήρας για ανάγνωση από το buffer εισόδου. Στη συνέχεια, αφού έχει εξασφαλιστεί με τη χρήση του σηματοφορέα ότι ο buffer εισόδου έχει χαρακτήρα για ανάγνωση, αποθηκεύεται στη διεύθυνση *c* ο χαρακτήρας στον οποίο δείχνει ο δείκτης ανάγνωσης από το buffer εισόδου και ο δείκτης μετακινείται μία θέση προς τα δεξιά στον θεωρητικά κυκλικό buffer. Κατόπιν η συνάρτηση *QSR_ReadChar* επιστρέφει.

Η συνάρτηση *int QSR_WriteChar(char *c)* παίρνει ως παράμετρο τη διεύθυνση της μνήμης από την οποία θα φορτωθεί ο χαρακτήρας που θα γραφτεί στη FIFO αποστολής της σειριακής θύρας. Στην αρχή της *QSR_WriteChar* καλείται η συνάρτηση *P* πάνω στο σηματοφορέα εξόδου της σειριακής θύρας. Η διεργασία κολλάει, ενδεχομένως, μέχρι να αδειάσει η FIFO αποστολής. Στη συνέχεια, αφού έχει εξασφαλιστεί με τη χρήση του σηματοφορέα ότι η FIFO αποστολής είναι άδεια, καλείται η συνάρτηση *SER_Write* για την εγγραφή του χαρακτήρα στη σειριακή θύρα και κατόπιν η *QSR_WriteChar* επιστρέφει.

Οι κλήσεις της συνάρτησης *V* πάνω στους σηματοφορείς εισόδου και εξόδου της σειριακής θύρας γίνονται μέσα στη ρουτίνα εξυπηρέτησης διακοπής, όταν ανιχνευτεί διακοπή της σειριακής θύρας. Η κλήση της συνάρτησης *V* πάνω στο σηματοφορέα εισόδου γίνεται κάθε φορά που ένας χαρακτήρας διαβάζεται από τη FIFO λήψης και τοποθετείται στο buffer εισόδου της σειριακής θύρας. Η κλήση της συνάρτησης *V* πάνω στο σηματοφορέα εξόδου γίνεται κάθε φορά που αδειάζει η FIFO αποστολής της σειριακής θύρας.

4.6.2.3 Γενικές Συναρτήσεις της Σειριακής Θύρας

Το MicroEmpix/FPGA διαθέτει, επίσης, τρεις συναρτήσεις οι οποίες μπορούν να χρησιμοποιηθούν και με τους δύο τρόπους λειτουργίας της σειριακής θύρας.

Η συνάρτηση *void PrintStr(char *str)* γράφει στη σειριακή θύρα τη συμβολοσειρά *str*. Γράφεται ένας χαρακτήρας από τη συμβολοσειρά κάθε φορά και χρησιμοποιείται είτε η συνάρτηση *SER_Write* είτε η συνάρτηση *QSR_WriteChar*, ανάλογα με την τιμή της μεταβλητής *uartIntMode*, που δείχνει ποιος τρόπος λειτουργίας της σειριακής θύρας χρησιμοποιείται.

Η συνάρτηση *void PrintInt(int n)* γράφει στη σειριακή θύρα τον ακέραιο *n*. Αφού γίνει η

μετατροπή του ακεραίου σε συμβολοσειρά, στη συνέχεια καλείται η συνάρτηση *PrintStr* με παράμετρο τη συμβολοσειρά αυτή.

Η συνάρτηση *int GetUartError()* μηδενίζει τη μεταβλητή *uartError*, που δείχνει αν έχουν συμβεί σφάλματα λήψης και ποια σφάλματα λήψης έχουν συμβεί, και επιστρέφει την προηγούμενη τιμή της, δηλαδή την τιμή που είχε πριν το μηδενισμό. Για τη σωστή λειτουργία της συνάρτησης τα παραπάνω αποτελούν ένα κρίσιμο τμήμα.

4.6.3 Οδηγοί Μετρητή

Το MicroEmpix/FPGA διαθέτει δύο συναρτήσεις για την πρόσβαση σε ένα μετρητή χρήστη, ο οποίος μπορεί να χρησιμοποιηθεί για τον έλεγχο της απόδοσης του συστήματος με τη λήψη των κατάλληλων μετρήσεων. Ο μετρητής χρήστη βρίσκεται μαζί με το χρονιστή συστήματος στο περιφερειακό OPB Timer/Counter. Δεν υπάρχει, ωστόσο, καμία αλληλεπίδραση μεταξύ τους και η λειτουργία του ενός είναι εντελώς ανεξάρτητη από τη λειτουργία του άλλου. Ο μετρητής χρήστη, σε αντίθεση με το χρονιστή συστήματος, δεν παράγει διακοπές. Οι συναρτήσεις που διατίθενται για τον έλεγχο του μετρητή χρήστη είναι η *StartCounter* και η *StopCounter*.

Η συνάρτηση *void StartCounter(int count)* παίρνει ως παράμετρο την τιμή εκκίνησης μέτρησης του μετρητή. Η *StartCounter* αρχικά φορτώνει την τιμή εκκίνησης στο Load Register του μετρητή. Στη συνέχεια φορτώνει την τιμή του Load Register στον Timer/Counter Register του μετρητή και, τέλος, κάνει το μετρητή να αρχίσει να τρέχει. Για τη σωστή λειτουργία του μετρητή η παραπάνω διαδικασία αποτελεί ένα κρίσιμο τμήμα.

Η συνάρτηση *int StopCounter()* αρχικά κάνει το μετρητή να σταματήσει να τρέχει, στη συνέχεια διαβάζει την τιμή του Timer/Counter Register και, τέλος, την επιστρέφει. Για τη σωστή λειτουργία του μετρητή η παραπάνω διαδικασία αποτελεί ένα κρίσιμο τμήμα.

4.6.4 Οδηγοί Διαπροσωπείας FSL0

Το MicroEmpix/FPGA διαθέτει δύο συναρτήσεις για την πρόσβαση στη διαπροσωπεία Fast Simplex Link 0 (FSL0). Οι συναρτήσεις αυτές είναι η *FSL0_ReadInt* και η *FSL0_WriteInt*.

Η συνάρτηση *int FSL0_ReadInt(int *value, int *status, int control)* χρησιμοποιείται για την ανάγνωση ενός ακεραίου (μίας λέξης) από τη διαπροσωπεία FSL0. Η τρίτη παράμετρος που παίρνει είναι ο τύπος, data ή control, του ακεραίου που αναμένεται να διαβαστεί. Αν είναι 0, τότε αναμένεται

να διαβαστεί ακέραιος τύπου data. Διαφορετικά, αναμένεται να διαβαστεί ακέραιος τύπου control. Αν διαφωνεί το control bit της διαπροσωπείας FSL0 με αυτό που αναμενόταν, τότε το δεύτερο λιγότερο σημαντικό bit του ακεραίου που βρίσκεται στη διεύθυνση *status* θα έχει την τιμή 1, διαφορετικά θα έχει την τιμή 0. Αν η ανάγνωση είναι επιτυχημένη, τότε στη διεύθυνση *value* βρίσκεται ο ακέραιος που διαβάστηκε, το πρώτο λιγότερο σημαντικό bit του ακεραίου που βρίσκεται στη διεύθυνση *status* θα έχει την τιμή 0 και η συνάρτηση θα επιστρέψει 1. Διαφορετικά, το πρώτο λιγότερο σημαντικό bit του ακεραίου που βρίσκεται στη διεύθυνση *status* θα έχει την τιμή 1 και η συνάρτηση θα επιστρέψει 0. Για τη σωστή λειτουργία του οδηγού του FSL0 η παραπάνω διαδικασία αποτελεί ένα κρίσιμο τμήμα.

Η συνάρτηση *int FSL0_WriteInt(int *value, int *status, int control)* χρησιμοποιείται για την εγγραφή ενός ακεραίου (μίας λέξης) στη διαπροσωπεία FSL0. Η πρώτη παράμετρος που παίρνει είναι η διεύθυνση από όπου θα φορτωθεί ο ακέραιος που θα γραφτεί στο FSL0. Η τρίτη παράμετρος που παίρνει η *FSL0_WriteInt* είναι ο τύπος, data ή control, του ακεραίου που θα γραφτεί. Αν είναι 0, τότε θα γραφτεί ακέραιος τύπου data και το control bit του FSL0 θα γίνει 0. Διαφορετικά θα γραφτεί ακέραιος τύπου control και το control bit του FSL0 θα γίνει 1. Αν η εγγραφή είναι επιτυχημένη, τότε το λιγότερο σημαντικό bit του ακεραίου που βρίσκεται στη διεύθυνση *status* θα έχει την τιμή 0 και η συνάρτηση θα επιστρέψει 1. Διαφορετικά, το λιγότερο σημαντικό bit του ακεραίου που βρίσκεται στη διεύθυνση *status* θα έχει την τιμή 1 και η συνάρτηση θα επιστρέψει 0. Για τη σωστή λειτουργία του οδηγού του FSL0 η παραπάνω διαδικασία αποτελεί ένα κρίσιμο τμήμα.

4.6.5 Αρχικοποίηση του Συστήματος

Η εκτέλεση του MicroEmpix/FPGA ξεκινάει από τη συνάρτηση *int main()*. Όταν αρχίζει η εκτέλεση της *main*, οι διακοπές του συστήματος είναι απενεργοποιημένες. Πρέπει, συνεπώς, όταν κριθεί απαραίτητο, οι διακοπές να ενεργοποιηθούν ρητά και αυτό γίνεται στο τέλος της *main*. Στην αρχή της *main* καλείται η συνάρτηση *void initialize(int count, int intMode)*, η οποία κάνει τις απαραίτητες αρχικοποιήσεις για τη λειτουργία του MicroEmpix/FPGA. Η συνάρτηση *initialize* παίρνει δύο παραμέτρους. Η πρώτη παράμετρος είναι η περίοδος σε κύκλους ρολογιού των διακοπών του χρονιστή του συστήματος. Η δεύτερη παράμετρος καθορίζει τον τρόπο λειτουργίας της σειριακής θύρας, δηλαδή καθορίζει αν η σειριακή θύρα δουλεύει με τον τρόπο λειτουργίας διακοπών ή αν δουλεύει με τον τρόπο λειτουργίας polling.

Στην αρχή της *initialize* αρχικοποιούνται οι σηματοφορείς του συστήματος. Συγκεκριμένα γίνονται μηδενικές οι τιμές τους ενώ αδειάζουν και οι ουρές αναμονής τους. Αυτό επιτυγχάνεται με

τον μηδενισμό των στοιχείων των πινάκων *semaphores* και *semavalue*. Στη συνέχεια προσδιορίζονται τα απαραίτητα στοιχεία για την ομαλή ενσωμάτωση της τρέχουσας διεργασίας στο σύστημα. Συγκεκριμένα η μεταβλητή *numproc* παίρνει την τιμή 1 ενώ η μεταβλητή *current* παίρνει την τιμή 0. Το σημαντικότερο στοιχείο που πρέπει να προσδιοριστεί στη δομή *pcb* της τρέχουσας διεργασίας είναι η κατάστασή της, δηλαδή πρέπει να προσδιοριστεί ότι εκτελείται. Η τρέχουσα διεργασία, αφού επιτελέσει το έργο της αρχικοποίησης του συστήματος και ενεργοποιήσει τις διακοπές του συστήματος θα καταλήξει να εκτελεί έναν άεργο ατέρμονα βρόχο. Στη συνέχεια, δηλαδή, θα επιτελέσει το ρόλο της system idle process. Στο MicroEmpix/FPGA, συνεπώς, καθορίζεται με τον παραπάνω τρόπο ότι υπάρχει ήδη μία διεργασία, αυτή που εκτελείται τη συγκεκριμένη χρονική στιγμή, η οποία καταλαμβάνει τη θέση 0 στον πίνακα *proctab* που περιέχει τις δομές *pcb* των διεργασιών. Μία ακόμα ιδιαιτερότητα που έχει η τρέχουσα διεργασία είναι ότι η στοίβα της είναι η στοίβα του προγράμματος. Οι στοίβες, αντίθετα, όλων των υπολοίπων διεργασιών και οι οποίες θα δημιουργηθούν αργότερα, βρίσκονται στον πίνακα *stacks*. Για αυτό το λόγο ο πίνακας *stacks* είναι σε μέγεθος κατά 1 μικρότερος από τον πίνακα *proctab*.

Στη συνέχεια της *initialize* αρχικοποιείται η μεταβλητή *preempt*, η οποία μειώνεται κατά 1 σε κάθε διακοπή χρονιστή, με τη σταθερά *QUANTUM*. Κατόπιν ρυθμίζεται ο χρονιστής, ώστε να παράγει περιοδικές διακοπές με περίοδο ίση με την τιμή της πρώτης παραμέτρου της συνάρτησης. Η *initialize* συνεχίζεται με το μηδενισμό της μεταβλητής *uartError*, η οποία κρατάει τα σφάλματα της UART, και με το άδειασμα των FIFOs λήψης και αποστολής της UART. Στη συνέχεια, αν η δεύτερη παράμετρος της συνάρτησης ορίζει ότι η UART θα δουλεύει με τον τρόπο λειτουργίας polling, τότε απενεργοποιούνται οι διακοπές στη UART και ενεργοποιείται μόνο η διακοπή του χρονιστή στον ελεγκτή διακοπών. Διαφορετικά, ενεργοποιούνται οι διακοπές στη UART και ενεργοποιούνται τόσο η διακοπή του χρονιστή όσο και η διακοπή της UART στον ελεγκτή διακοπών. Επίσης, στη δεύτερη περίπτωση μηδενίζεται η μεταβλητή *uartWritten*, που δείχνει αν έχει γραφτεί κάτι στη σειριακή θύρα, αρχικοποιούνται οι δείκτες ανάγνωσης και εγγραφής από το buffer εισόδου της σειριακής θύρας στην αρχή του buffer και εκτελείται η συνάρτηση *V* πάνω στο σηματοφορέα εξόδου της σειριακής θύρας, που σημαίνει ότι μία διεργασία μπορεί να γράψει στη σειριακή θύρα. Σε κάθε περίπτωση ενημερώνεται η μεταβλητή *uartIntMode* με το αν η σειριακή θύρα δουλεύει ή όχι με τον τρόπο λειτουργίας διακοπής.

Στο τέλος της *initialize* ενεργοποιείται η αίτηση διακοπής και οι διακοπές υλικού στον ελεγκτή διακοπών και ο χρονιστής αρχίζει να μετράει. Υπενθυμίζεται ότι οι διακοπές του MicroBlaze είναι ακόμα απενεργοποιημένες, δηλαδή το bit Interrupt Enable του Machine Status Register του

MicroBlaze είναι ακόμα 0.

Στη συνέχεια της *main* καλούνται οι συναρτήσεις αρχικοποίησης του χρήστη και δημιουργούνται οι διεργασίες του χρήστη με κλήσεις της συνάρτησης *create*. Αμέσως μετά ενεργοποιούνται οι διακοπές του MicroBlaze με την κλήση της συνάρτησης *enable_all_intr*. Τέλος, εκτελείται ένας άεργος ατέρμων βρόχος, χαρακτηριστικός μίας idle process. Με τα παραπάνω ολοκληρώθηκε η αρχικοποίηση του συστήματος.

4.6.6 Κρίσιμα Τμήματα

Ένα κρίσιμο τμήμα (critical section) είναι ένα τμήμα κώδικα που πρέπει να εκτελεστεί αδιαίρετα, δηλαδή αν ξεκινήσει να εκτελείται δεν πρέπει να διακοπεί πριν ολοκληρωθεί η εκτέλεση ολόκληρου του τμήματος. Κρίσιμα τμήματα παρουσιάζονται σε όλες τις περιπτώσεις σχεδίασης που θα ακολουθήσουν, από τη διαχείριση διεργασιών μέχρι και τους οδηγούς της σειριακής θύρας. Τα κρίσιμα τμήματα προστατεύονται έχοντας τις διακοπές απενεργοποιημένες κατά τη διάρκεια εκτέλεσής τους. Για το σκοπό αυτό και για τον γενικότερο έλεγχο των διακοπών έχουν υλοποιηθεί και χρησιμοποιηθεί οι εξής συναρτήσεις στο MicroEmpix/FPGA:

1. *void enable_all_intr()*. Η συνάρτηση αυτή ενεργοποιεί τις διακοπές θέτοντας το δεύτερο λιγότερο σημαντικό bit του Machine Status Register του επεξεργαστή MicroBlaze, το bit Interrupt Enable.
2. *void disable_all_intr()*. Η συνάρτηση αυτή απενεργοποιεί τις διακοπές καθαρίζοντας το δεύτερο λιγότερο σημαντικό bit του Machine Status Register του επεξεργαστή MicroBlaze, το bit Interrupt Enable.
3. *void storemsr(int *destination)*. Η συνάρτηση αυτή παίρνει ως όρισμα τη διεύθυνση ενός ακεραίου και σε αυτή τη διεύθυνση αποθηκεύει τον Machine Status Register.
4. *void loadmsr(int *source)*. Η συνάρτηση αυτή παίρνει ως όρισμα τη διεύθυνση ενός ακεραίου και από αυτή τη διεύθυνση φορτώνει τον Machine Status Register.

Εκ πρώτης όψεως θα μπορούσε κάποιος να υποθέσει ότι η προστασία ενός κρίσιμου τμήματος μπορεί να γίνει απλά με τον εξής τρόπο:

...

```
disable_all_intr();
// Critical Section Starts Here
...
// Critical Section Ends Here
enable_all_intr();
...
```

Το πρόβλημα με αυτή τη λύση είναι ότι δε λαμβάνει υπόψη της την περίπτωση φωλιασμένων κρίσιμων τμημάτων. Ως παράδειγμα μπορεί να ληφθεί το εξής:

```
...
disable_all_intr();
// Critical Section One Starts Here
...
disable_all_intr();
// Critical Section Two Starts Here
...
// Critical Section Two Ends Here
enable_all_intr();
...
// Critical Section One Ends Here
enable_all_intr();
...
```

Σε αυτή την περίπτωση μετά το τέλος του δεύτερου κρίσιμου τμήματος θα ενεργοποιηθούν οι διακοπές και θα είναι ενεργοποιημένες μέχρι το τέλος του πρώτου κρίσιμου τμήματος. Οπότε θα μπορούσε να γίνει κάποιο σφάλμα στην εκτέλεση του πρώτου κρίσιμου τμήματος, καθώς η εκτέλεση θα μπορούσε να διακοπεί πριν ολοκληρωθεί. Το πρόβλημα μπορεί να λυθεί με τον εξής πολύ απλό τρόπο:

```
...
int msr;
...
```

storemsr(&msr);
disable_all_intr();
// Critical Section Starts Here
...
// Critical Section Ends Here
loadmsr(&msr);
...

Με αυτή τη λύση αφενός απενεργοποιούνται οι διακοπές καθ' όλη τη διάρκεια της εκτέλεσης του κρίσιμου τμήματος και αφετέρου γίνεται τελικά επαναφορά στην κατάσταση που υπήρχε πριν την απενεργοποίηση των διακοπών. Αν οι διακοπές ήταν ενεργοποιημένες ενεργοποιούνται ξανά ενώ αν ήταν απενεργοποιημένες μένουν απενεργοποιημένες.

4.6.7 Εξυπηρέτηση Διακοπών

Στην αρχή του αρχείου micro.c υπάρχει η εξής δήλωση:

void interruptServiceRoutine() __attribute__((interrupt_handler));

Με το *attribute interrupt_handler* δηλώνεται ότι η συνάρτηση *void interruptServiceRoutine()* θα είναι η ρουτίνα εξυπηρέτησης διακοπής του προγράμματος. Το γεγονός αυτό σημαίνει ότι η συνάρτηση αυτή θα μεταγλωττιστεί με ειδικό τρόπο που διαφέρει από τον τρόπο με τον οποίο θα μεταγλωττιστούν οι υπόλοιπες συναρτήσεις. Κατά τη μεταγλώττιση, ο μεταγλωττιστής παράγει τον κώδικα για αυτή τη συνάρτηση, ώστε αφενός να επιστρέφει ενεργοποιώντας ταυτόχρονα τις διακοπές και αφετέρου να σώζει στην αρχή και να επαναφέρει στο τέλος το περιεχόμενο όλων των καταχωρητών. Επίσης, ο μεταγλωττιστής παράγει κώδικα για την αρχικοποίηση της διεύθυνσης της μνήμης όπου συμβαίνει άλμα, όταν ανιχνευτεί διακοπή, ώστε να περιέχονται εκεί οι εντολές που θα οδηγήσουν την εκτέλεση στη ρουτίνα εξυπηρέτησης της διακοπής.

Όταν ανιχνευτεί διακοπή, ο MicroBlaze σταματάει την κανονική ροή του προγράμματος, απενεργοποιεί τις διακοπές και κάνει άλμα στη διεύθυνση 0x10. Ακολουθεί άλμα στη ρουτίνα εξυπηρέτησης διακοπής. Στην περίπτωση που υπάρχει μόνο μία διακοπή που μπορεί να συμβεί, αυτή θα πρέπει να εκτελεί κατά σειρά τις εξής λειτουργίες:

1. Σώζει το περιεχόμενο (context save) των καταχωρητών.
2. Αναγνωρίζει (acknowledge) τη διακοπή.
3. Εξυπηρετεί (service) τη διακοπή.
4. Επαναφέρει το περιεχόμενο (context restore) των καταχωρητών και επιστρέφει στο πρόγραμμα που διακόπηκε ενεργοποιώντας τις διακοπές.

Λόγω του γεγονότος ότι ο MicroBlaze έχει μόνο μία είσοδο διακοπής, στην (σίγουρα πιο συνηθισμένη) περίπτωση ύπαρξης περισσότερων διακοπών που μπορούν να συμβούν κατά τη διάρκεια εκτέλεσης του προγράμματος, θα χρειαστεί να χρησιμοποιηθεί ο OPB Interrupt Controller. Τότε θα πρέπει η ρουτίνα εξυπηρέτησης διακοπής να τροποποιηθεί και να λειτουργεί ως εξής:

1. Σώζει το περιεχόμενο των καταχωρητών.
2. Για όλες τις διακοπές που εκκρεμούν και με σειρά προτεραιότητας:

    1. Αναγνωρίζει τη διακοπή.
    2. Εξυπηρετεί τη διακοπή.

3. Επαναφέρει το περιεχόμενο των καταχωρητών και επιστρέφει στο πρόγραμμα που διακόπηκε ενεργοποιώντας τις διακοπές.

Αυτός είναι ο πιο φυσικός τρόπος αντιμετώπισης των πολλαπλών διακοπών και αυτός θα χρησιμοποιηθεί στη συνέχεια. Μία άλλη λύση θα ήταν η εξυπηρέτηση μίας μόνο διακοπής στη ρουτίνα εξυπηρέτησης διακοπής (προφανώς της διακοπής που έχει τη μεγαλύτερη προτεραιότητα από όλες τις διακοπές που έχουν συμβεί). Οι υπόλοιπες διακοπές θα προκαλέσουν με τη σειρά τους νέες διακοπές, αφού επιστρέψει η τρέχουσα ρουτίνα εξυπηρέτησης. Μία ακόμα λύση θα ήταν η ύπαρξη φωλιασμένων κλήσεων ρουτινών εξυπηρέτησης διακοπής εξαιτίας διακοπών μεγαλύτερης προτεραιότητας από την προτεραιότητα της τρέχουσας εξυπηρετούμενης διακοπής. Στην τελευταία λύση, ωστόσο, θα ήταν αρκετά πιο περίπλοκος ο κώδικας ενώ και στις δύο τελευταίες εναλλακτικές θα ήταν πολύ μικρότερη η απόδοσή του, αφού τα πολύ χρονοβόρα τμήματα του σωσίματος και της επαναφοράς του περιεχομένου των καταχωρητών θα επαναλαμβάνονταν περισσότερες φορές. Συνεπώς, η παραπάνω λύση κρίνεται αρκετά απλή και αποδοτική και γι' αυτό θα χρησιμοποιηθεί στην τρέχουσα εφαρμογή.

Η ρουτίνα εξυπηρέτησης διακοπής περιλαμβάνει, λοιπόν, ένα βρόχο που επαναλαμβάνεται όσο

υπάρχουν διακοπές που εκκρεμούν. Οι διακοπές που ενδέχεται να εκκρεμούν είναι η διακοπή χρονιστή και η διακοπή της σειριακής θύρας, αν η σειριακή θύρα έχει αρχικοποιηθεί ώστε να δουλεύει με τον τρόπο λειτουργίας διακοπών. Στην αρχή του βρόχου διαβάζονται οι καταχωρητές Interrupt Status Register και Interrupt Enable Register του OPB Interrupt Controller και από αυτούς ελέγχεται αν υπάρχει διακοπή που εκκρεμεί (που είναι active και enabled ταυτόχρονα). Δε χρησιμοποιείται ο καταχωρητής Interrupt Pending Register, καθώς είναι προαιρετικός στις υλοποιήσεις του ελεγκτή διακοπών και ενδέχεται να μην υπάρχει στην υλοποίηση του συστήματος. Αν δεν υπάρχει διακοπή που εκκρεμεί συμβαίνει έξοδος από το βρόχο και στη συνέχεια από τη ρουτίνα εξυπηρέτησης διακοπής. Διαφορετικά ελέγχεται αν εκκρεμεί διακοπή χρονιστή. Αν εκκρεμεί διακοπή χρονιστή, τότε αρχικά αναγνωρίζεται η διακοπή και στη συνέχεια μειώνεται κατά 1 η μεταβλητή *preempt*. Αν η μεταβλητή μηδενιστεί, τότε καλείται η συνάρτηση *reschedule*, καθώς αυτό σημαίνει ότι ήρθε η ώρα η παλιά διεργασία να διακόψει την εκτέλεσή της και μία νέα να την συνεχίσει.

Η άλλη διακοπή που ενδέχεται να εκκρεμεί είναι η διακοπή της σειριακής θύρας. Όπως αναφέρθηκε και προηγουμένως, αυτό ισχύει μόνο αν η σειριακή θύρα δουλεύει με τον τρόπο λειτουργίας διακοπής. Αν βρεθεί ότι εκκρεμεί η συγκεκριμένη διακοπή, τότε αρχικά, όπως και κάθε διακοπή, αναγνωρίζεται. Στη συνέχεια διαβάζεται ο καταχωρητής κατάστασης της σειριακής θύρας και τυχόν σφάλματα που συνέβησαν κατά τη λήψη χαρακτήρων, τα οποία υποδεικνύει ο καταχωρητής αυτός, ενημερώνουν τη μεταβλητή *uartError*. Διακοπή η σειριακή θύρα μπορεί να προκαλέσει επειδή άδειασε η FIFO αποστολής ή επειδή ήρθε κάποιος χαρακτήρας στην έως τότε άδεια FIFO λήψης. Ένα πρόβλημα που υπάρχει είναι ότι η διακοπή, που είναι ευαίσθητη σε θετική ακμή, είναι μία και για τις δύο περιπτώσεις. Αν βρεθεί ότι η FIFO λήψης έχει χαρακτήρες και ταυτόχρονα η FIFO αποστολής είναι άδεια, τότε δεν είναι άμεσα γνωστό αν άδειασε η FIFO αποστολής ή ήταν ήδη άδεια. Επειδή αυτό είναι σημαντικό για τους οδηγούς της σειριακής θύρας που αφορούν στον τρόπο λειτουργίας διακοπής, χρησιμοποιείται η μεταβλητή *uartWritten*, η οποία γίνεται 1 κατά την εγγραφή στη σειριακή θύρα και γίνεται 0 μέσα στη ρουτίνα εξυπηρέτησης διακοπής, αν βρεθεί να έχει την τιμή 1 και η FIFO αποστολής βρεθεί άδεια. Το συγκεκριμένο σενάριο συνεπάγεται (με δεδομένο ότι η μεταβλητή αυτή αρχικοποιείται στο 0) ότι η FIFO αποστολής πράγματι άδειασε και δεν ήταν άδεια, αφού γράφτηκε κάποιος χαρακτήρας σε αυτή. Σε αυτή την περίπτωση καλείται η συνάρτηση *V* πάνω στο σηματοφορέα εξόδου της σειριακής θύρας, που σημαίνει ότι ένας νέος χαρακτήρας μπορεί να γραφτεί πλέον στη σειριακή θύρα. Στη συνέχεια διαβάζονται χαρακτήρες από τη FIFO λήψης μέχρι αυτή να αδειάσει. Αν διαβαστεί ένας νέος χαρακτήρας και ο buffer εισόδου της σειριακής θύρας είναι γεμάτος, τότε ο χαρακτήρας θα απορριφθεί και θα ενημερωθεί η μεταβλητή *uartError* για το σφάλμα αυτό. Σε

διαφορετική περίπτωση, ο χαρακτήρας θα γραφτεί στο buffer και θα κληθεί η συνάρτηση V πάνω στο σηματοφορέα εισόδου της σειριακής θύρας, που σημαίνει ότι ένας νέος χαρακτήρας μπορεί να διαβαστεί από τον buffer εισόδου της σειριακής θύρας.

Κεφάλαιο 5

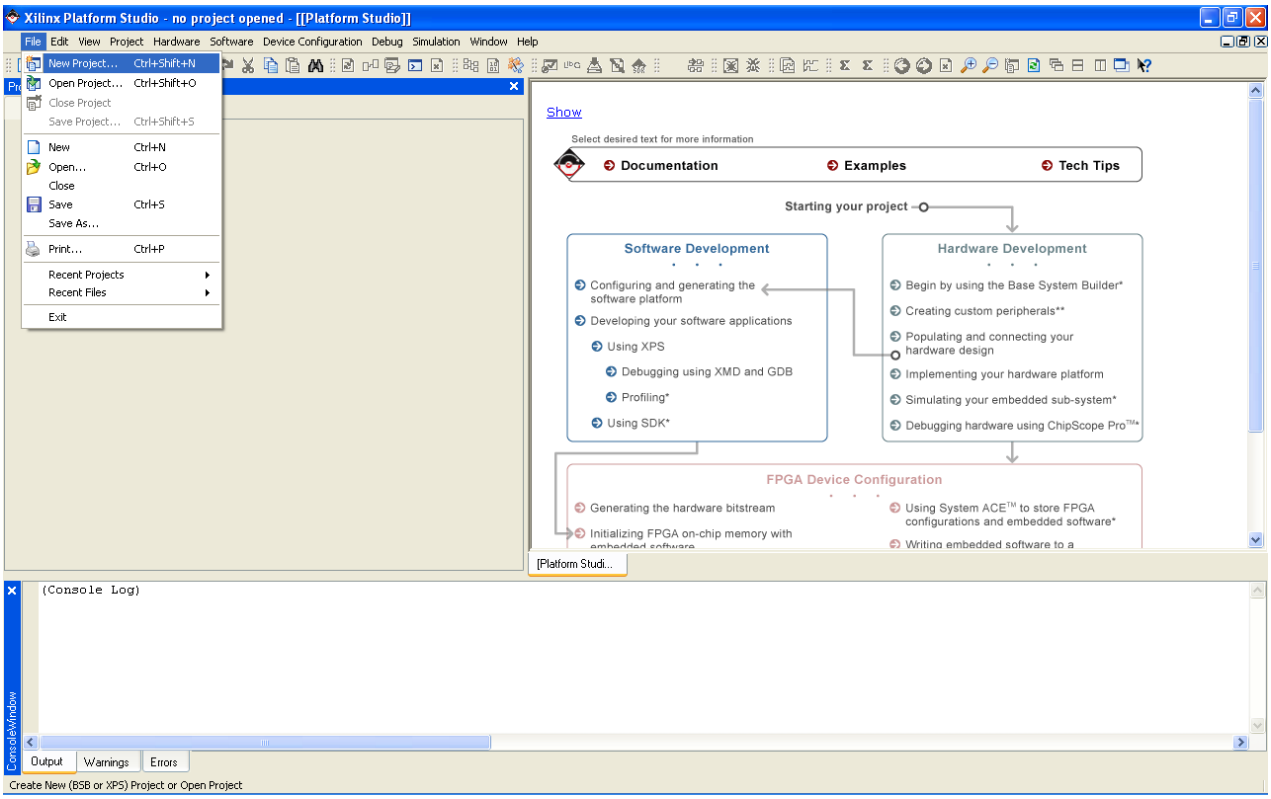
Παραδείγματα Εκτέλεσης

Σε αυτό το κεφάλαιο θα παρουσιαστούν αναλυτικά ποικίλες περιπτώσεις χρήσης (use cases) του Xilinx Platform Studio (XPS), από τη δημιουργία ενός νέου project μέχρι την ανάπτυξη εφαρμογών για το MicroEmpix/FPGA. Οι εφαρμογές θα τρέξουν τόσο σε εικονική πλατφόρμα (virtual platform) όσο και στο Spartan-3E Starter Board. Θα καλυφθούν, δηλαδή, περιπτώσεις χρήσης από όλη τη διαδικασία ανάπτυξης ενός ενσωματωμένου συστήματος. Η έκδοση του XPS που χρησιμοποιήθηκε είναι η 8.1i και η ανάπτυξη έγινε σε Windows XP. Πέρα από το Xilinx Platform Studio και τα εργαλεία που υποστηρίζει θα χρησιμοποιηθεί το πρόγραμμα HyperTerminal των Windows XP για την επικοινωνία ενός υπολογιστή με το Spartan-3E μέσω της σειριακής θύρας και τη δοκιμή του ενσωματωμένου συστήματος στο τελευταίο στάδιο της ανάπτυξης. Για την καλύτερη κατανόηση των παραδειγμάτων, τη λεκτική περιγραφή θα συνοδεύουν και τα κατάλληλα screenshots, όπου αυτό κρίνεται απαραίτητο.

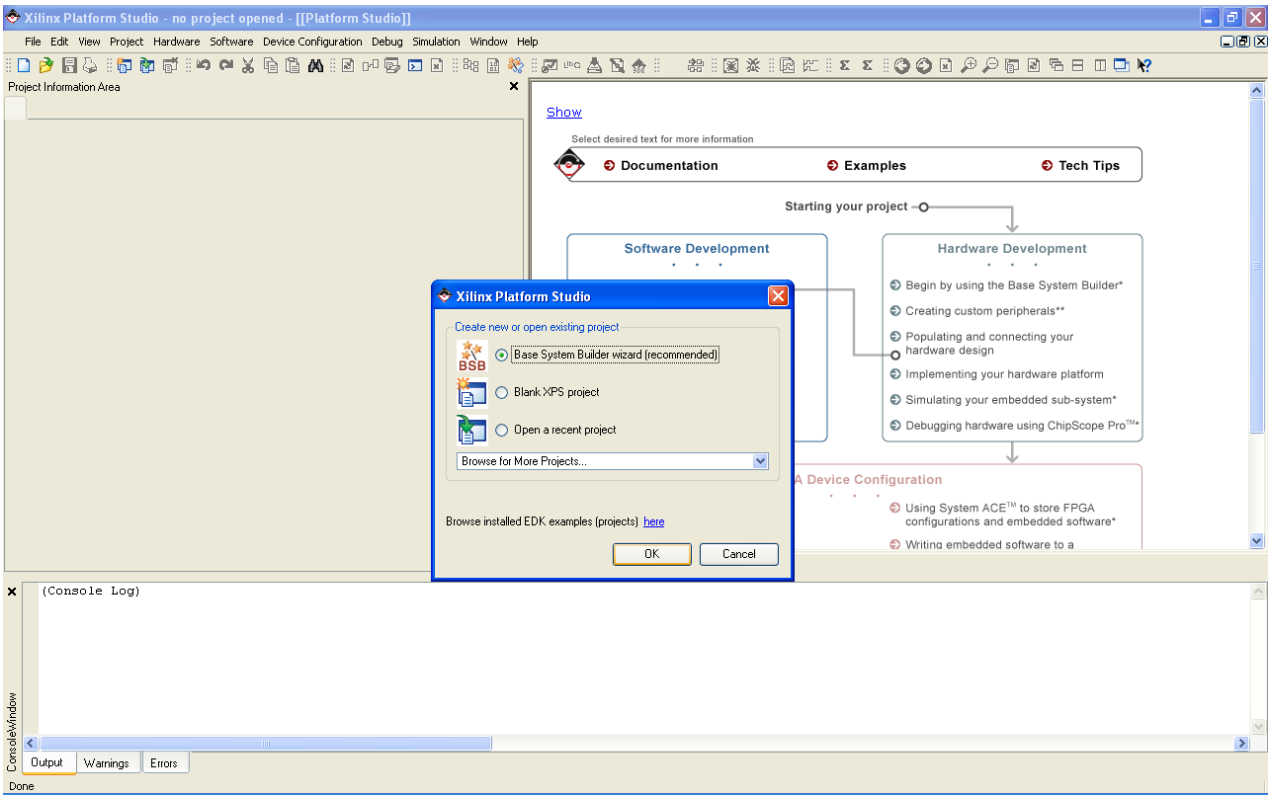
5.1 Δημιουργία Νέου XPS Project

5.1.1 Δημιουργία Νέου Project File

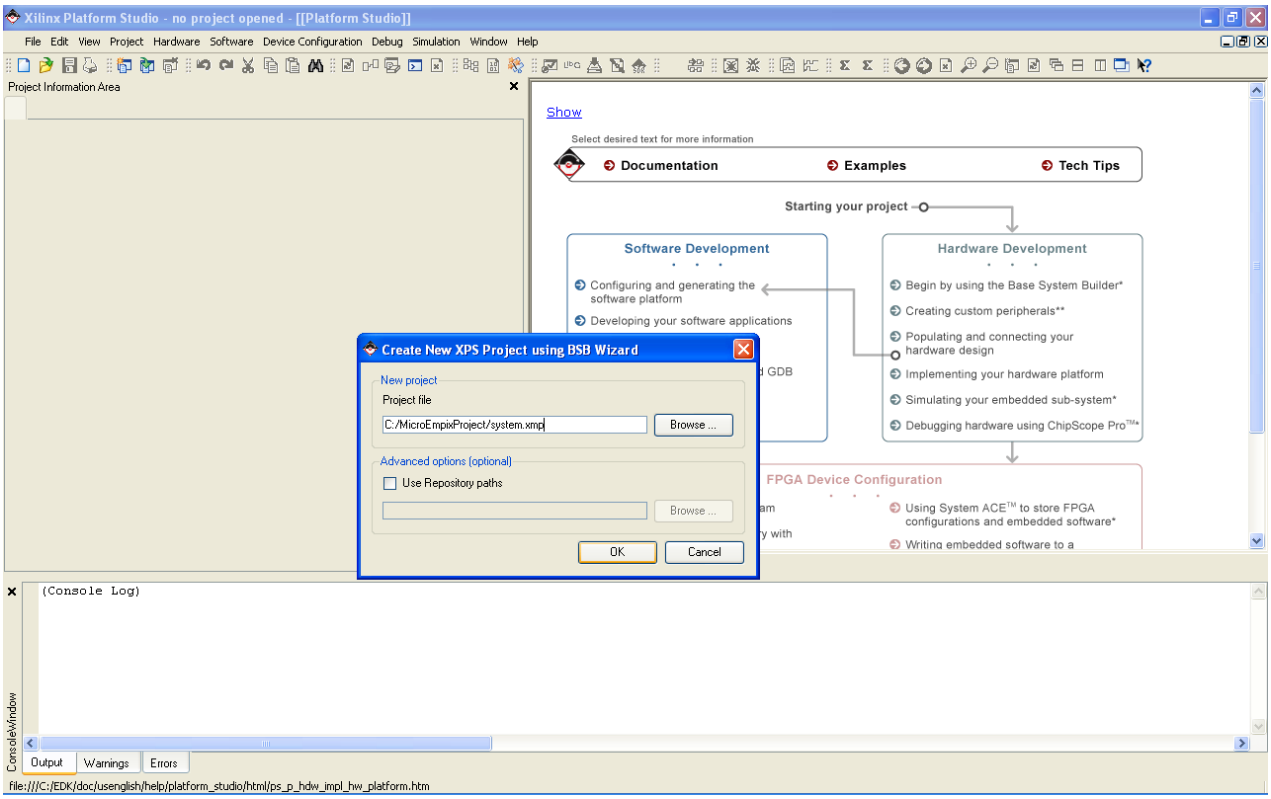
Ο χρήστης ανοίγει το Xilinx Platform Studio και επιλέγει από το μενού “File” το “New Project”. Από το παράθυρο που θα έρθει στο προσκήνιο επιλέγει “Base System Builder Wizard” και πατάει το πλήκτρο “OK”. Στη συνέχεια ζητείται ένα project file. Ο χρήστης πατάει το πλήκτρο “Browse...” και κατευθύνεται εν συνεχεία μέσα στο φάκελο όπου θέλει να δημιουργηθεί το νέο project file. Ένα αρχείο XPS project έχει την επέκταση .xmp και θα δημιουργηθεί μέσα στο φάκελο στον οποίο θα κατευθυνθεί ο χρήστης. Το πλήρες μονοπάτι του αρχείου στη συγκεκριμένη περίπτωση θα είναι C:/MicroEmpixProject/system.xmp. Αφού το κάνει αυτό, πατάει το πλήκτρο “Save” και στη συνέχεια “OK”. Τα βήματα αυτά φαίνονται από το Σχήμα 5.1 έως και το Σχήμα 5.3.



Σχήμα 5.1: New Project



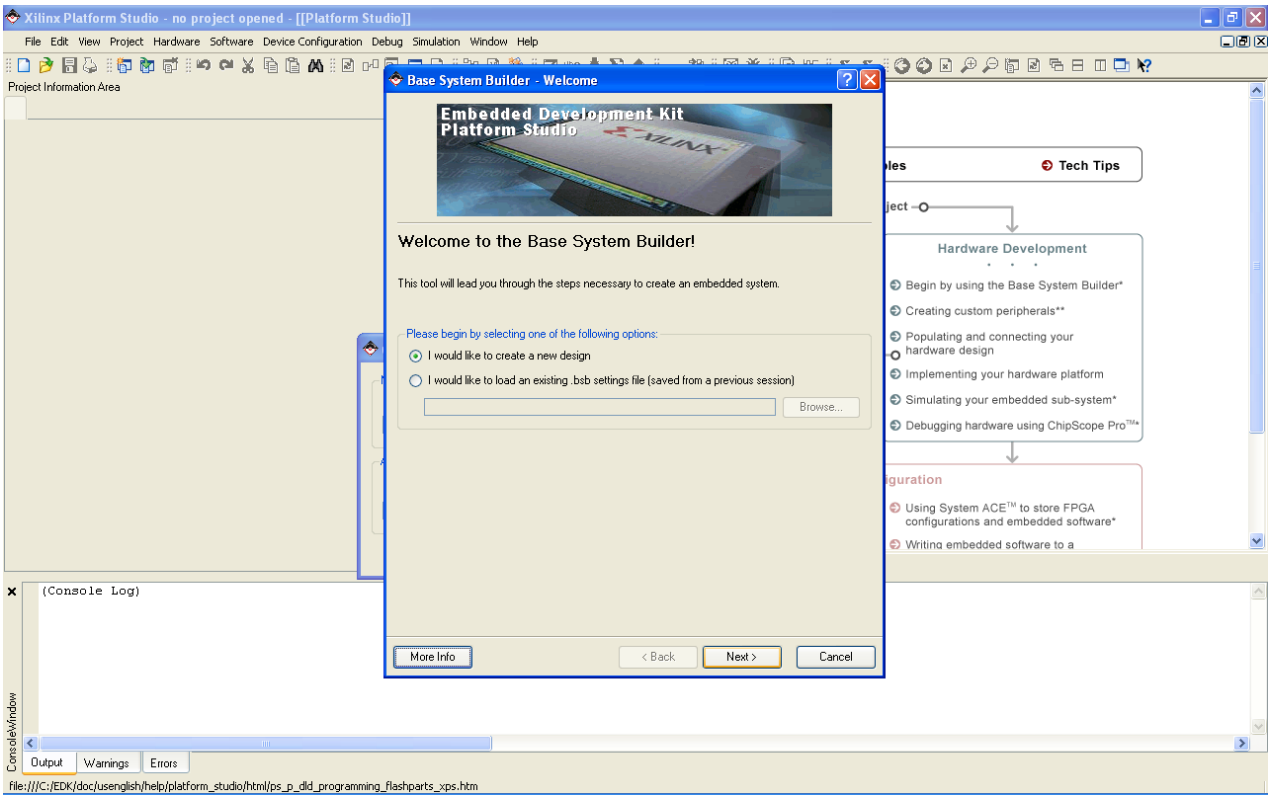
Σχήμα 5.2: Base System Builder Wizard



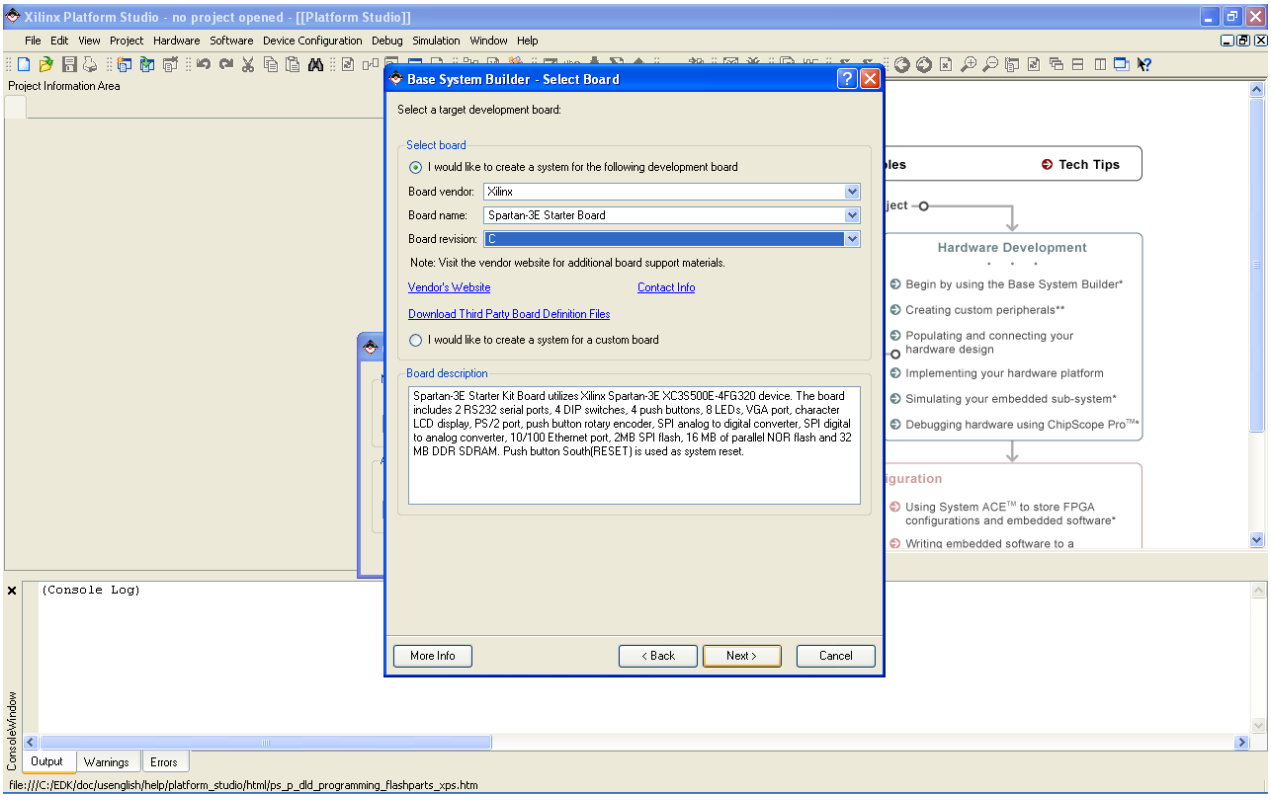
Σχήμα 5.3: Project file

5.1.2 Επιλογή Πλακέτας Στόχου

Αφού ανοίξει ο BSB Wizard, ο χρήστης επιλέγει “I would like to create a new design” και πατάει το πλήκτρο “Next >”. Μετά επιλέγει “I would like to create a system for the following development board” και κατόπιν επιλέγει κατά σειρά Board vendor: “Xilinx”, Board name: “Spartan-3E Starter Board” και Board revision: “C”. Στο πεδίο Board description ο χρήστης βλέπει τα χαρακτηριστικά της πλακέτας που επέλεξε. Πατάει το πλήκτρο “Next >”. Τα παραπάνω βήματα φαίνονται στο Σχήμα 5.4 και στο Σχήμα 5.5.



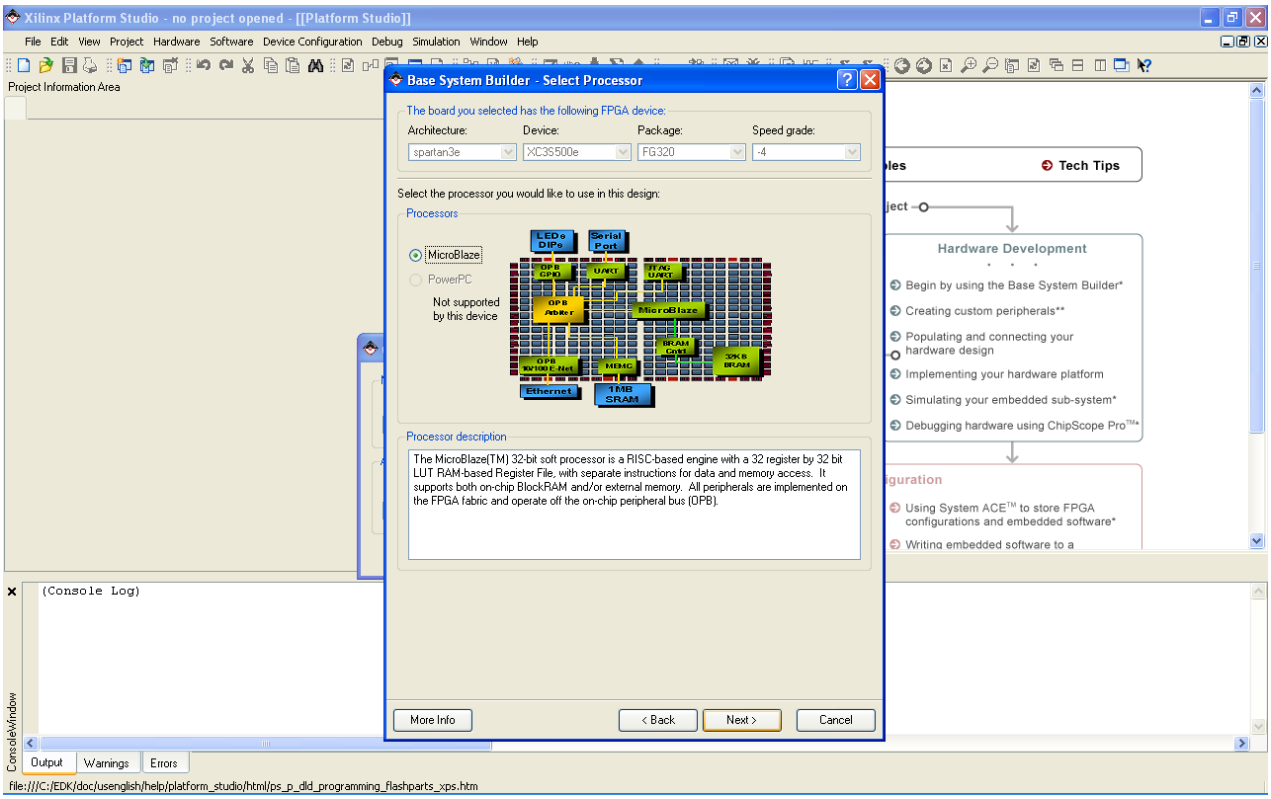
Σχήμα 5.4: Welcome to the Base System Builder



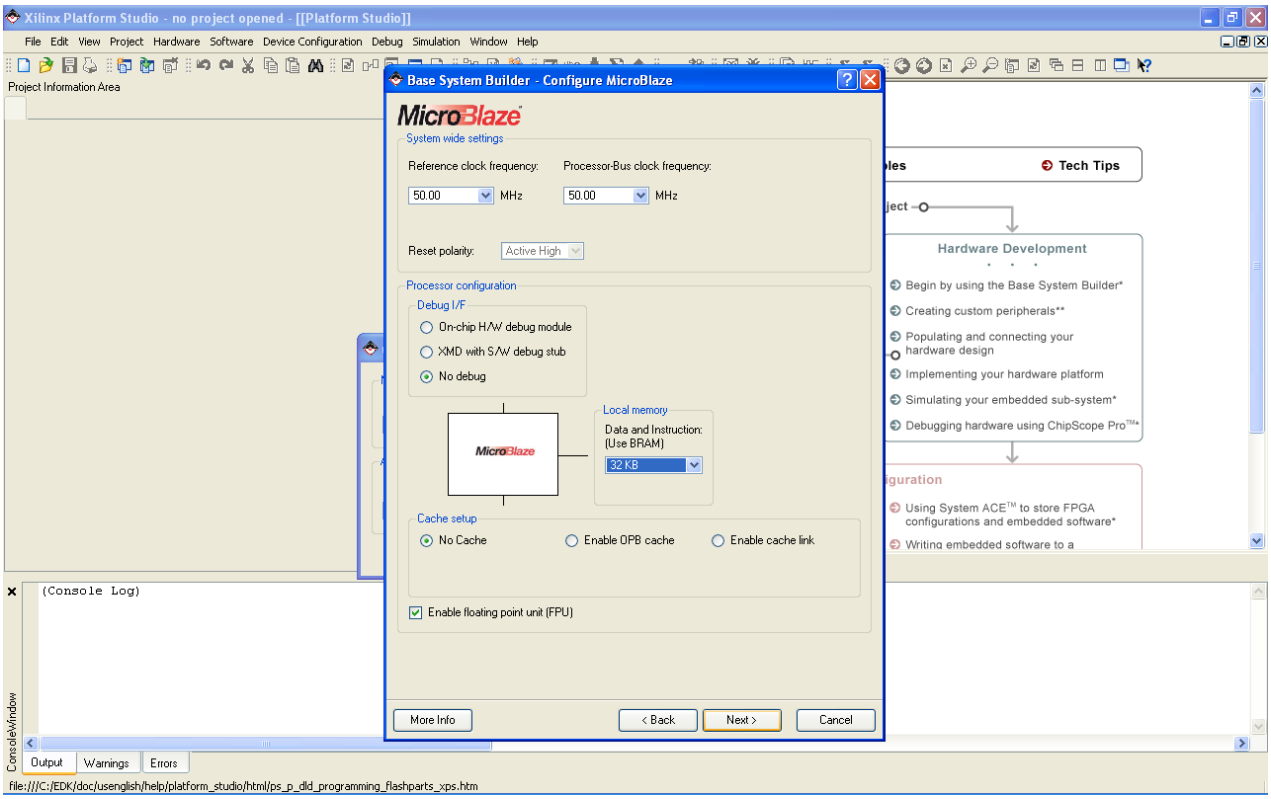
Σχήμα 5.5: Select Board

5.1.3 Επιλογή και Ρύθμιση Επεξεργαστή

Στο παράθυρο που ανοίγει ο χρήστης επιλέγει τον επεξεργαστή που θέλει να χρησιμοποιήσει. Μόνο ο "MicroBlaze" είναι διαθέσιμος στη συγκεκριμένη περίπτωση και αυτόν επιλέγει. Στο πεδίο Processor description ο χρήστης βλέπει τα χαρακτηριστικά του εν λόγω επεξεργαστή. Πατάει το πλήκτρο "Next >" για συνέχεια. Στο νέο παράθυρο επιλέγει "No debug" στο πεδίο Debug I/F και "32 KB" στο πεδίο Local memory, το οποίο αναφέρεται στην Block RAM που θα διαθέτει το σύστημα. Επίσης, μαρκάρει το "Enable floating point unit (FPU)", καθώς επιθυμεί ο MicroBlaze να διαθέτει αυτό το προαιρετικό χαρακτηριστικό για την επιτάχυνση των πράξεων κινητής υποδιαστολής. Στο πεδίο Cache setup επιλέγει "No Cache", καθώς δε θα χρησιμοποιήσει εξωτερική μνήμη στο εν λόγω σύστημα. Αφήνει τα πεδία Reference clock frequency και Processor-Bus clock frequency ως έχουν. Πατάει και πάλι το πλήκτρο "Next >" για μετάβαση στο επόμενο παράθυρο. Τα παραπάνω βήματα φαίνονται στο Σχήμα 5.6 και στο Σχήμα 5.7.



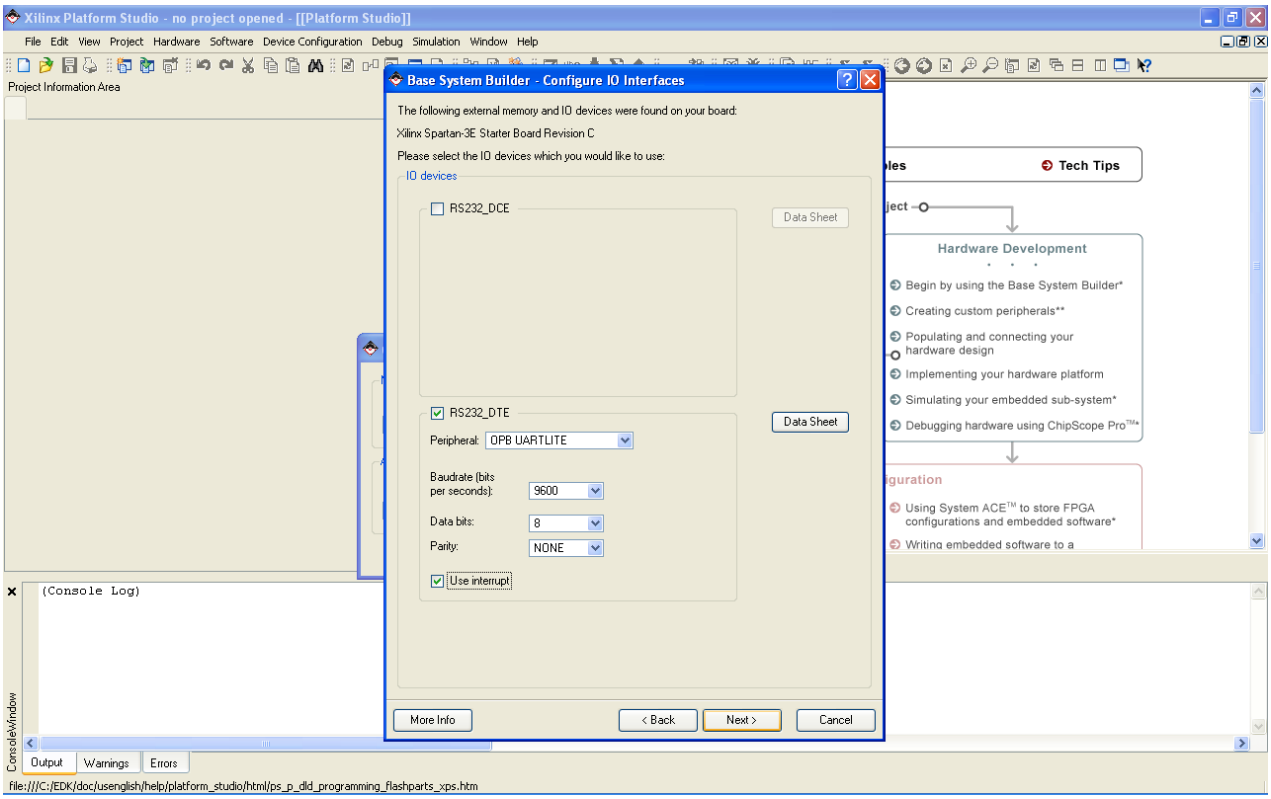
Σχήμα 5.6: Select Processor



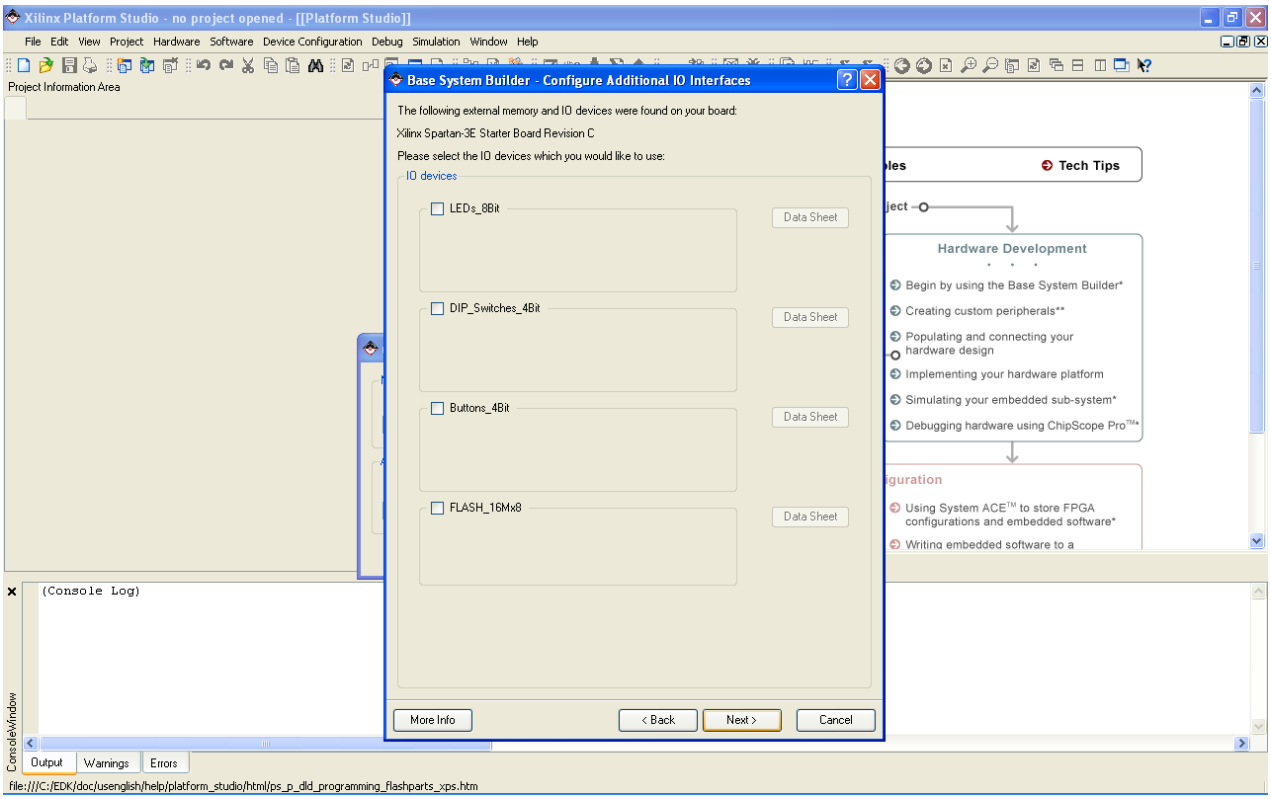
Σχήμα 5.7: Configure MicroBlaze

5.1.4 Ρύθμιση Συσκευών

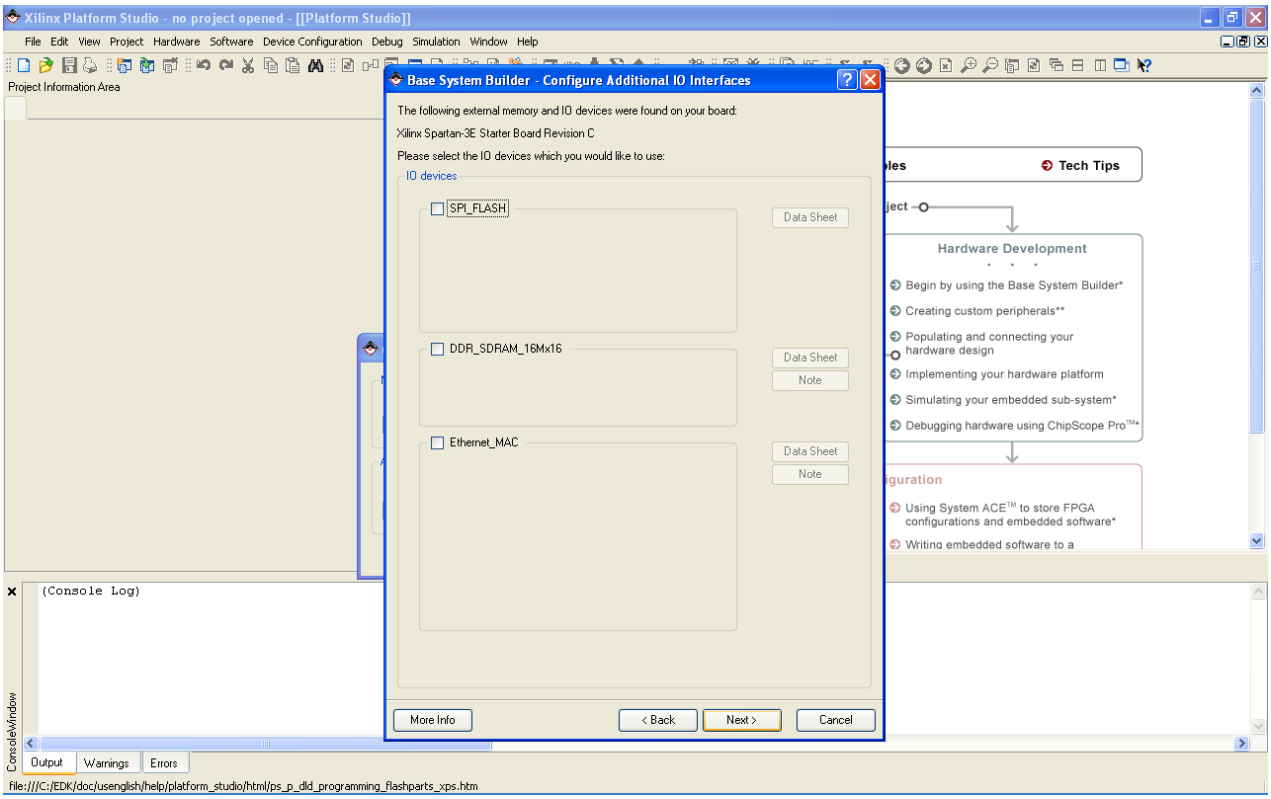
Από εδώ και πέρα ο χρήστης επιλέγει τα περιφερειακά που θα συμπληρώσουν το σύστημά του. Στα πρώτα παράθυρα επιλέγει τις συσκευές E/E. Από το τρέχον παράθυρο μαρκάρει μόνο το "RS232_DTE", που αποτελεί τη θηλυκή σειριακή θύρα του Spartan-3E Starter Board, και την επιλογή του "Use interrupt". Στις άλλες επιλογές της σειριακής θύρας αφήνει τις προκαθορισμένες τιμές, δηλαδή Peripheral: "OPB UARTLITE", Baudrate: "9600", Data bits: "8" και Parity: "NONE". Πατάει το πλήκτρο "Next >" για συνέχεια. Από το παράθυρο στο οποίο βρίσκεται τώρα ξεμαρκάρει όλα τα περιφερειακά E/E και πατάει το πλήκτρο "Next >" για συνέχεια. Ακριβώς τα ίδια κάνει και στο τρέχον παράθυρο και πηγαίνει στο επόμενο παράθυρο. Στο παράθυρο όπου βρίσκεται τώρα (Add Internal Peripherals) ο χρήστης πατάει το πλήκτρο "Add Peripheral..." και στο παράθυρο που εμφανίζεται επιλέγει "OPB TIMER" και κατόπιν πατάει το πλήκτρο "OK". Τώρα επιλέγει το "Use interrupt" αφήνοντας τις προκαθορισμένες τιμές Counter bit width: "32" και "Two timers are present" στο πεδίο Timer mode. Πατάει το πλήκτρο "Next >" για συνέχεια. Τα παραπάνω φαίνονται από το Σχήμα 5.8 έως και το Σχήμα 5.12.



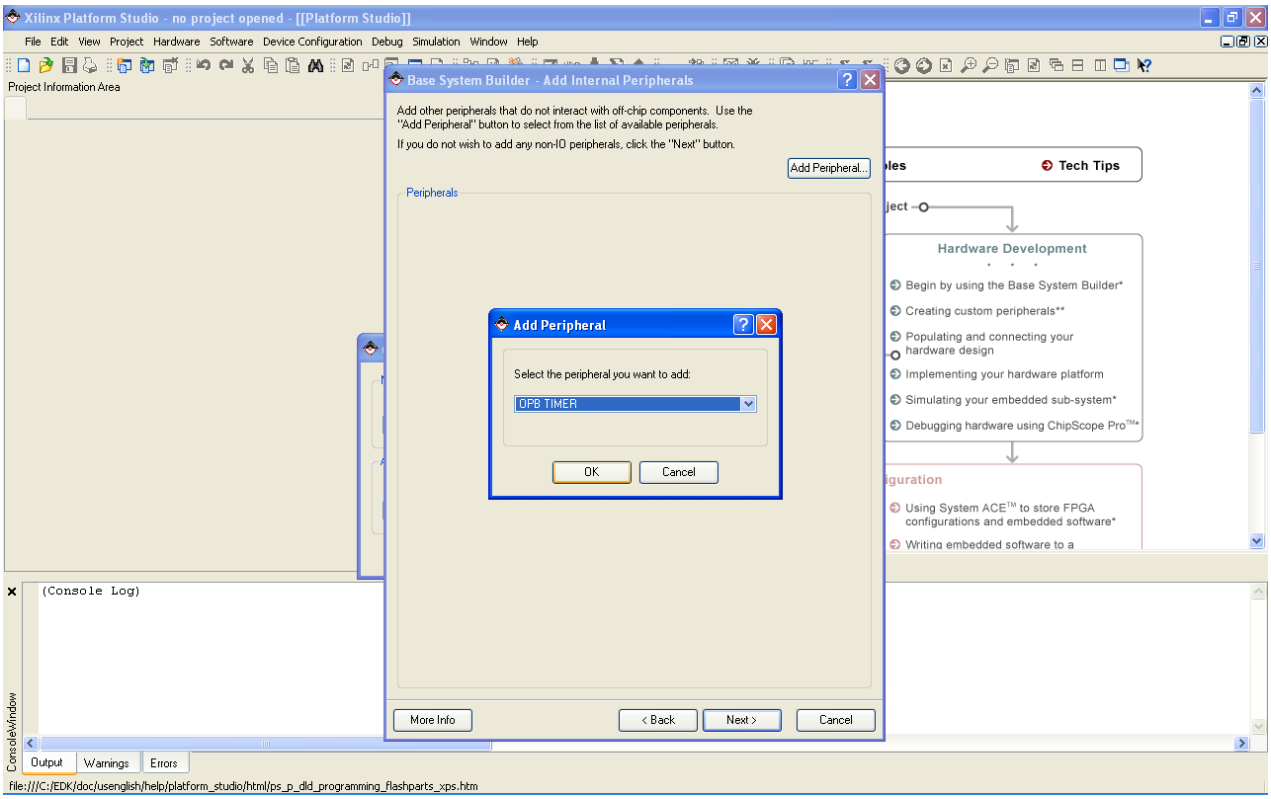
Σχήμα 5.8: Configure IO Interfaces



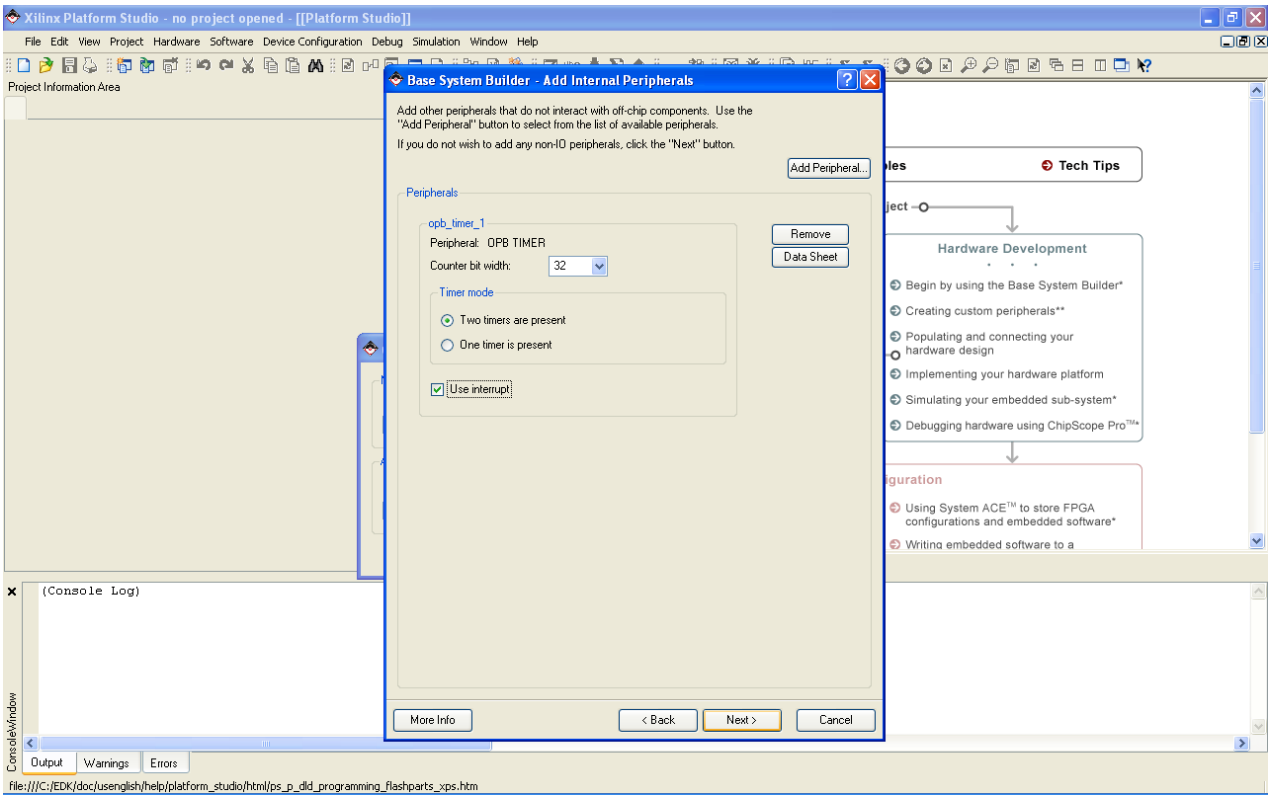
Σχήμα 5.9: Configure Additional IO Interfaces



Σχήμα 5.10: Configure Additional IO Interfaces (continued)



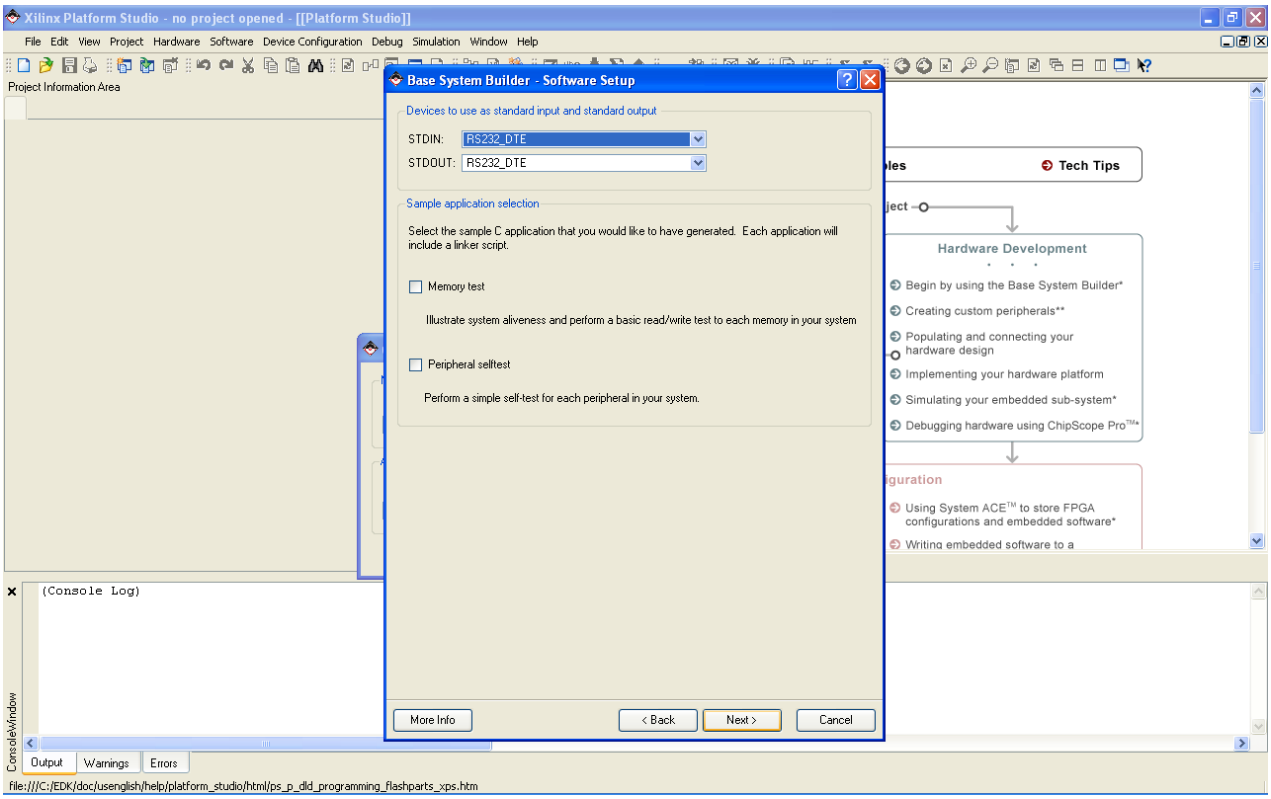
Σχήμα 5.11: Add Internal Peripherals



Σχήμα 5.12: Add Internal Peripherals (continued)

5.1.5 Ρύθμιση Λογισμικού

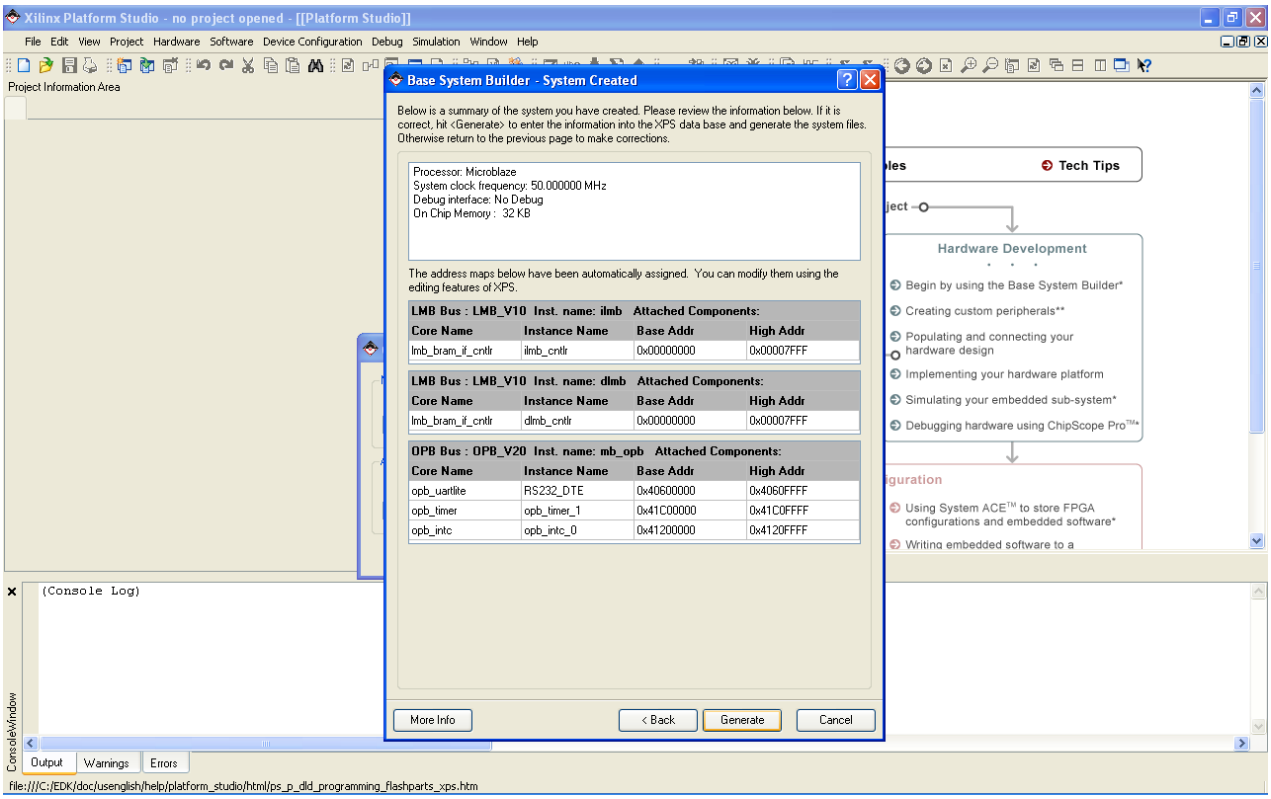
Στο πεδίο Sample application selection ο χρήστης ξεμαρκάρει και τις δύο επιλογές που βλέπει, καθώς θα δημιουργήσει τις δικές του εφαρμογές παρακάτω. Πατάει το πλήκτρο “Next >” για συνέχεια. Το παραπάνω φαίνεται στο Σχήμα 5.13.



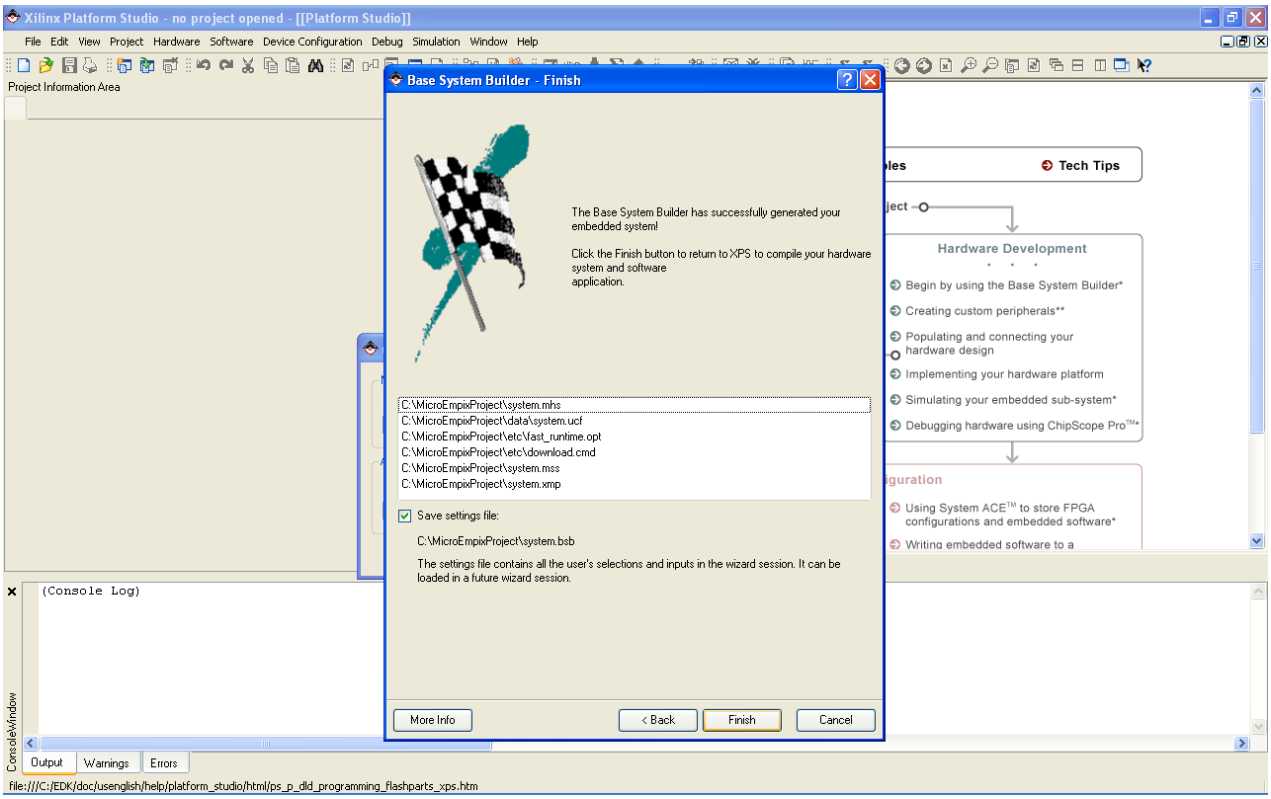
Σχήμα 5.13: Software Setup

5.1.6 Δημιουργία Συστήματος

Στο τρέχον παράθυρο ο χρήστης βλέπει το σύστημα που θα δημιουργήσει. Πιο συγκεκριμένα, βλέπει ορισμένα χαρακτηριστικά του επεξεργαστή που θα χρησιμοποιήσει, καθώς και τις συνδέσεις των περιφερειακών στο On-chip Peripheral Bus και των δύο ελεγκτών της Block RAM στο Local Memory Bus για τα δεδομένα και στο Local Memory Bus για τις εντολές. Για συνέχεια πατάει το πλήκτρο “Generate”. Πατώντας, τελικά, το πλήκτρο “Finish” ολοκληρώνει τη δημιουργία του συστήματός του και μπορεί πλέον να δουλέψει πάνω σε αυτό με το Xilinx Platform Studio. Τα παραπάνω φαίνονται στο Σχήμα 5.14 και στο Σχήμα 5.15.



Σχήμα 5.14: System Created

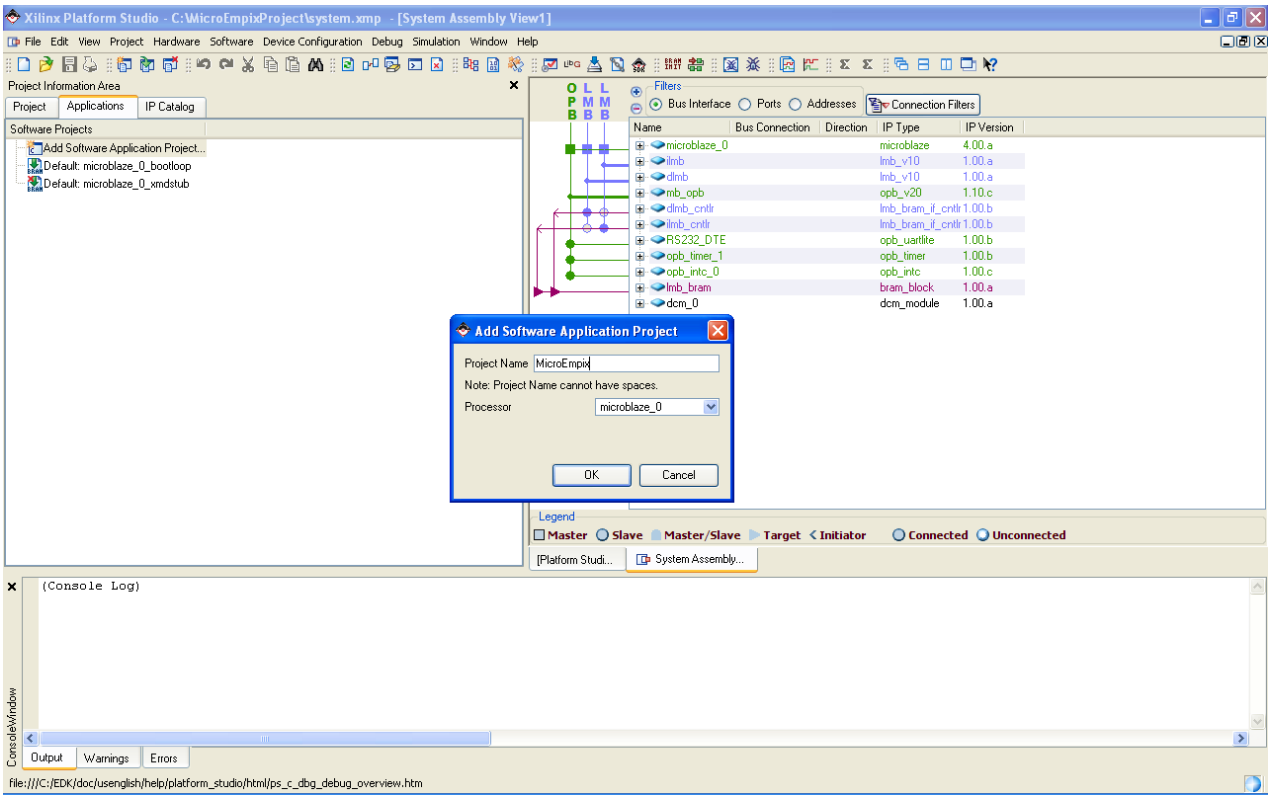


Σχήμα 5.15: Finish

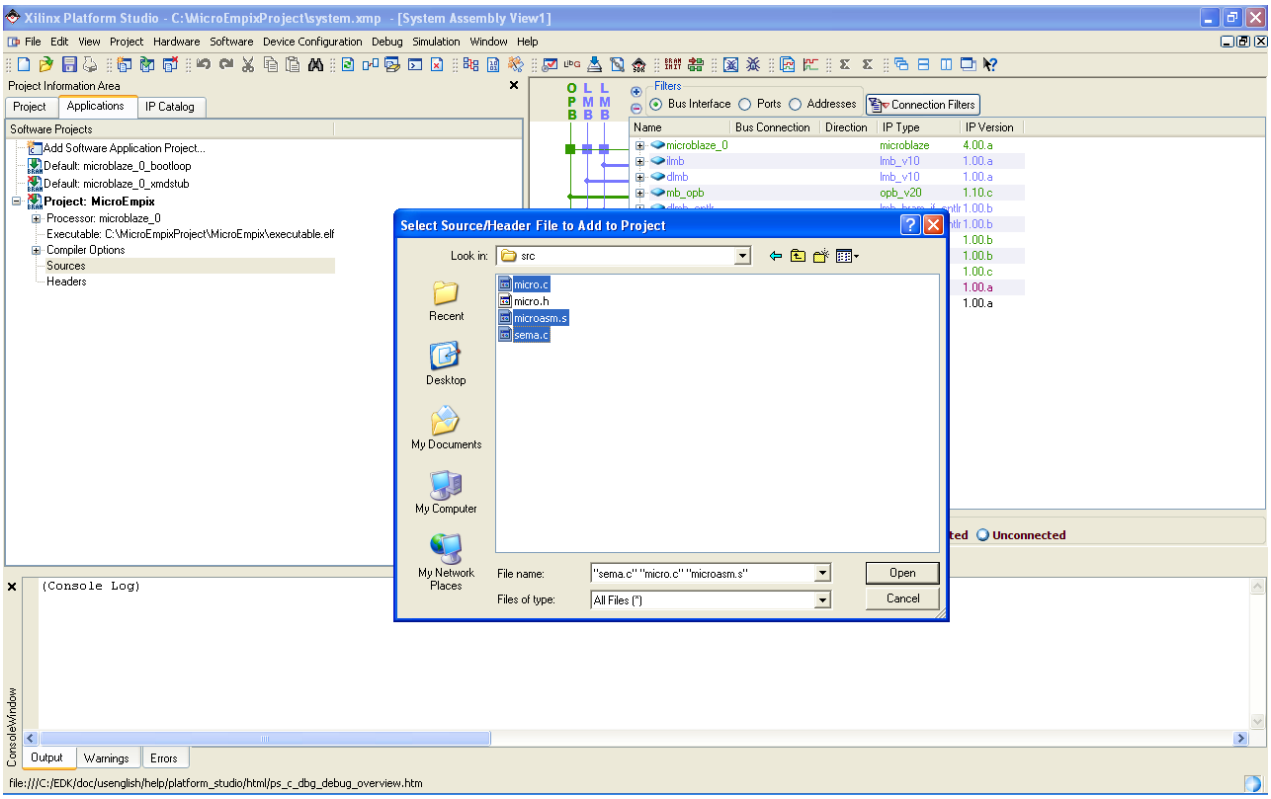
5.2 Δημιουργία Νέας Εφαρμογής Λογισμικού

5.2.1 Αρχεία Εφαρμογής

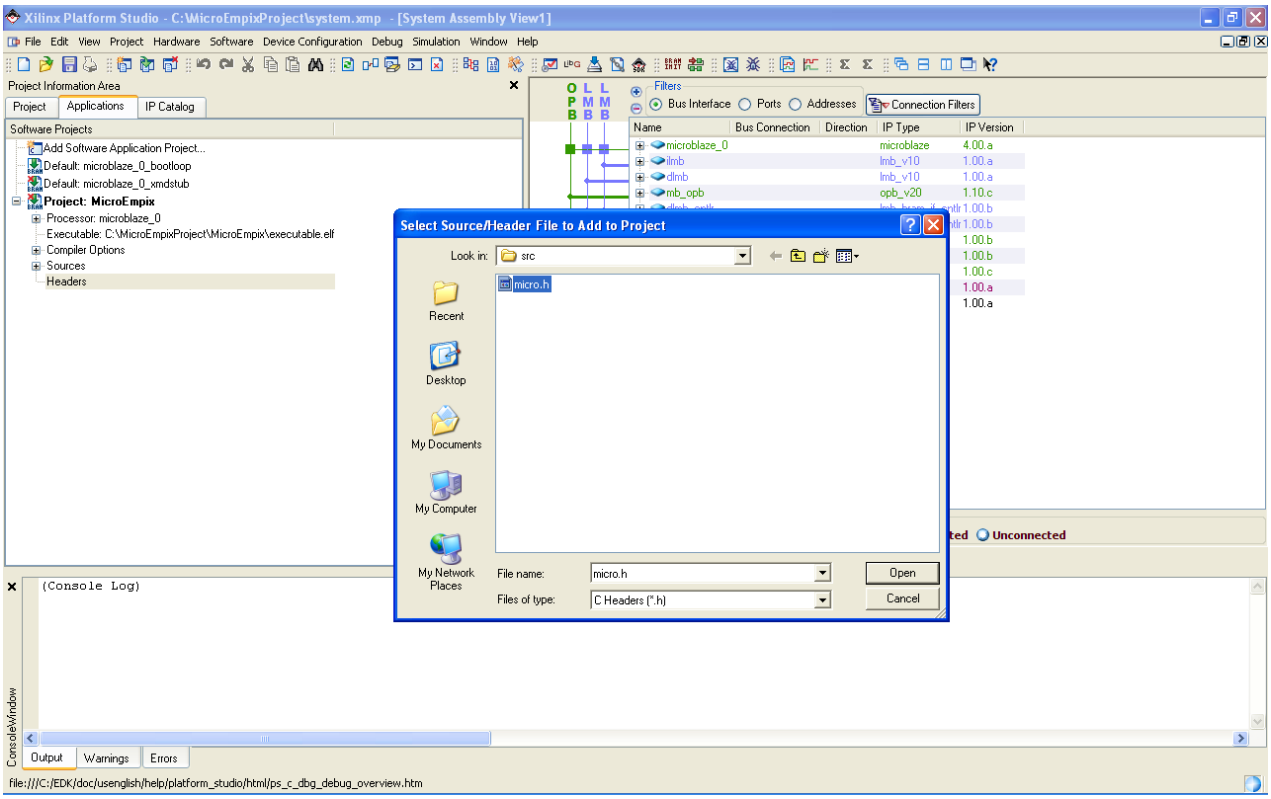
Για να δημιουργήσει μία νέα εφαρμογή λογισμικού, εν προκειμένω το MicroEmpix/FPGA, ο χρήστης κάνει κλικ αρχικά στο tab “Applications” από το πεδίο Project Information Area και κατόπιν κάνει διπλό κλικ στο “Add Software Application Project...”. Στο παράθυρο που εμφανίζεται θέτει στο πεδίο Project Name το όνομα που θέλει να δώσει στην εφαρμογή του, στη συγκεκριμένη περίπτωση MicroEmpix, και στη συνέχεια πατάει το πλήκτρο “OK”. Μέσα στο φάκελο MicroEmpixProject δημιουργεί το φάκελο MicroEmpix. Μέσα στο φάκελο MicroEmpix δημιουργεί το φάκελο src. Σε αυτόν τον φάκελο τοποθετούνται τα αρχεία micro.h, micro.c και microasm.s, που αποτελούν τον κώδικα του MicroEmpix/FPGA, καθώς και το αρχείο sema.c, που αποτελεί μία εφαρμογή γραμμένη για το MicroEmpix/FPGA και θα εξηγηθεί παρακάτω. Στη συνέχεια ο χρήστης κάνει διπλό κλικ στο πεδίο Sources του MicroEmpix, εντοπίζει και επιλέγει τα αρχεία micro.c, microasm.s και sema.c, που απαρτίζουν τον πηγαίο κώδικα της εφαρμογής. Πατάει εν συνεχεία το πλήκτρο “Open” και τα αρχεία προστίθενται στα Sources. Κάνει το ίδιο με το πεδίο Headers για την προσθήκη του αρχείου micro.h, που χρησιμοποιείται ως επικεφαλίδα στο micro.c και στα αρχεία των εφαρμογών του χρήστη του MicroEmpix/FPGA, όπως στο αρχείο sema.c λόγου χάρη. Τα παραπάνω φαίνονται από το Σχήμα 5.16 έως και το Σχήμα 5.18.



Σχήμα 5.16: Add Software Application Project



Σχήμα 5.17: Select Source Files



Σχήμα 5.18: Select Header Files

Το αρχείο sema.c περιέχει μία εφαρμογή γραμμένη για το MicroEmpix/FPGA. Ο κώδικας της εφαρμογής είναι ο εξής:

```
#include "micro.h"

int n;

void P1()
{
        while (1)
        {
                n = n + 1;
                PrintStr("==P1:");
                PrintInt(n);
                if (n == 100)
```

```
                {
                        V(2);
                        P(1);
                }
        }
}

void P2()
{
        P(2);
        while (1)
        {
                n = n - 1;
                PrintStr("==P2:");
                PrintInt(n);
                if (n == 0)
                {
                        V(1);
                        P(2);
                }
        }
}

void Initialize()
{
        n = 0;
}
```

Στην εφαρμογή αυτή υπάρχουν μία ακέραιη καθολική μεταβλητή, μία συνάρτηση που την αρχικοποιεί και δύο συναρτήσεις που πρόκειται να εκτελεστούν από δύο διεργασίες του MicroEmpix/FPGA. Οι δύο διεργασίες συγχρονίζονται με τη χρήση σηματοφορέων, αλλάζουν και γράφουν στη σειριακή θύρα την τιμή της καθολικής μεταβλητής. Η καθολική μεταβλητή αρχικοποιείται στο 0. Στην αρχή τον

έλεγχο έχει η πρώτη διεργασία, η οποία συνεχώς αυξάνει κατά 1 την τιμή της μεταβλητής και την γράφει στη σειριακή θύρα. Όταν η τιμή της μεταβλητής γίνει 100, αφού την γράψει, θα δώσει τον έλεγχο στη δεύτερη διεργασία. Η δεύτερη διεργασία συνεχώς μειώνει την τιμή της μεταβλητής κατά 1 και την γράφει. Όταν η μεταβλητή πάρει την τιμή 0, αφού την γράψει, θα δώσει τον έλεγχο στην πρώτη διεργασία. Αυτό θα επαναλαμβάνεται συνεχώς.

Για την ενσωμάτωση της εφαρμογής του χρήστη στο MicroEmpix/FPGA πρέπει να γίνουν δύο τροποποιήσεις στον κώδικα του micro.c. Η πρώτη τροποποίηση γίνεται στην αρχή του αρχείου και πρόκειται για τη δήλωση των εξωτερικών συναρτήσεων του χρήστη, δηλαδή της συνάρτησης αρχικοποίησης και των δύο συναρτήσεων που θα εκτελεστούν από τις δύο διεργασίες. Η προσθήκη είναι η εξής:

```
// Declaration of external user functions
extern void Initialize();
extern void P1();
extern void P2();
// End of declaration of external user functions
```

Η δεύτερη προσθήκη γίνεται στο τέλος του αρχείου micro.c, μέσα στη συνάρτηση main, και πρόκειται για την κλήση της συνάρτησης αρχικοποίησης και τη δημιουργία των δύο διεργασιών. Η προσθήκη είναι η εξής:

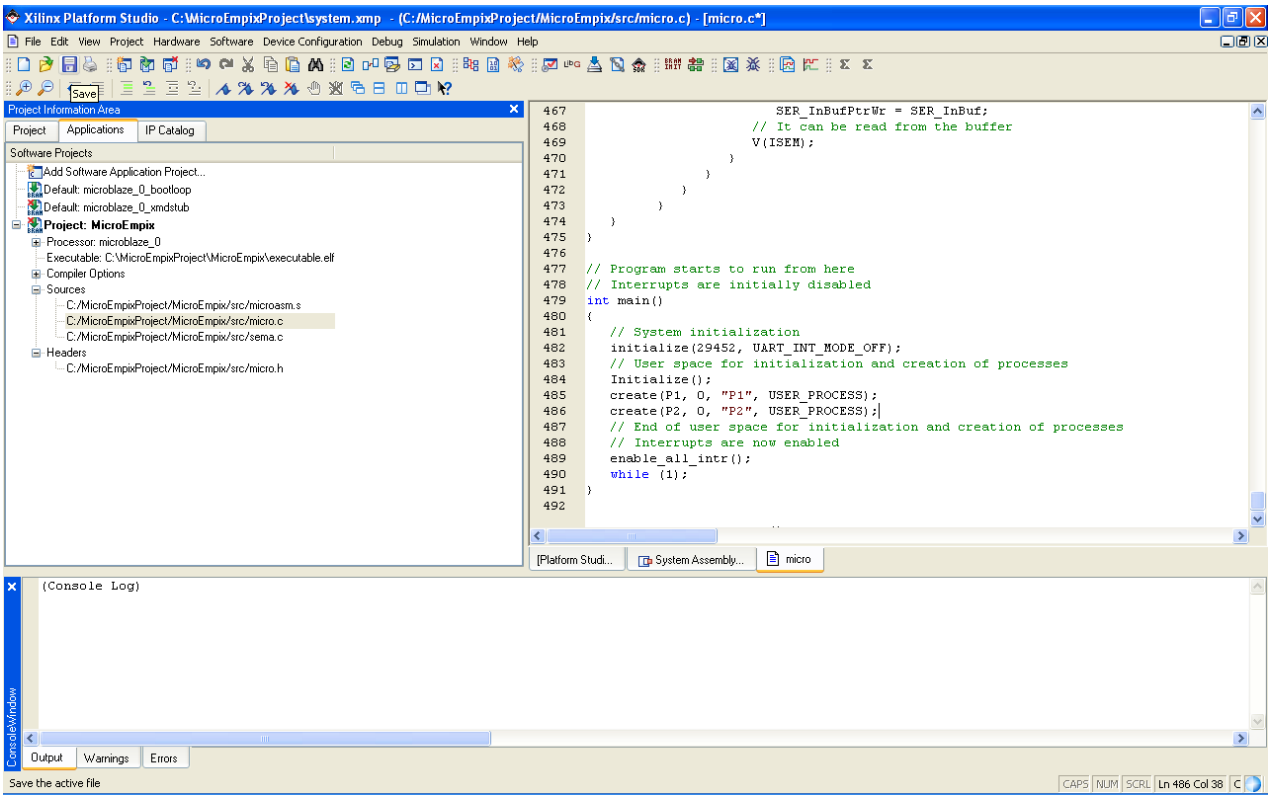
```
// User space for initialization and creation of processes
Initialize();
create(P1, 0, "P1", USER_PROCESS);
create(P2, 0, "P2", USER_PROCESS);
// End of user space for initialization and creation of processes
```

Να σημειωθεί, τέλος, ότι στην αρχή της main γίνεται η αρχικοποίηση του συστήματος με τον εξής τρόπο:

```
// System initialization
initialize(29452, UART_INT_MODE_OFF);
```

Εφόσον οι δύο διεργασίες του sema.c καλούν μόνο τις συναρτήσεις PrintInt και PrintStr όσον αφορά στις λειτουργίες της σειριακής θύρας, δε χρειάζεται ο χρήστης να αλλάξει το δεύτερο όρισμα στην παραπάνω εντολή. Αυτό συμβαίνει, γιατί οι συναρτήσεις αυτές ελέγχουν αν είναι ενεργοποιημένος ή όχι ο τρόπος λειτουργίας διακοπής της σειριακής θύρας και χρησιμοποιούν τις κατάλληλες μεθόδους. Ο χρήστης θα είχε σωστή λειτουργία και στην περίπτωση που το παραπάνω δεύτερο όρισμα ήταν UART_INT_MODE_ON. Σε περίπτωση που χρησιμοποιούσε συναρτήσεις με το πρόθεμα SER, θα έπρεπε η αρχικοποίηση να γίνει όπως παραπάνω. Σε περίπτωση που χρησιμοποιούσε συναρτήσεις με το πρόθεμα QSR, θα έπρεπε η αρχικοποίηση να γίνει υποχρεωτικά με δεύτερο όρισμα UART_INT_MODE_ON.

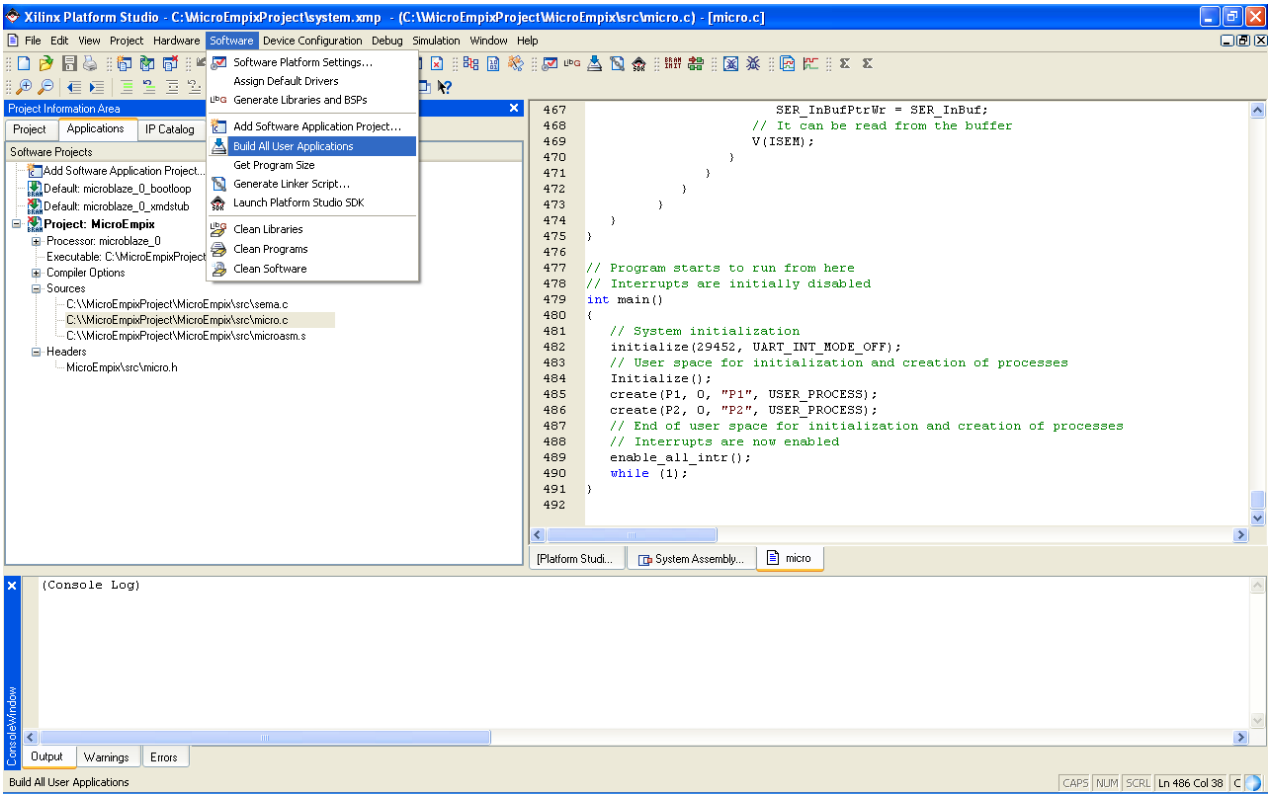
Οι τροποποιήσεις σε ένα αρχείο μίας εφαρμογής μπορούν να γίνουν είτε με τη χρήση ενός ανεξάρτητου text editor είτε μέσα από το ίδιο το XPS κάνοντας διπλό κλικ στο όνομα του αρχείου. Ανοίγει τότε το αρχείο, ο χρήστης μπορεί να το επεξεργαστεί και στο τέλος να το σώσει κάνοντας κλικ στο εικονίδιο της δισκέτας. Αυτό φαίνεται στο Σχήμα 5.19.



Σχήμα 5.19: File Edit

5.2.2 Δημιουργία Εκτελέσιμου Εφαρμογής

Σε αυτή τη φάση ο χρήστης θα δημιουργήσει το εκτελέσιμο αρχείο του προγράμματός του. Από το μενού “Software” κάνει κλικ στην επιλογή “Build All User Applications”. Αν δεν υπάρχουν σφάλματα στο πρόγραμμα, θα δημιουργηθεί ένα αρχείο executable.elf μέσα στο φάκελο MicroEmpix. Αυτό είναι το εκτελέσιμο αρχείο του προγράμματος. Αν υπάρχουν σφάλματα, τότε αυτά θα εμφανιστούν στο κάτω μέρος της οθόνης και θα πρέπει να διορθωθούν, ώστε στη συνέχεια να δημιουργηθεί το εκτελέσιμο. Αυτό φαίνεται στο Σχήμα 5.20.

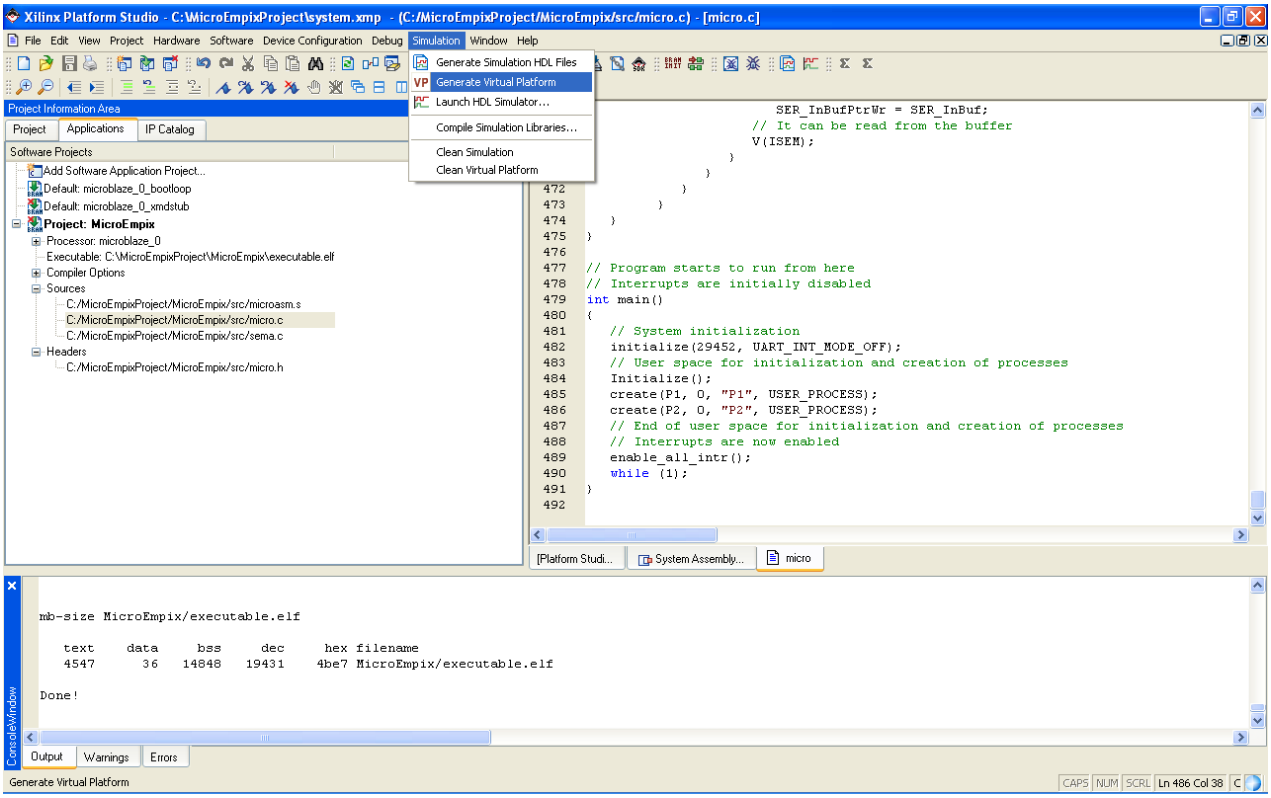


Σχήμα 5.20: Build All User Applications

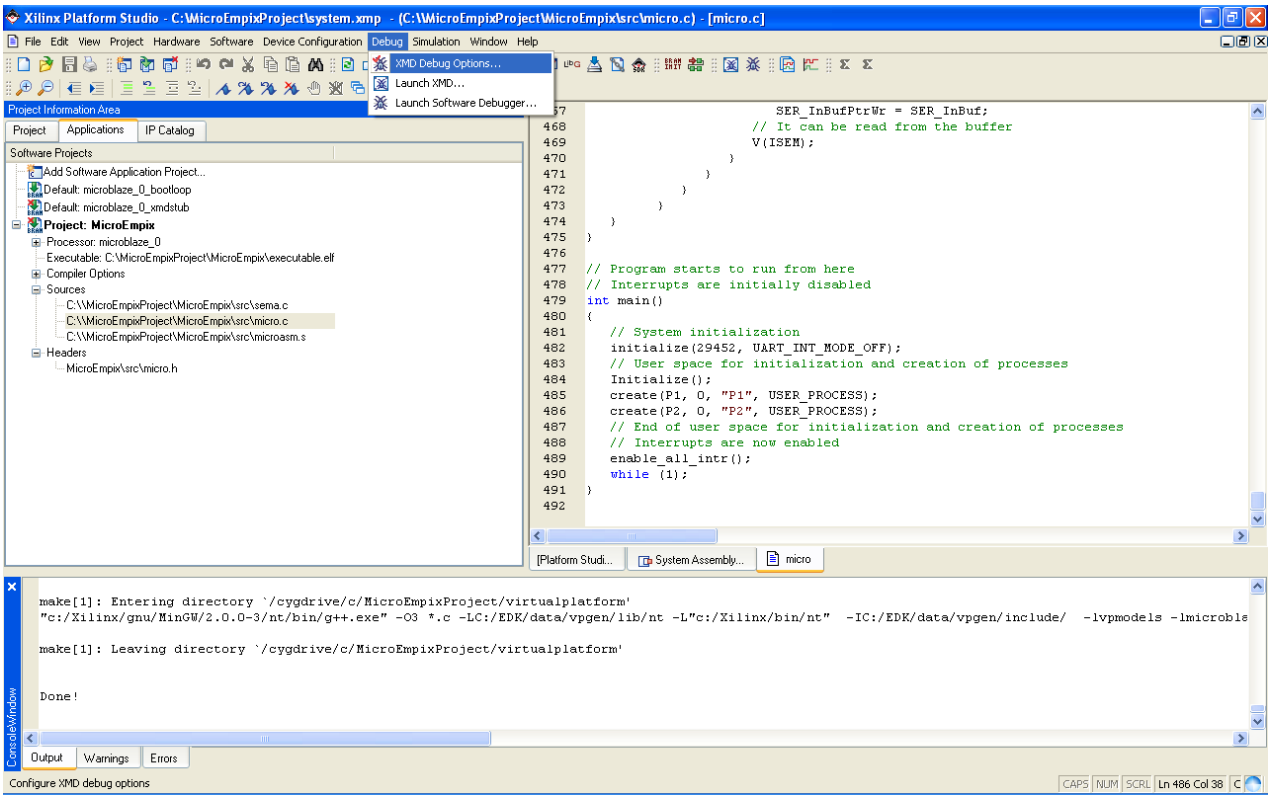
5.3 Προσομοίωση σε Εικονική Πλατφόρμα

Αφού πρώτα ο χρήστης έχει δημιουργήσει το εκτελέσιμο αρχείο του προγράμματος, για να ελέγξει και να αποσφαλματώσει, ενδεχομένως, την εφαρμογή του, πριν την κατεβάσει στην πλακέτα, μπορεί να κάνει προσομοίωση σε Εικονική Πλατφόρμα (Virtual Platform). Για το σκοπό αυτό επιλέγει αρχικά από το μενού “Simulation” το “Generate Virtual Platform”. Στη συνέχεια από το μενού “Debug” επιλέγει το “XMD Debug Options...”. Τότε ανοίγει ένα παράθυρο. Στο παράθυρο αυτό και συγκεκριμένα στο πεδίο Connection Type επιλέγει “Virtual platform”. Για έξοδο με αποθήκευση των

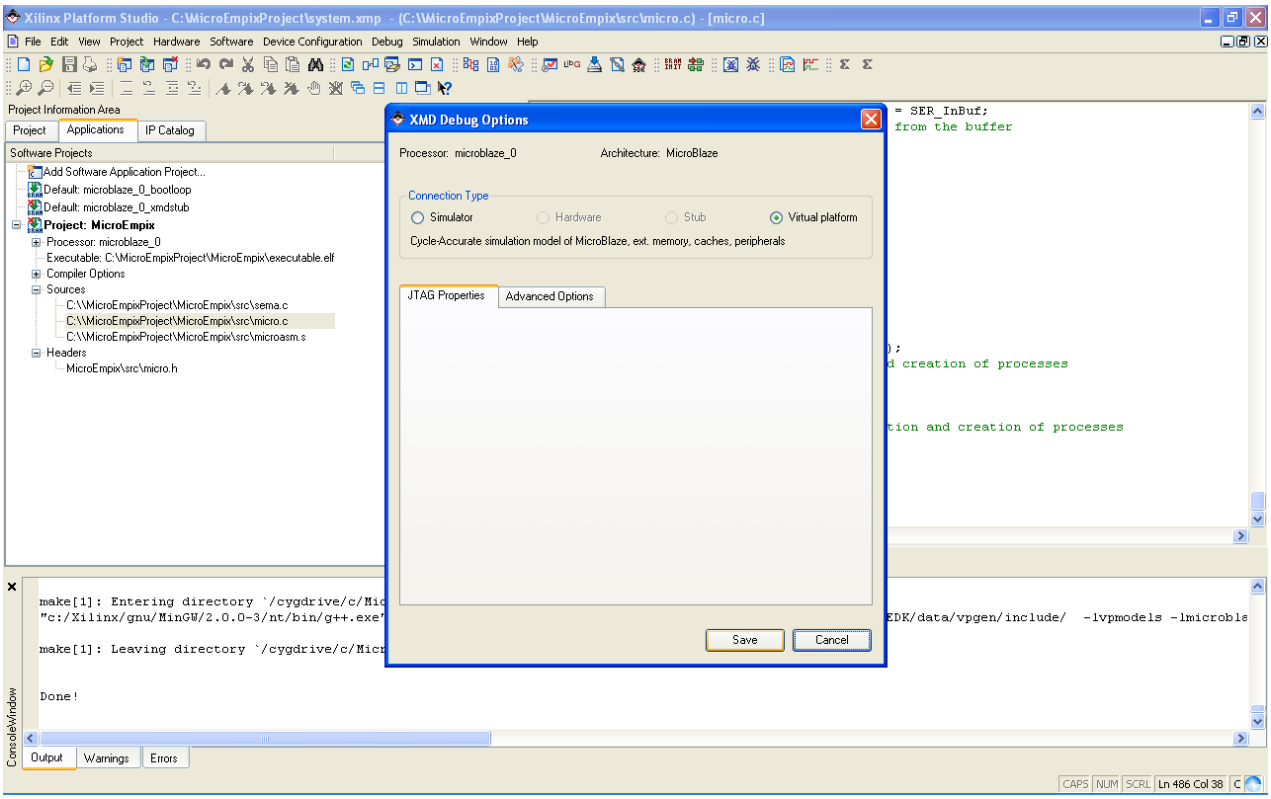
επιλογών του ο χρήστης πατάει το πλήκτρο “Save”. Για να ξεκινήσει το Xilinx Microprocessor Debugger επιλέγει από το μενού “Debug” την επιλογή “Launch XMD...”. Τα παραπάνω φαίνονται από το Σχήμα 5.21 έως και το Σχήμα 5.24.



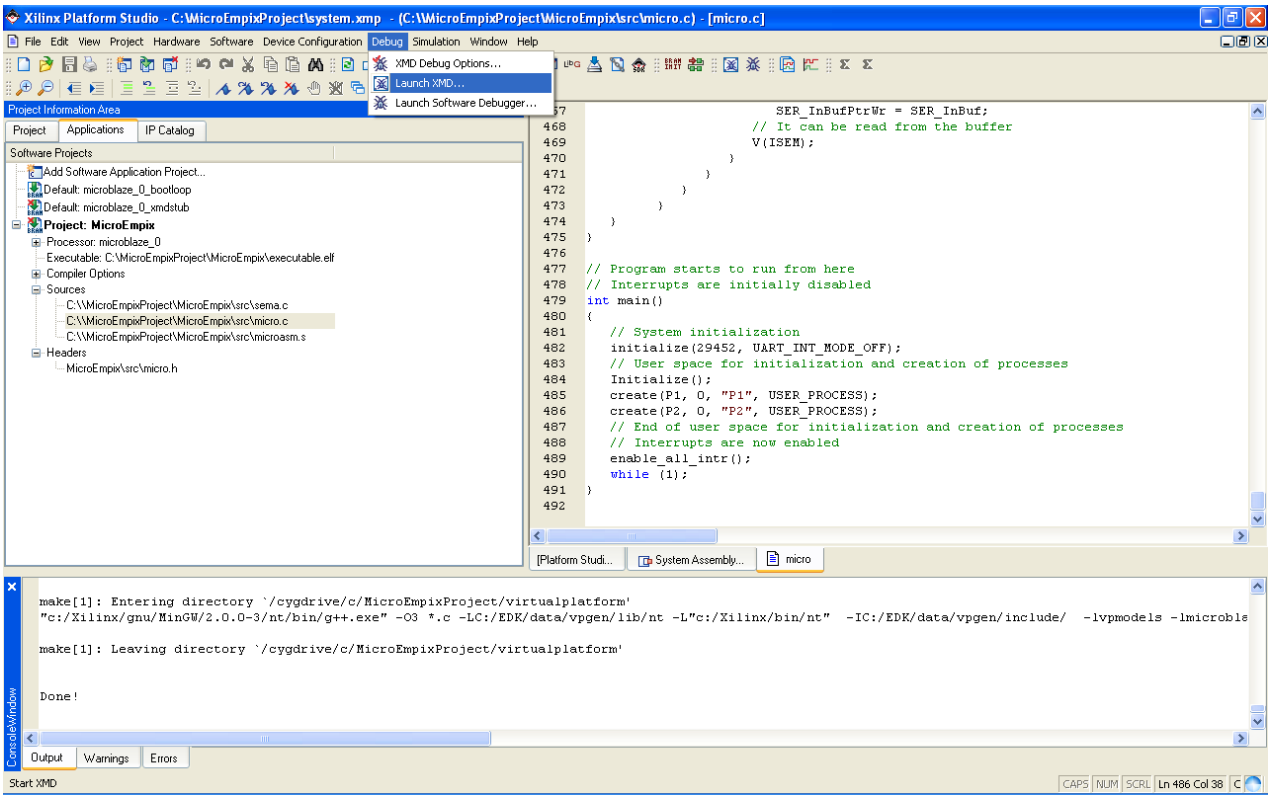
Σχήμα 5.21: Generate Virtual Platform



Σχήμα 5.22: XMD Debug Options

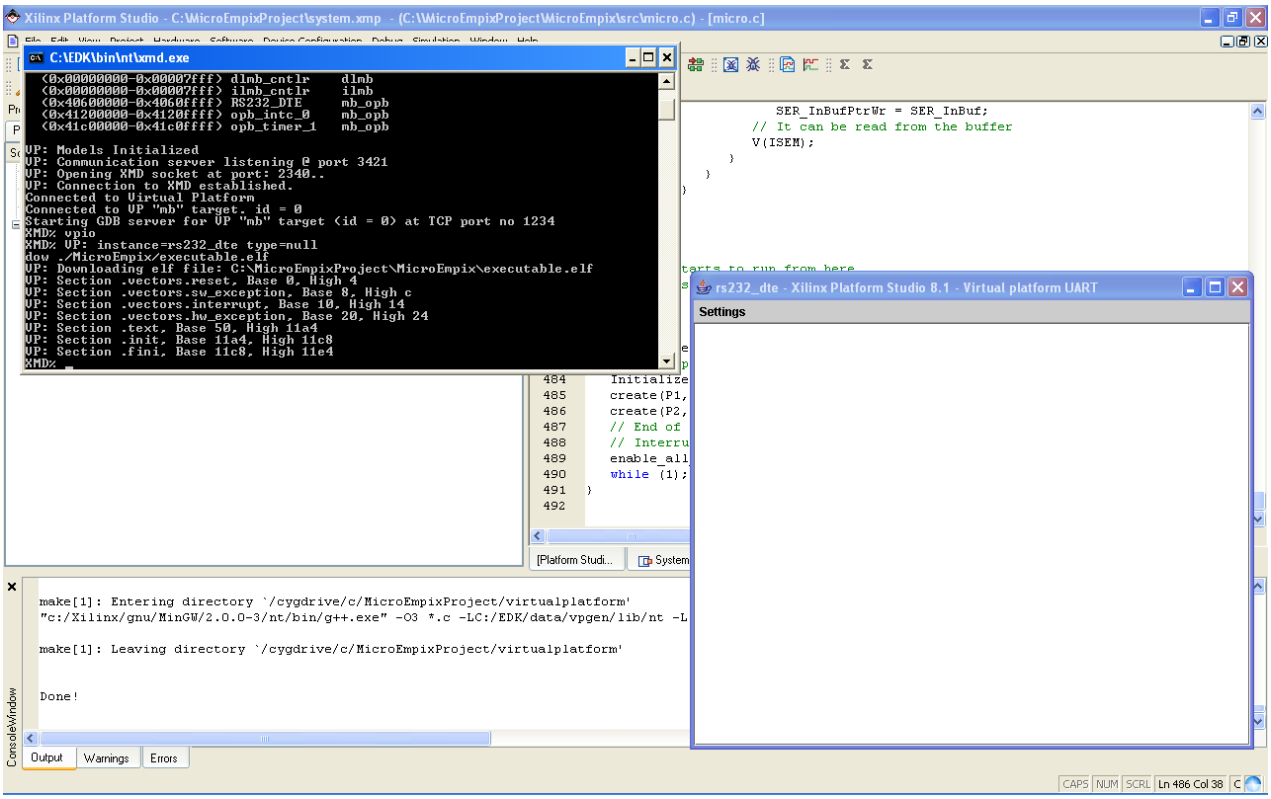


Σχήμα 5.23: Connection Type

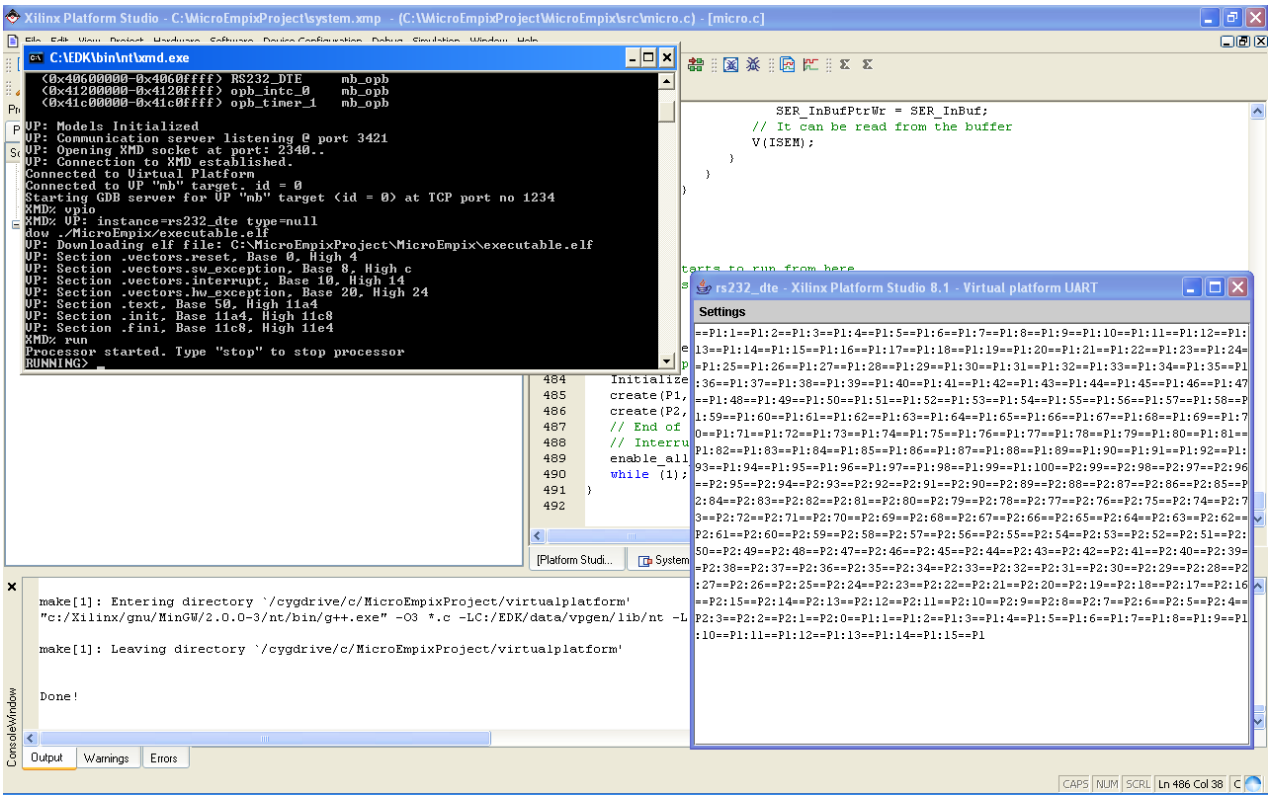


Σχήμα 5.24: Launch XMD

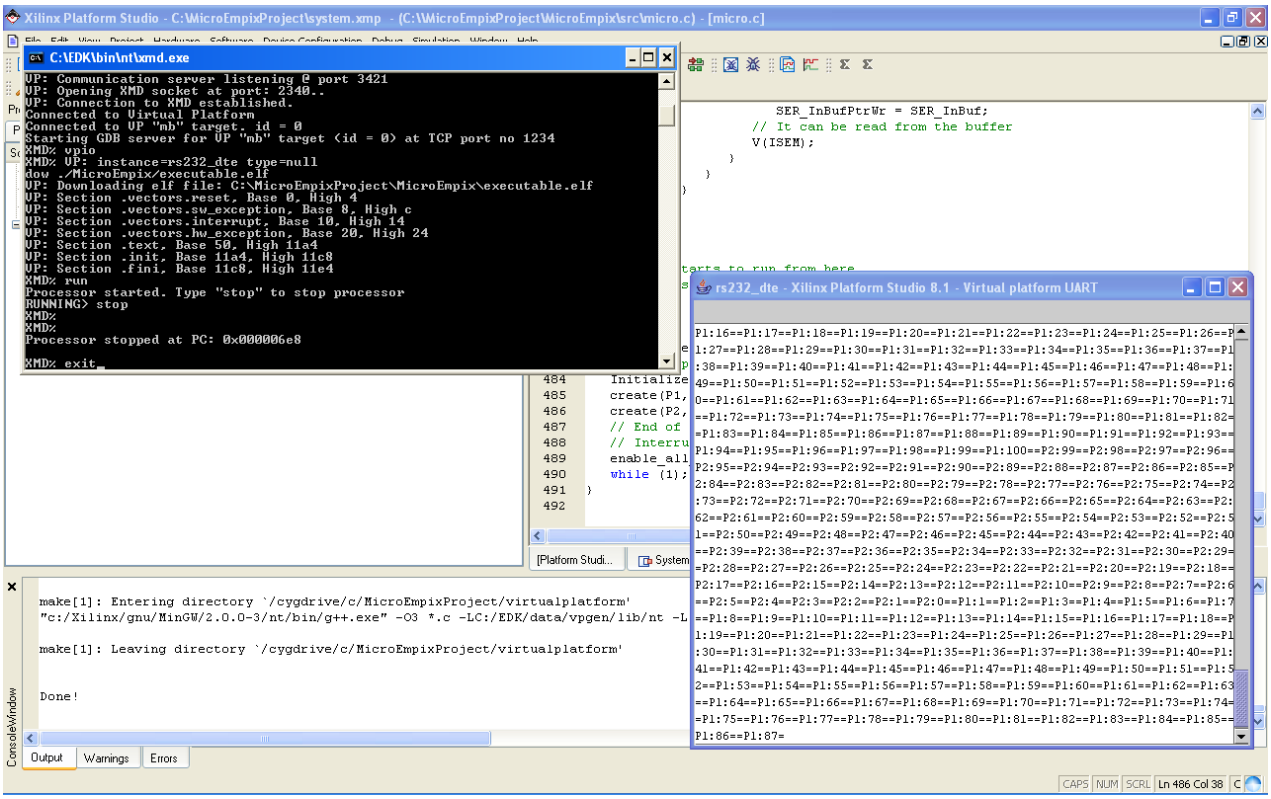
Πληκτρολογώντας τώρα vpio στη γραμμή εντολών που έχει ανοίξει ανοίγει ένα παράθυρο στο οποίο ο χρήστης βλέπει την έξοδο της σειριακής θύρας και στο οποίο δίνει την είσοδο της σειριακής θύρας. Για να κατεβάσει το εκτελέσιμο του προγράμματός του, πληκτρολογεί στη συνέχεια dow ./MicroEmpix/executable.elf. Πληκτρολογώντας τώρα run τρέχει το πρόγραμμα. Οποιαδήποτε στιγμή ο χρήστης μπορεί να το σταματήσει πληκτρολογώντας stop και να το συνεχίσει πληκτρολογώντας con. Επίσης, μπορεί να ξεκινήσει από την αρχή την εκτέλεση πληκτρολογώντας rst. Για περισσότερες πληροφορίες για την αποσφαλμάτωση ο αναγνώστης μπορεί να ανατρέξει στο "Embedded System Tools Reference Manual" στο κεφάλαιο "Xilinx Microprocessor Debugger". Για να ολοκληρώσει την προσομοίωση ο χρήστης πληκτρολογεί exit. Τα παραπάνω φαίνονται από το Σχήμα 5.25 έως και το Σχήμα 5.27. Στο Σχήμα 5.27 φαίνεται ότι το πρόγραμμα λειτουργεί ακριβώς όπως είχε προδιαγραφεί.



Σχήμα 5.25: Download



Σχήμα 5.26: Run

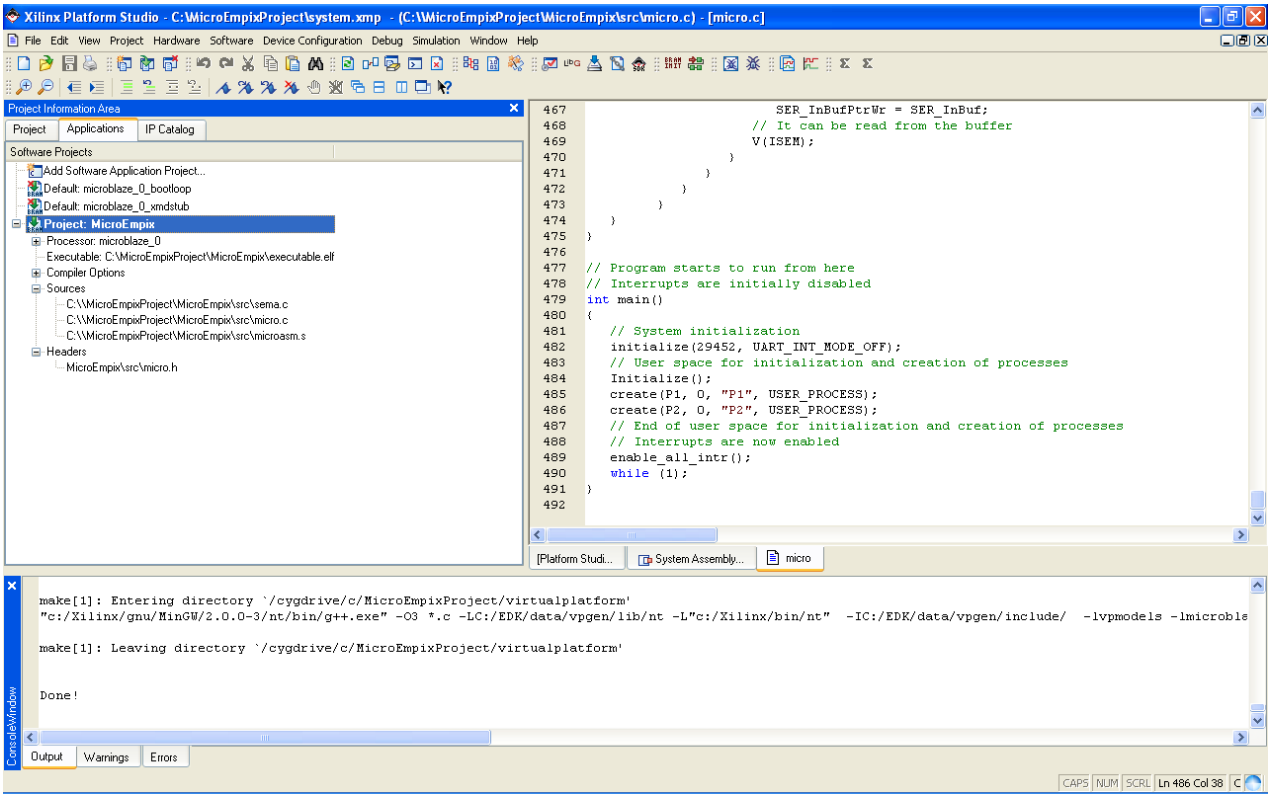


Σχήμα 5.27: Exit

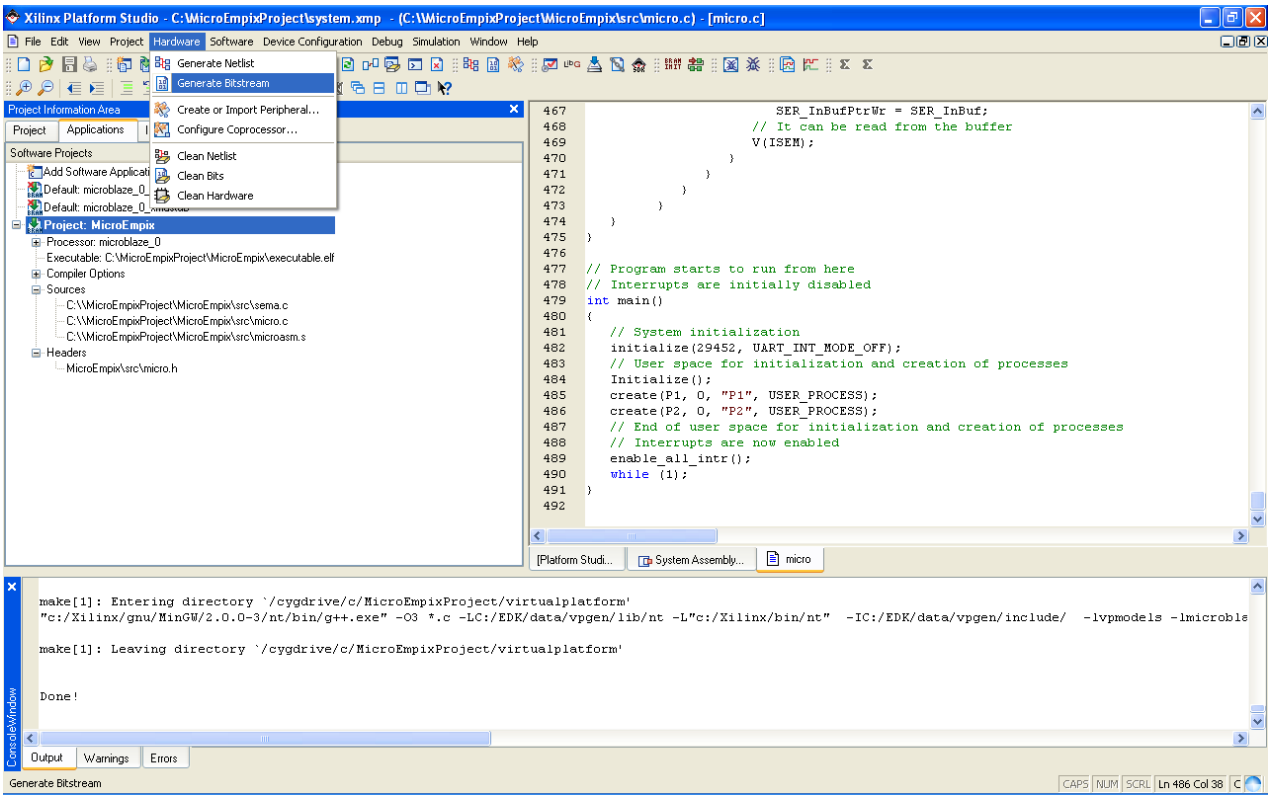
5.4 Προγραμματισμός του FPGA

Αφού με τη χρήση της προσομοίωσης ο χρήστης εξασφάλισε ότι η εφαρμογή του δουλεύει όπως αναμενόταν, μπορεί πλέον να κατεβάσει το project στην πλακέτα. Για να αρχικοποιηθεί η Block RAM με την εφαρμογή του, κάνει δεξί κλικ στο "Project: MicroEmpix" και στη συνέχεια κάνει κλικ στο "Mark to Initialize BRAMs". Στις υπόλοιπες εφαρμογές που υπάρχουν στο tab "Applications" απενεργοποιεί, αν δεν είναι ήδη απενεργοποιημένη, με τον ίδιο τρόπο τη συγκεκριμένη επιλογή. Κατόπιν από το μενού "Hardware" επιλέγει το "Generate Bitstream". Στη συνέχεια από το "Device Configuration" επιλέγει το "Update Bitstream". Σε αυτό το σημείο θα πρέπει ο υπολογιστής στον οποίο εκτελείται το Xilinx Platform Studio να έχει συνδεθεί μέσω της θύρας USB με το Spartan-3E Starter Board για τον προγραμματισμό της πλακέτας και το Spartan-3E Starter Board να έχει συνδεθεί μέσω της σειριακής θύρας με κάποιον υπολογιστή στον οποίο θα τρέχει το πρόγραμμα HyperTerminal. Η πλακέτα θα πρέπει να είναι, τέλος, σε λειτουργία. Από το μενού "Device Configuration" ο χρήστης επιλέγει, τελικά, το "Download Bitstream". Το project κατεβαίνει, τότε, στην πλακέτα και το MicroEmpix/FPGA αρχίζει να εκτελείται κατευθείαν. Από το Σχήμα 5.28 έως και το Σχήμα 5.31

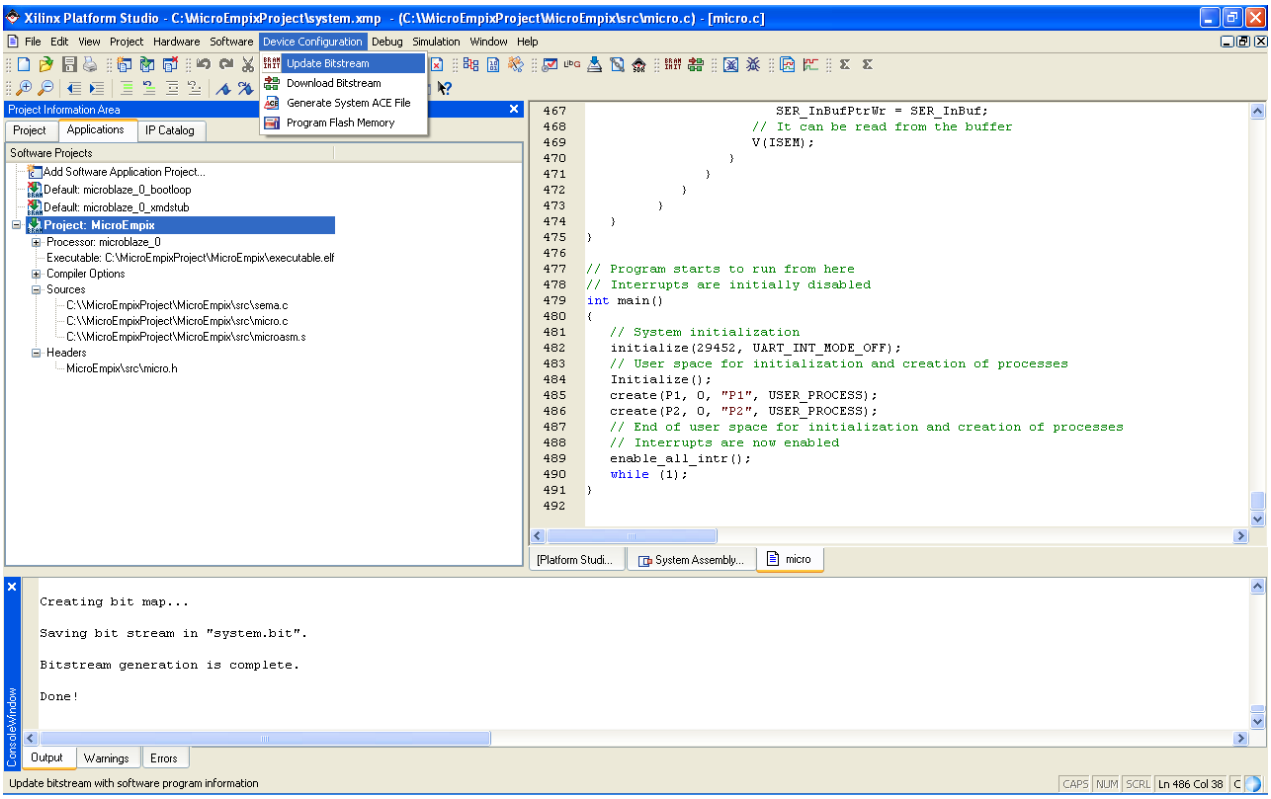
φαίνεται η παραπάνω διαδικασία ενώ στο Σχήμα 5.32 φαίνονται τα αποτελέσματα της εκτέλεσης στο HyperTerminal, το οποίο επικοινωνεί μέσω της σειριακής θύρας με το Spartan-3E Starter Board. Η ρύθμιση του HyperTerminal για την εκτέλεση της λειτουργίας αυτής θα περιγραφεί αμέσως μετά.



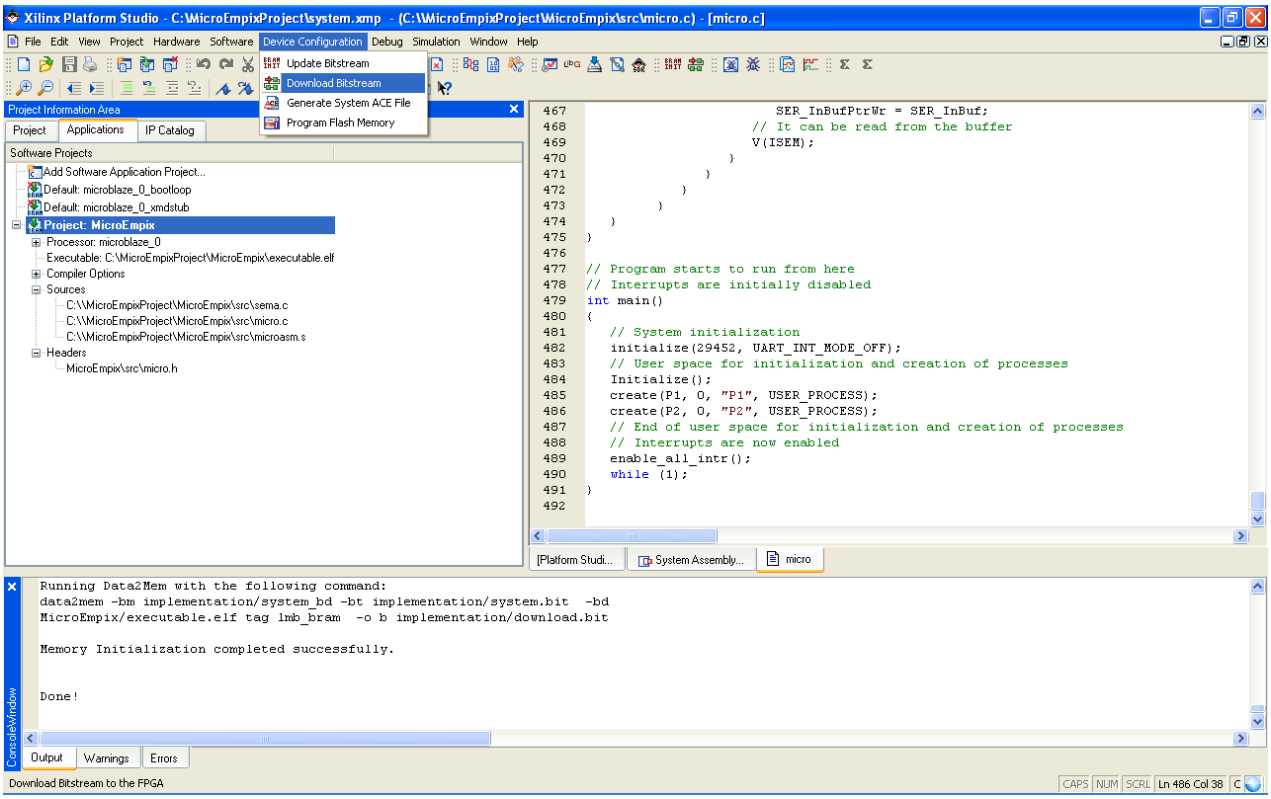
Σχήμα 5.28: Mark to Initialize BRAMs



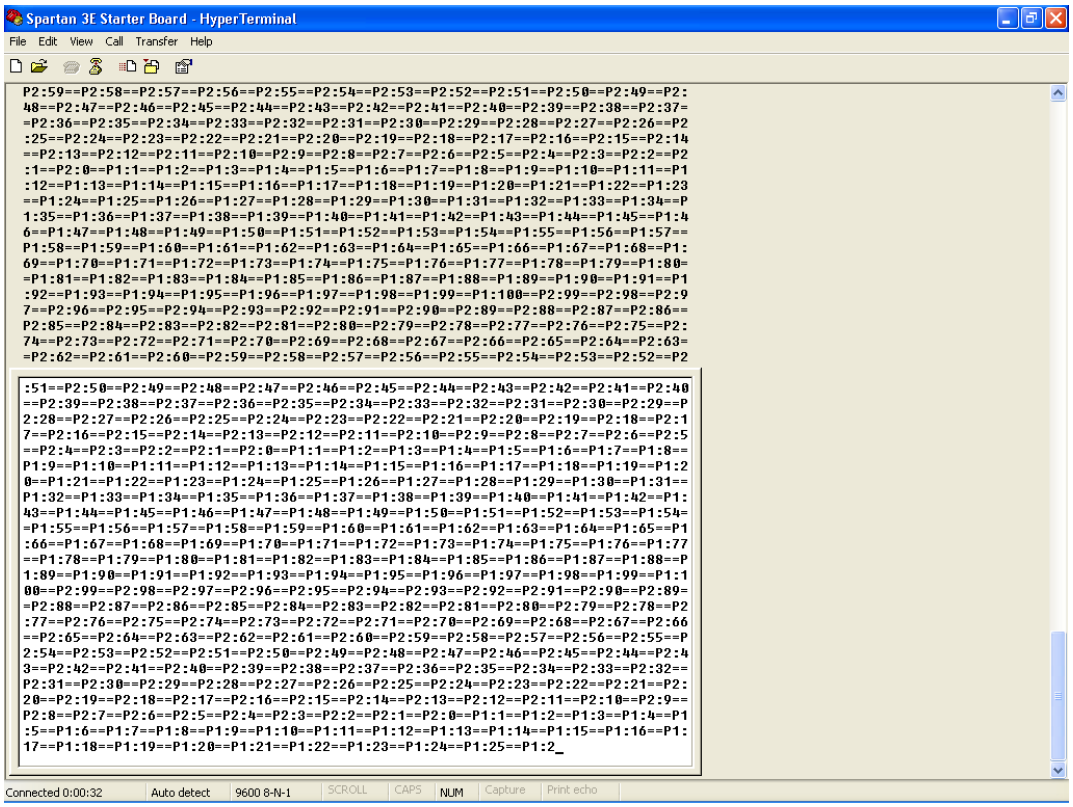
Σχήμα 5.29: Generate Bitstream



Σχήμα 5.30: Update Bitstream



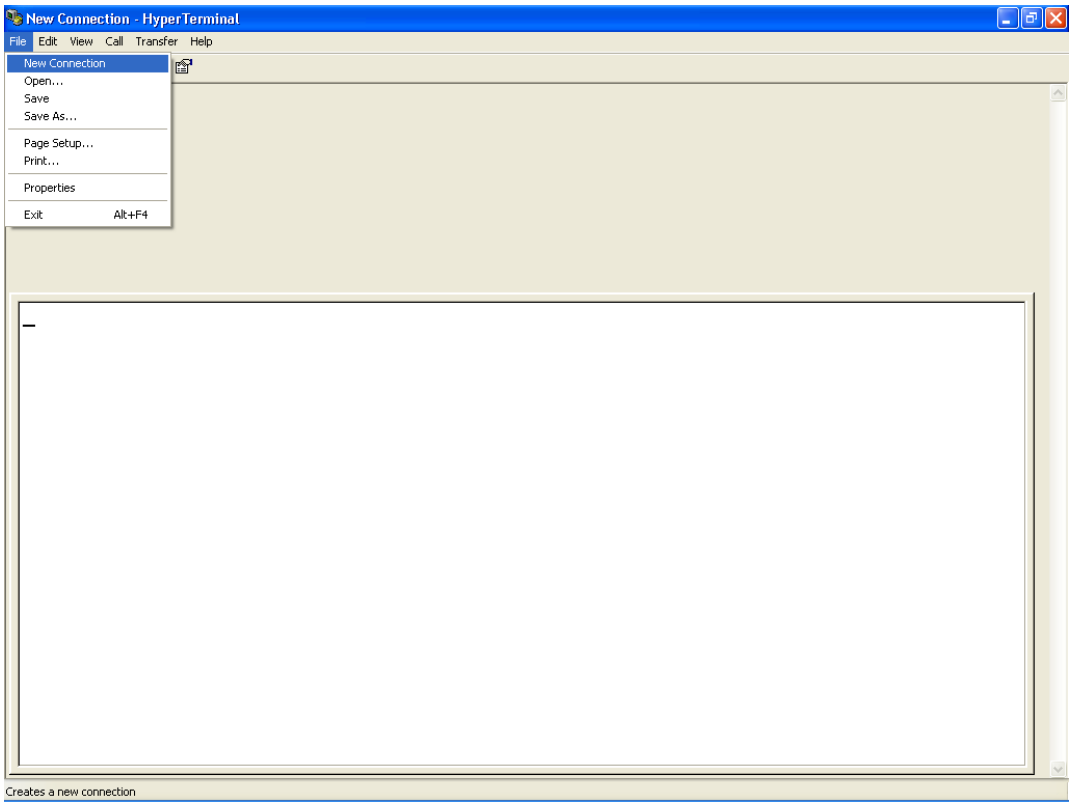
Σχήμα 5.31: Download Bitstream



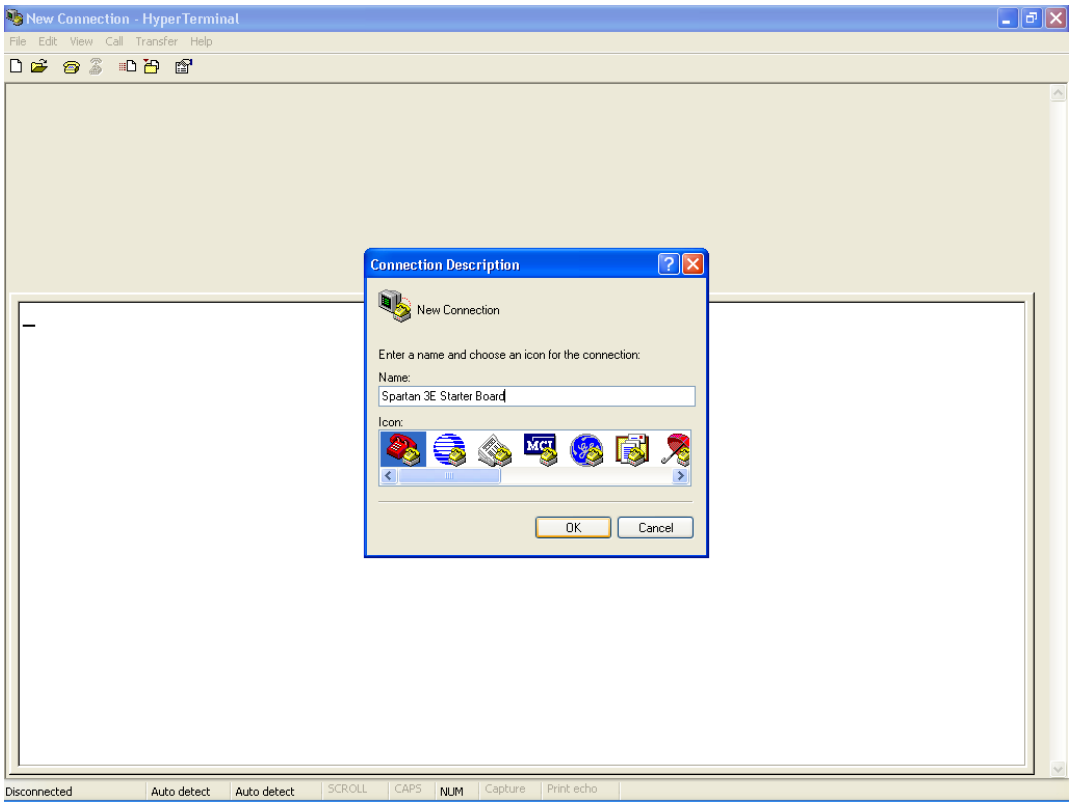
Σχήμα 5.32: Αποτελέσματα Εκτέλεσης

5.5 Ρύθμιση HyperTerminal

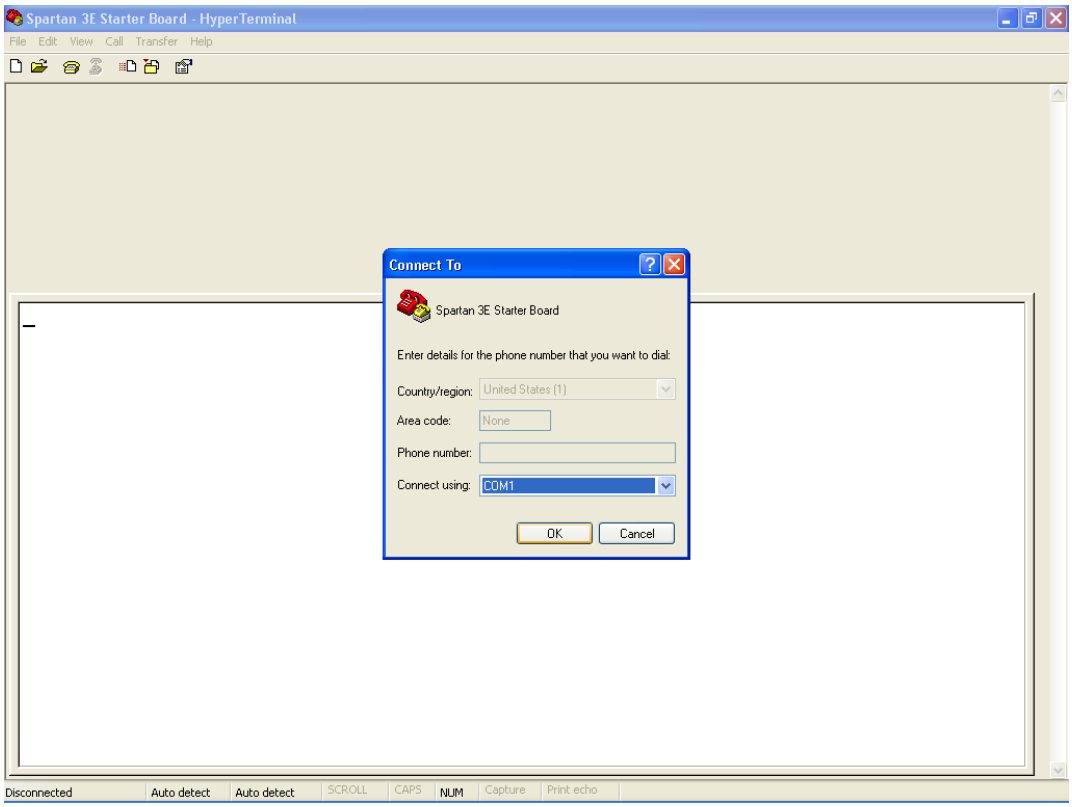
Η ρύθμιση του HyperTerminal για την επικοινωνία με το Spartan-3E Starter Board μέσω της σειριακής θύρας που χρησιμοποιούν οι εφαρμογές φαίνεται από το Σχήμα 5.33 έως και το Σχήμα 5.36. Αφού ο χρήστης έχει ανοίξει το HyperTerminal, επιλέγει από το μενού "File" το "New Connection". Στο παράθυρο που ανοίγει δίνει ένα όνομα στη νέα σύνδεση, εν προκειμένω Spartan 3E Starter Board, και πατάει το πλήκτρο "OK". Στο παράθυρο με τον τίτλο "Connect To" που ανοίγει επιλέγει στο πεδίο Connect using: "COM1", για να χρησιμοποιήσει την COM1 για σύνδεση με την πλακέτα. Πατάει "OK" για συνέχεια. Στο νέο παράθυρο με τίτλο "COM1 Properties" ρυθμίζει τη σειριακή θύρα ακριβώς όπως έχει ρυθμιστεί το περιφερειακό OPB UART Lite, δηλαδή Bits per second: "9600", Data bits: "8", Parity: "None", Stop bits: "1" και Flow control: "None". Πατώντας το πλήκτρο "OK" γίνεται η σύνδεση και μπορεί πλέον να γίνει αποστολή και λήψη χαρακτήρων μέσω της σειριακής θύρας.



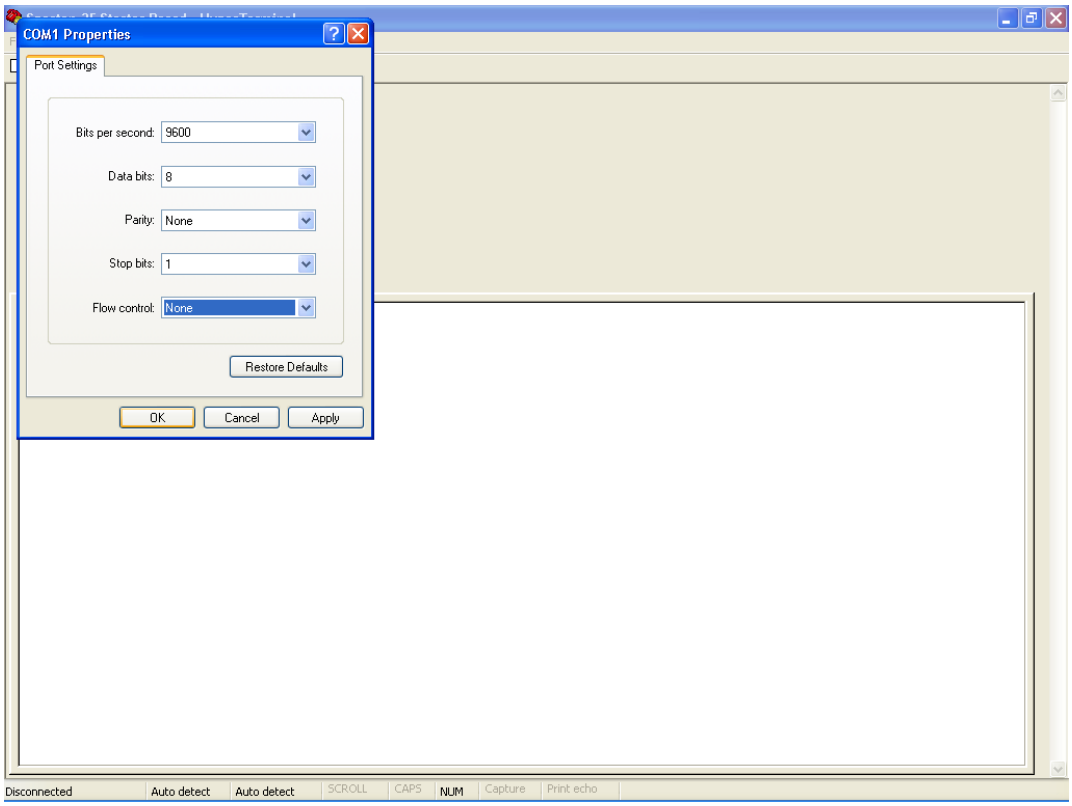
Σχήμα 5.33: New Connection



Σχήμα 5.34: Connection Description



Σχήμα 5.35: Connect To

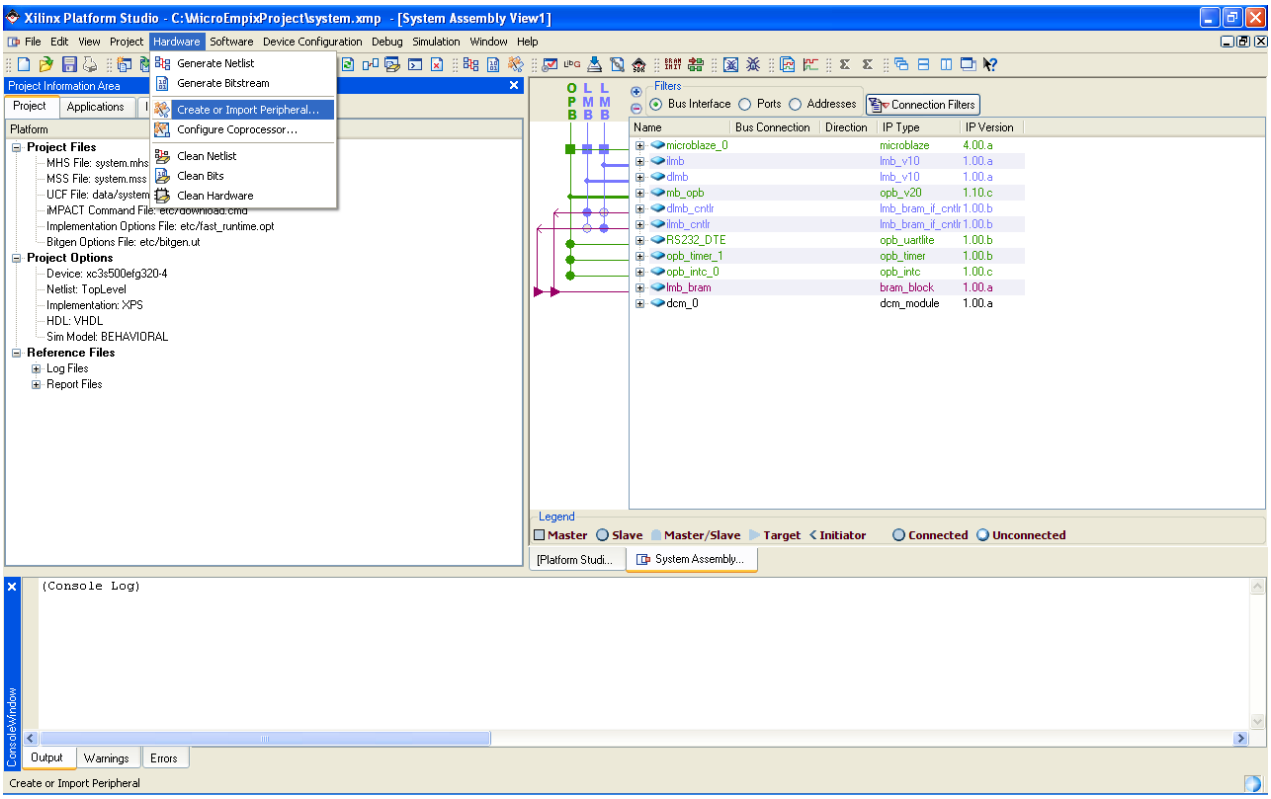


Σχήμα 5.36: Port Settings

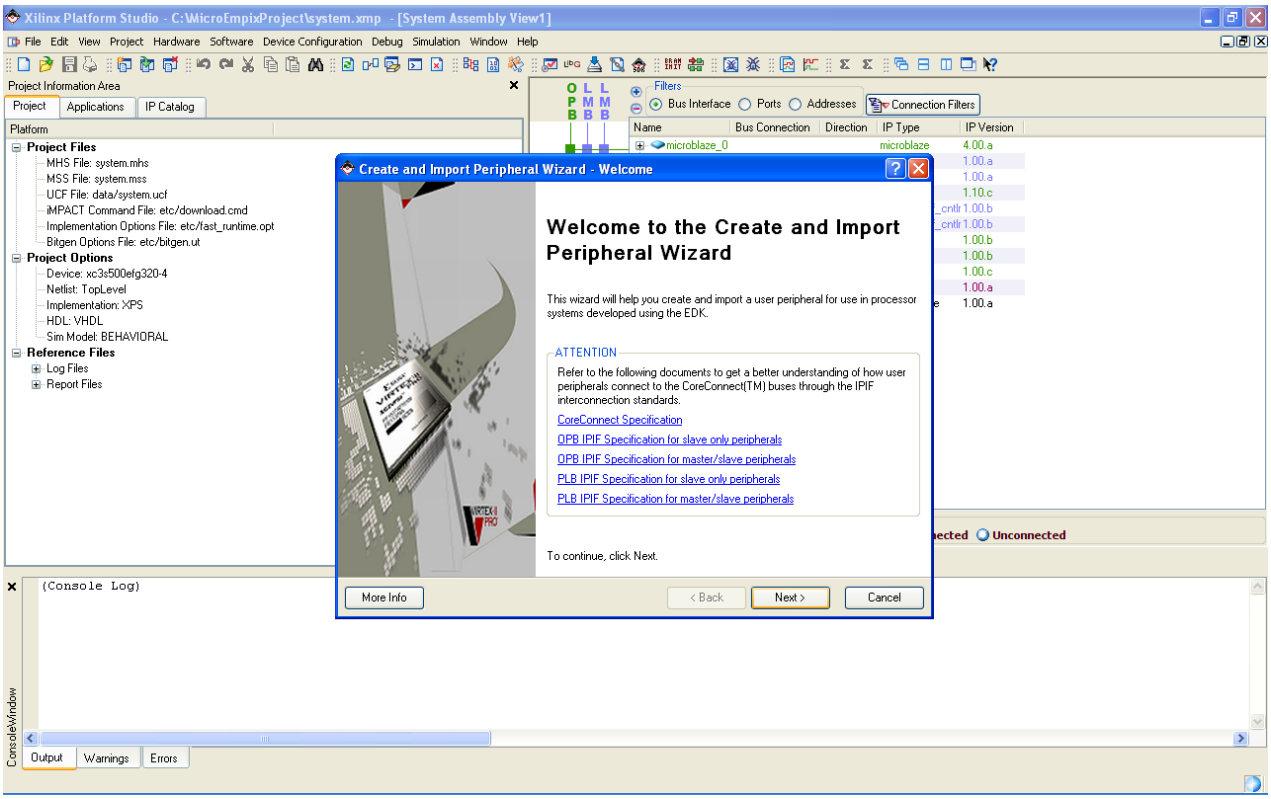
5.6 Δημιουργία και Χρήση Περιφερειακού

5.6.1 Δημιουργία Περιφερειακού

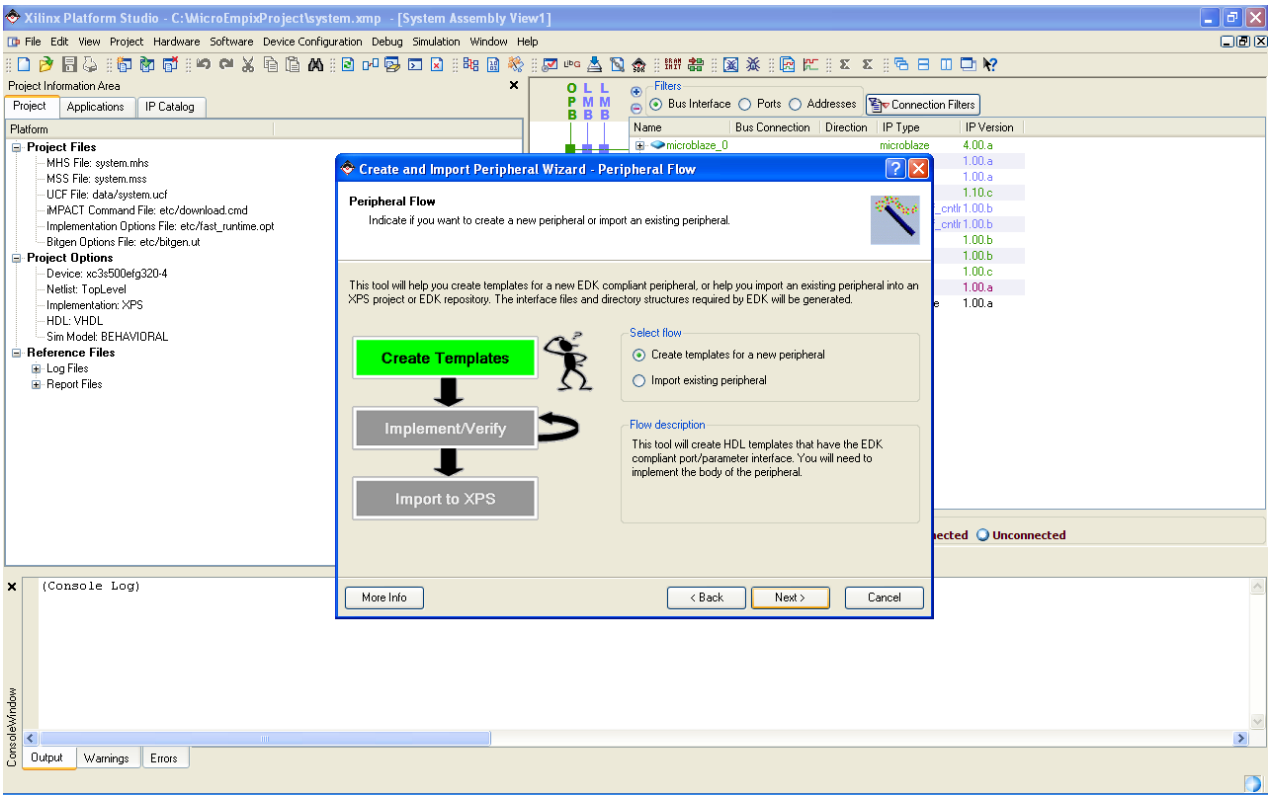
Σε αυτό το σημείο ο χρήστης θα δημιουργήσει ένα περιφερειακό για το FSL, δηλαδή ένα συνεπεξεργαστή. Αρχικά από το μενού “Hardware” επιλέγει “Create or Import Peripheral...”. Στα παράθυρα που ανοίγουν πατάει τρεις φορές το πλήκτρο “Next >”, για να δημιουργήσει ένα νέο περιφερειακό και να το αποθηκεύσει στο φάκελο MicroEmpixProject. Τα παραπάνω φαίνονται από το Σχήμα 5.37 έως και το Σχήμα 5.40.



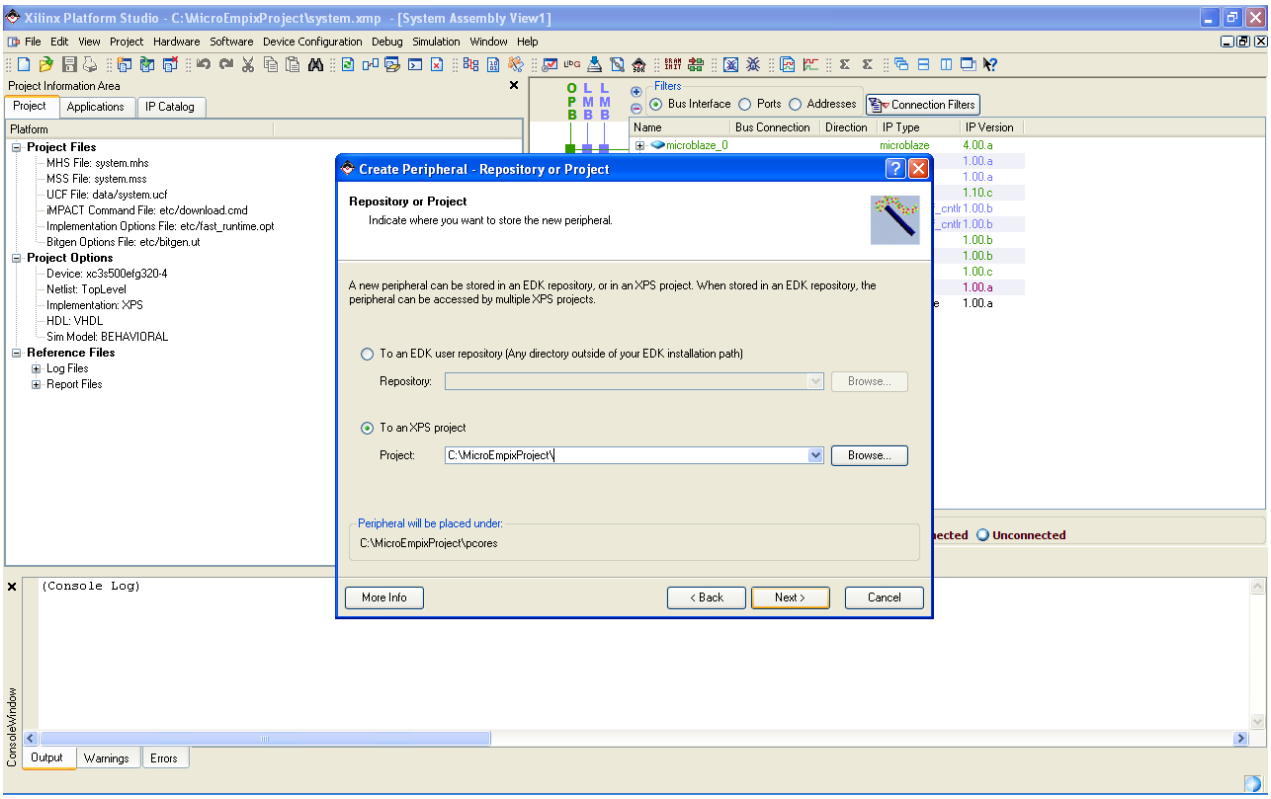
Σχήμα 5.37: Create or Import Peripheral



Σχήμα 5.38: Welcome to the Create and Import Peripheral Wizard

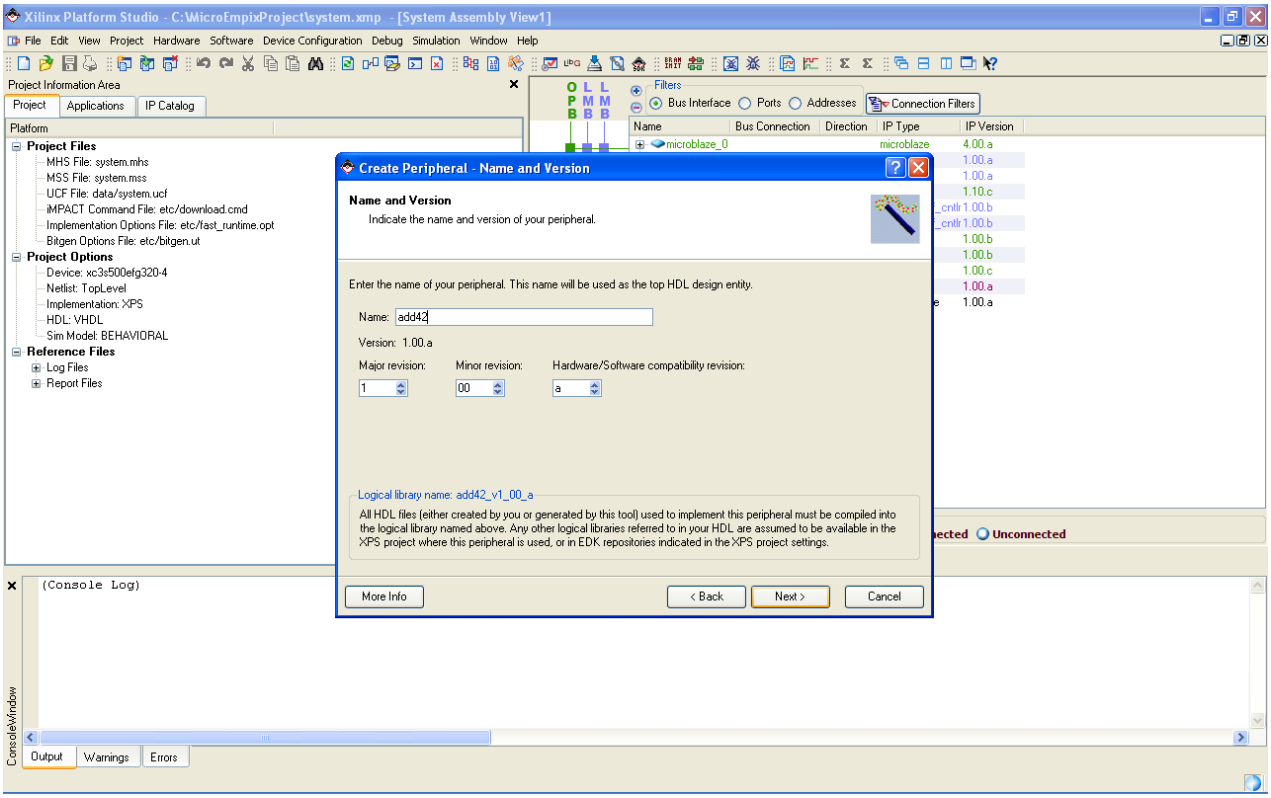


Σχήμα 5.39: New Peripheral

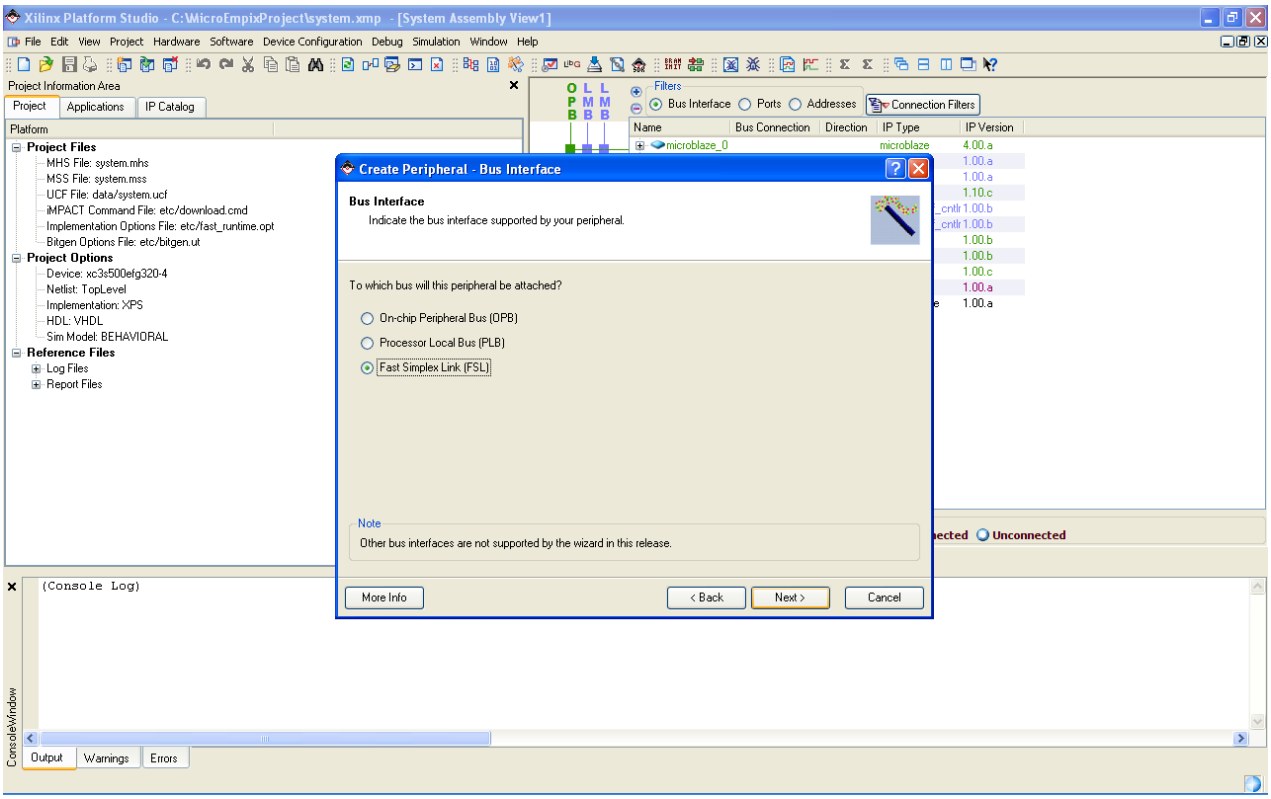


Σχήμα 5.40: Store in Project

Στο τρέχον παράθυρο ο χρήστης θέτει στο πεδίο Name το όνομα του περιφερειακού. Στη συγκεκριμένη περίπτωση ονομάζει το περιφερειακό add42, καθώς σκοπός του θα είναι η έξοδός του να ισούται με την είσοδό του συν 42. Πατάει “Next >” για συνέχεια. Στο παράθυρο στο οποίο βρίσκεται τώρα επιλέγει “Fast Simplex Link (FSL)” και πατάει το πλήκτρο “Next >” για συνέχεια. Τα παραπάνω φαίνονται στο Σχήμα 5.41 και στο Σχήμα 5.42.

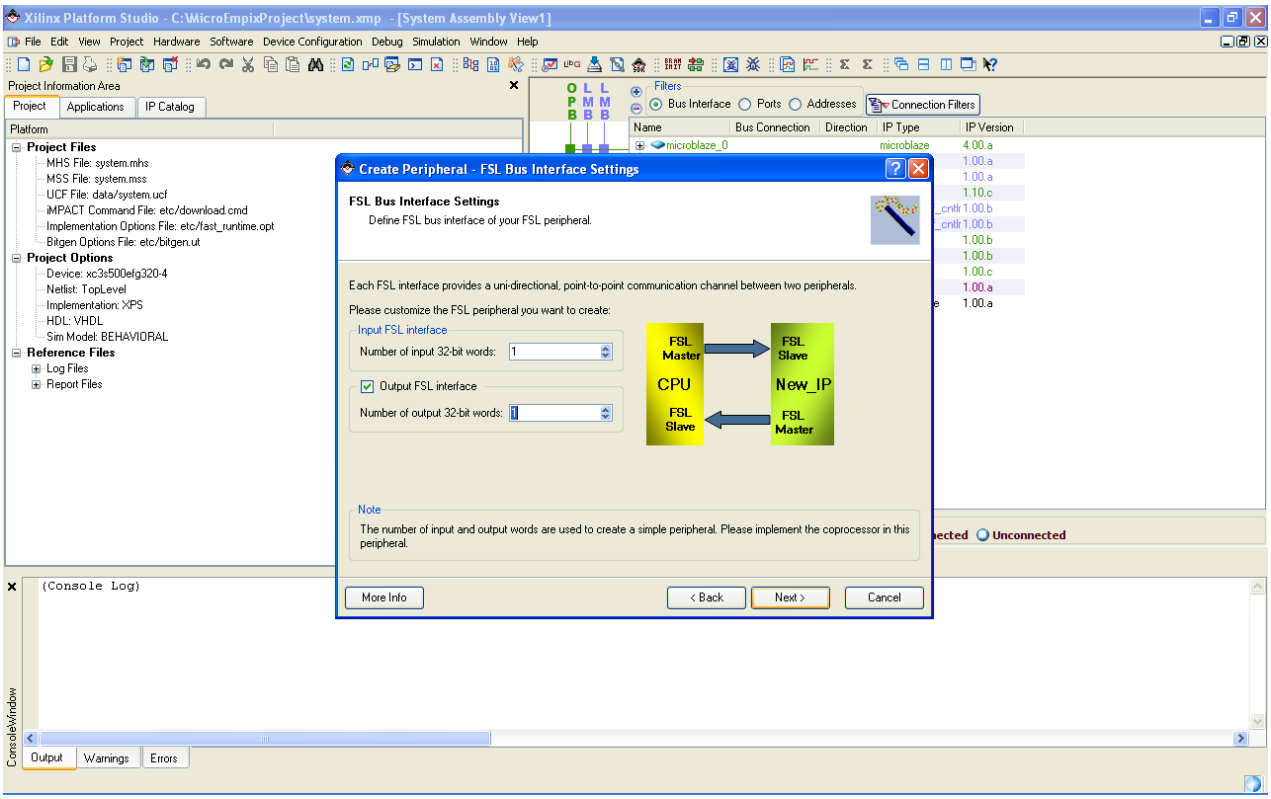


Σχήμα 5.41: Name and Version

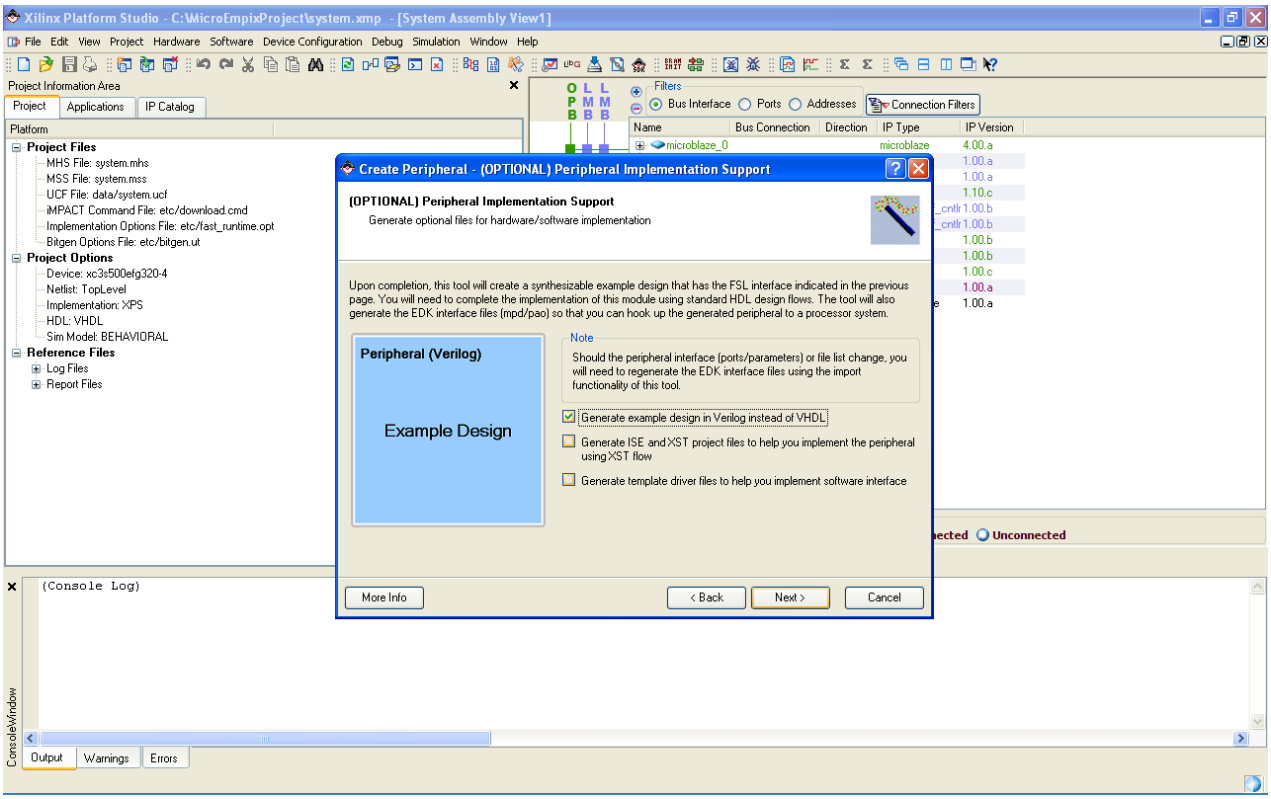


Σχήμα 5.42: Bus Interface

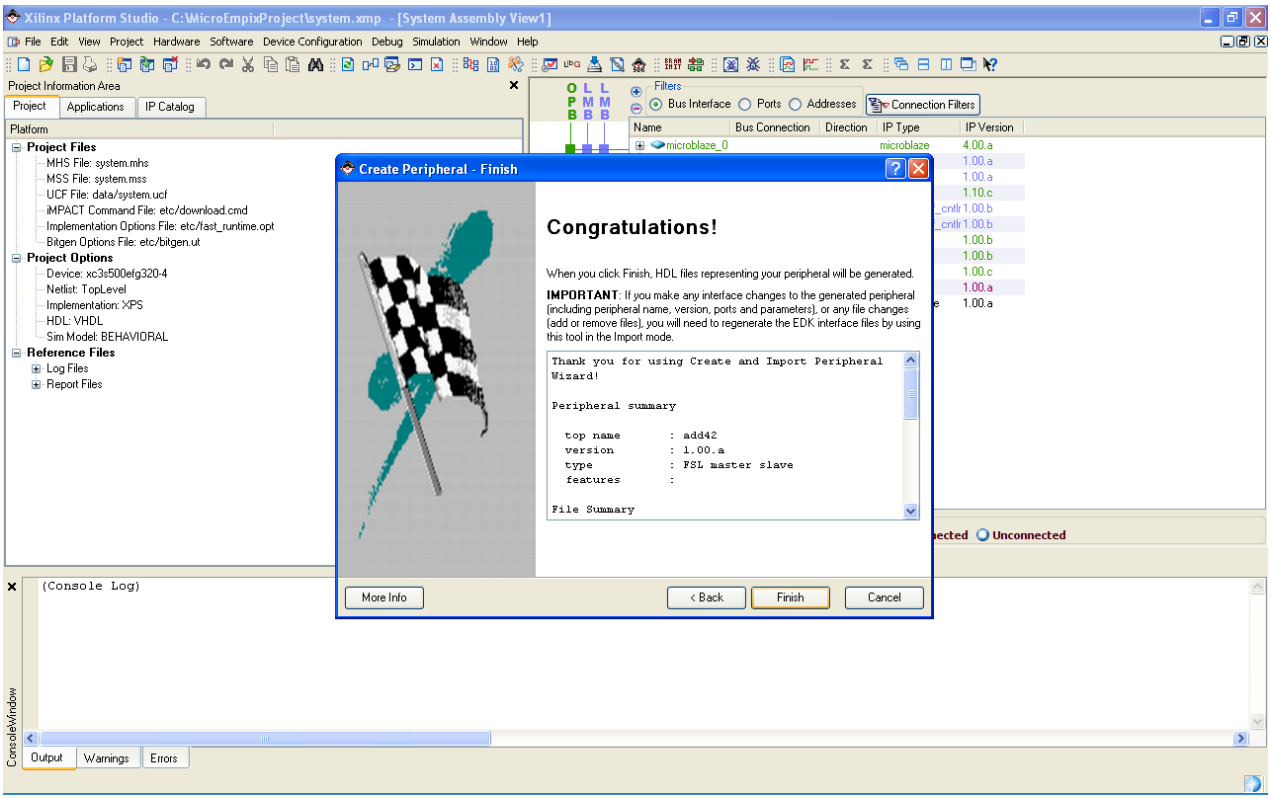
Στο τρέχον παράθυρο ο χρήστης επιλέγει το πλήθος των 32-bit λέξεων εισόδου και εξόδου του περιφερειακού. Στη συγκεκριμένη περίπτωση θέτει και τα δύο νούμερα στο 1 και πατάει “Next >” για συνέχεια. Στο παράθυρο στο οποίο βρίσκεται αυτή τη στιγμή μαρκάρει την πρώτη επιλογή και ξεμαρκάρει τη δεύτερη και την τρίτη επιλογή. Για συνέχεια πατάει “Next >” και στο νέο παράθυρο πατάει το πλήκτρο “Finish”, για να ολοκληρώσει τη δημιουργία του περιφερειακού. Τα παραπάνω φαίνονται από το Σχήμα 5.43 έως και το Σχήμα 5.45. Με τις συγκεκριμένες ρυθμίσεις θα παραχθεί ένα example design μίας εισόδου και μίας εξόδου σε Verilog. Για την υλοποίηση του ζητούμενου περιφερειακού θα πρέπει να τροποποιηθεί ο κώδικας του example design.



Σχήμα 5.43: FSL Bus Interface Settings



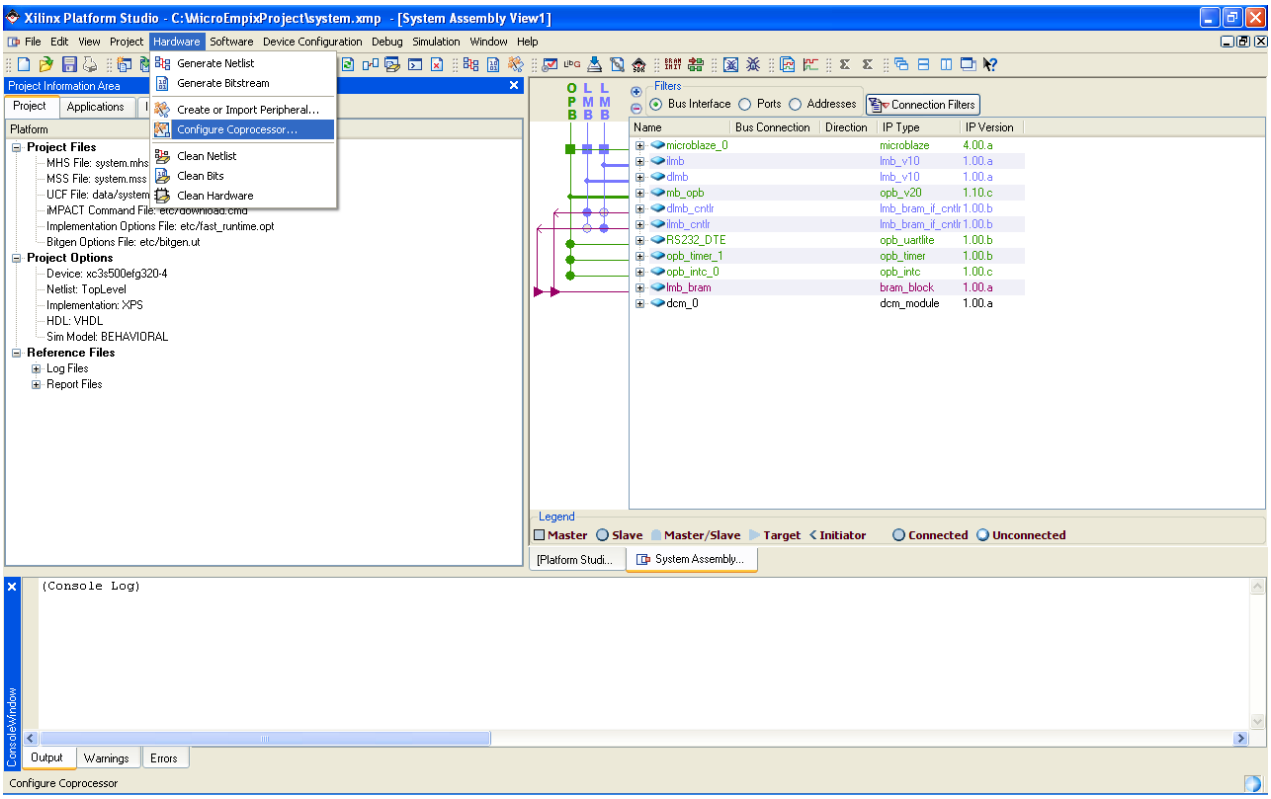
Σχήμα 5.44: Peripheral Implementation Support



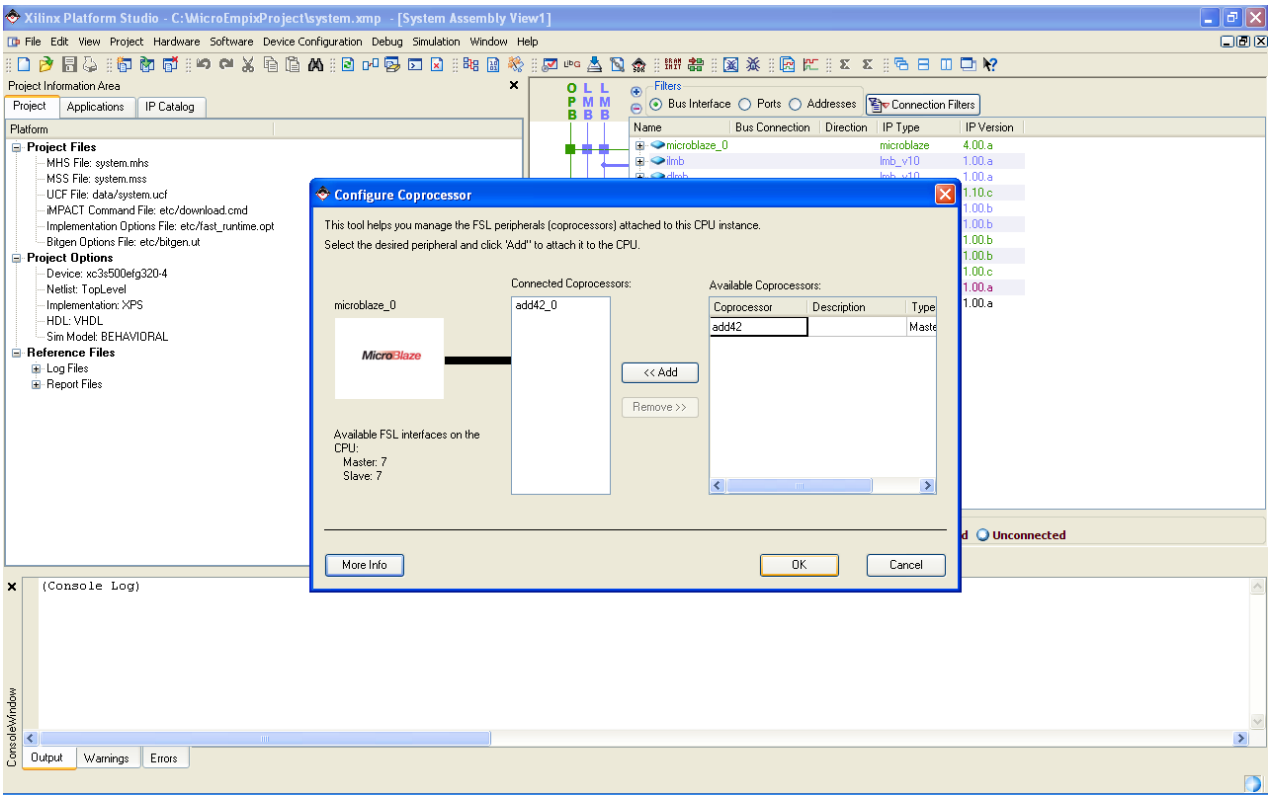
Σχήμα 5.45: Finish

5.6.2 Ρύθμιση Συνεπεξεργαστή

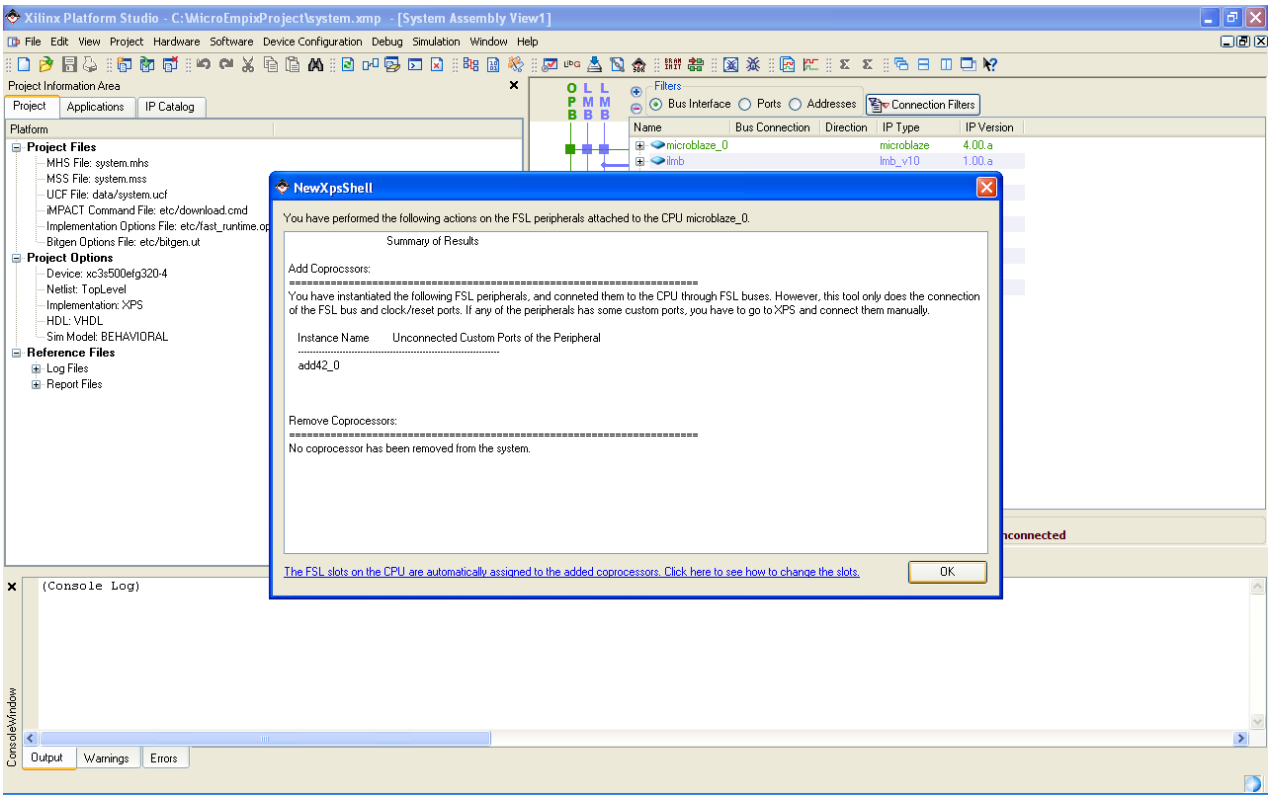
Για την εισαγωγή και τη σύνδεση του περιφερειακού στο σύστημα ο χρήστης επιλέγει από το μενού “Hardware” το “Configure Coprocessor...”. Στο παράθυρο που ανοίγει επιλέγει το add42, πατάει το πλήκτρο “<< Add” και κατόπιν το “Next >”. Στο τρέχον παράθυρο πατάει το πλήκτρο “OK” για τέλος. Τα παραπάνω φαίνονται από το Σχήμα 5.46 έως και το Σχήμα 5.48. Το στιγμιότυπο add42_0 του περιφερειακού add42 συνδέεται τότε στη διαπροσωπεία FSL0 του MicroBlaze.



Σχήμα 5.46: Configure Coprocessor



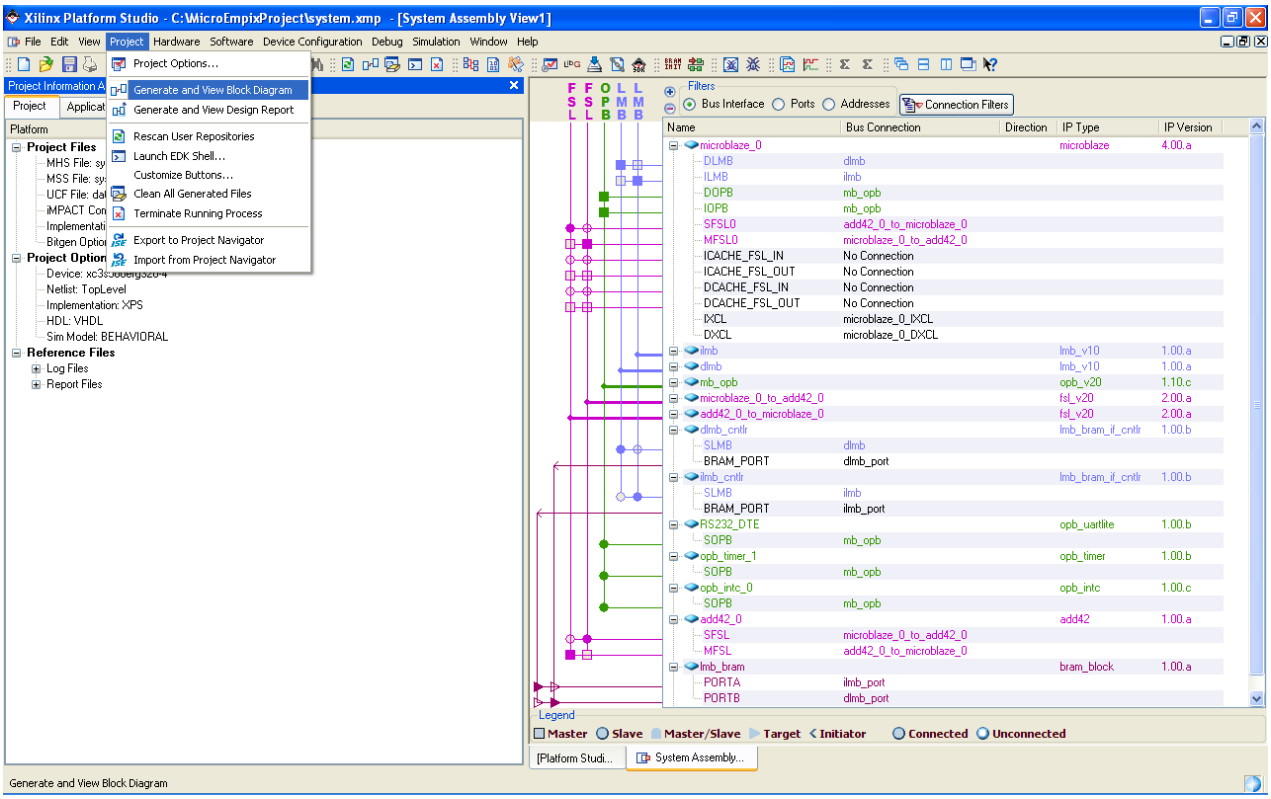
Σχήμα 5.47: Add Coprocessor



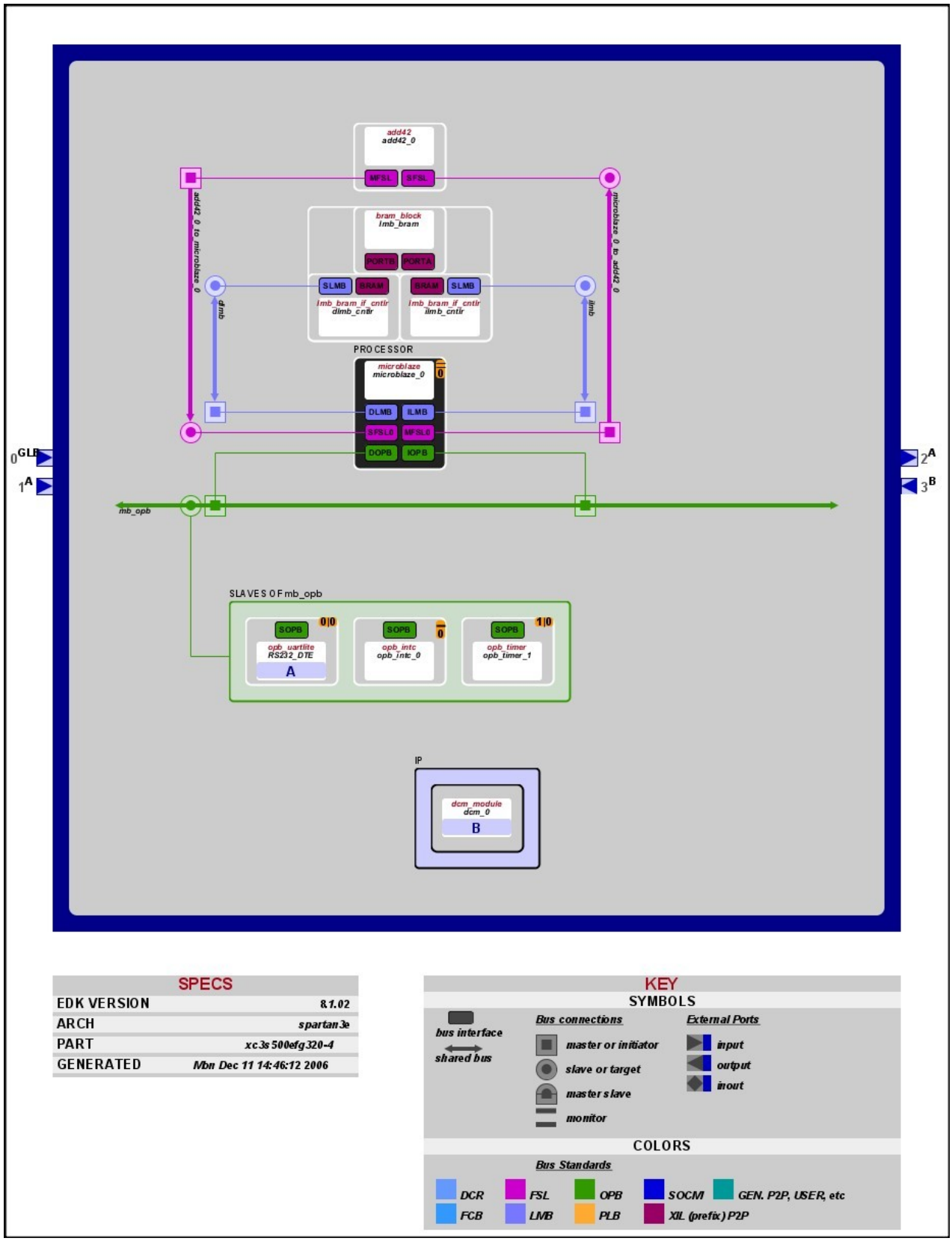
Σχήμα 5.48: Coprocessor Added

5.6.3 Διάγραμμα Συστήματος

Από το μενού “Project” ο χρήστης επιλέγει “Generate and View Block Diagram”. Αυτό φαίνεται στο Σχήμα 5.49, όπου φαίνονται, επίσης, γραφικά οι συνδέσεις ανάμεσα στις συνιστώσες του συστήματος. Το διάγραμμα του συστήματος φαίνεται στο Σχήμα 5.50.



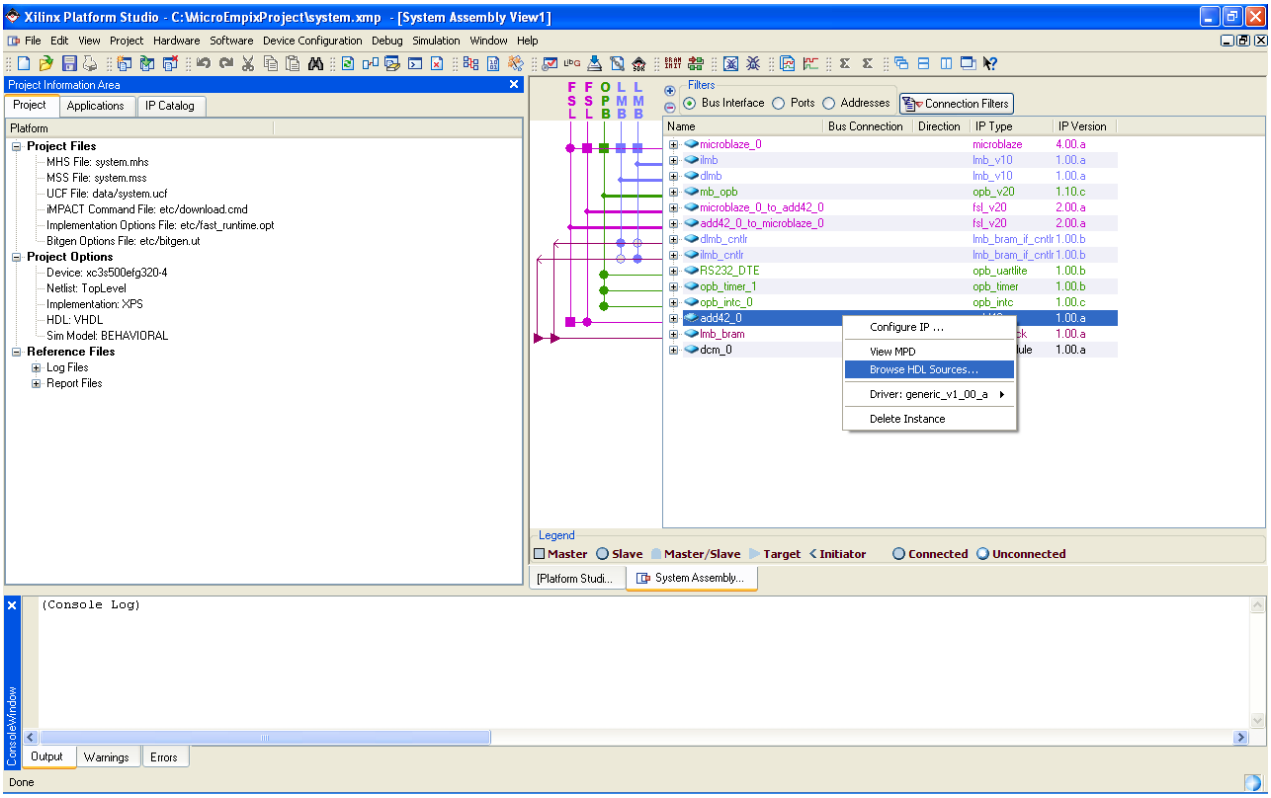
Σχήμα 5.49: Generate and View Block Diagram



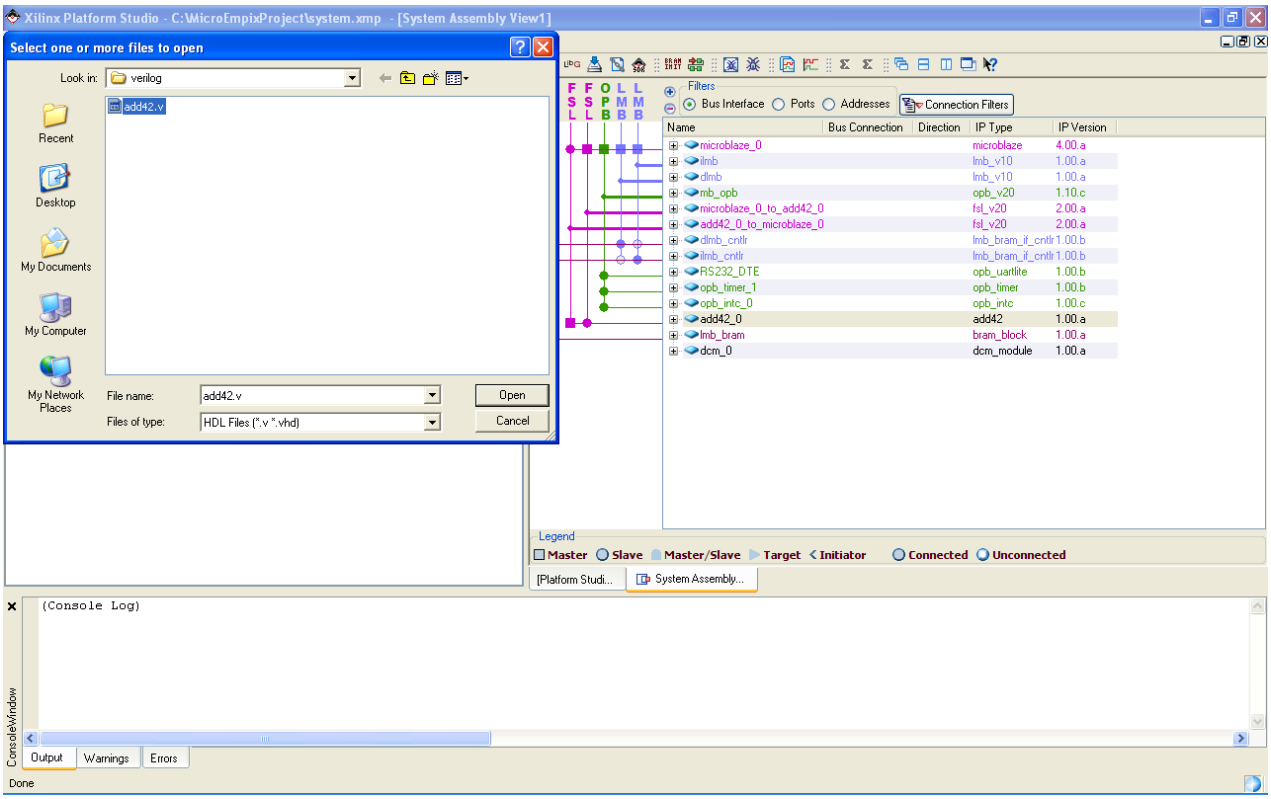
Σχήμα 5.50: Block Diagram

5.6.4 Υλοποίηση Περιφερειακού

Για να υλοποιήσει ο χρήστης τη ζητούμενη λειτουργία στο περιφερειακό του, πρέπει να τροποποιήσει τον κώδικά του στη γλώσσα περιγραφής υλικού στην οποία αυτό έχει υλοποιηθεί. Αρχικά κάνει δεξί κλικ στο στιγμιότυπο του περιφερειακού (add42_0) και επιλέγει “Browse HDL Sources...”. Στο παράθυρο που ανοίγει επιλέγει το αρχείο της γλώσσας περιγραφής υλικού που αντιστοιχεί στο περιφερειακό (add42.v) και πατάει το πλήκτρο “Open”. Τα παραπάνω φαίνονται στο Σχήμα 5.51 και στο Σχήμα 5.52.

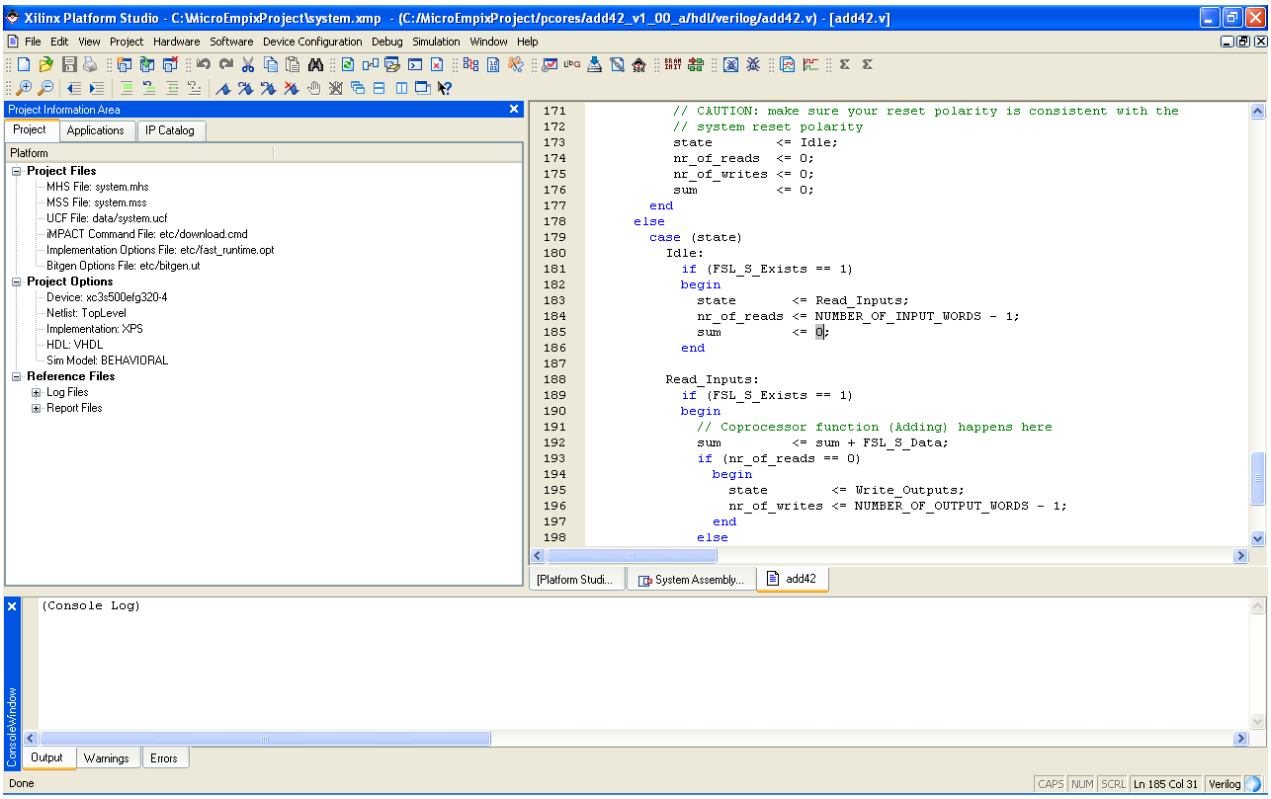


Σχήμα 5.51: Browse HDL Sources

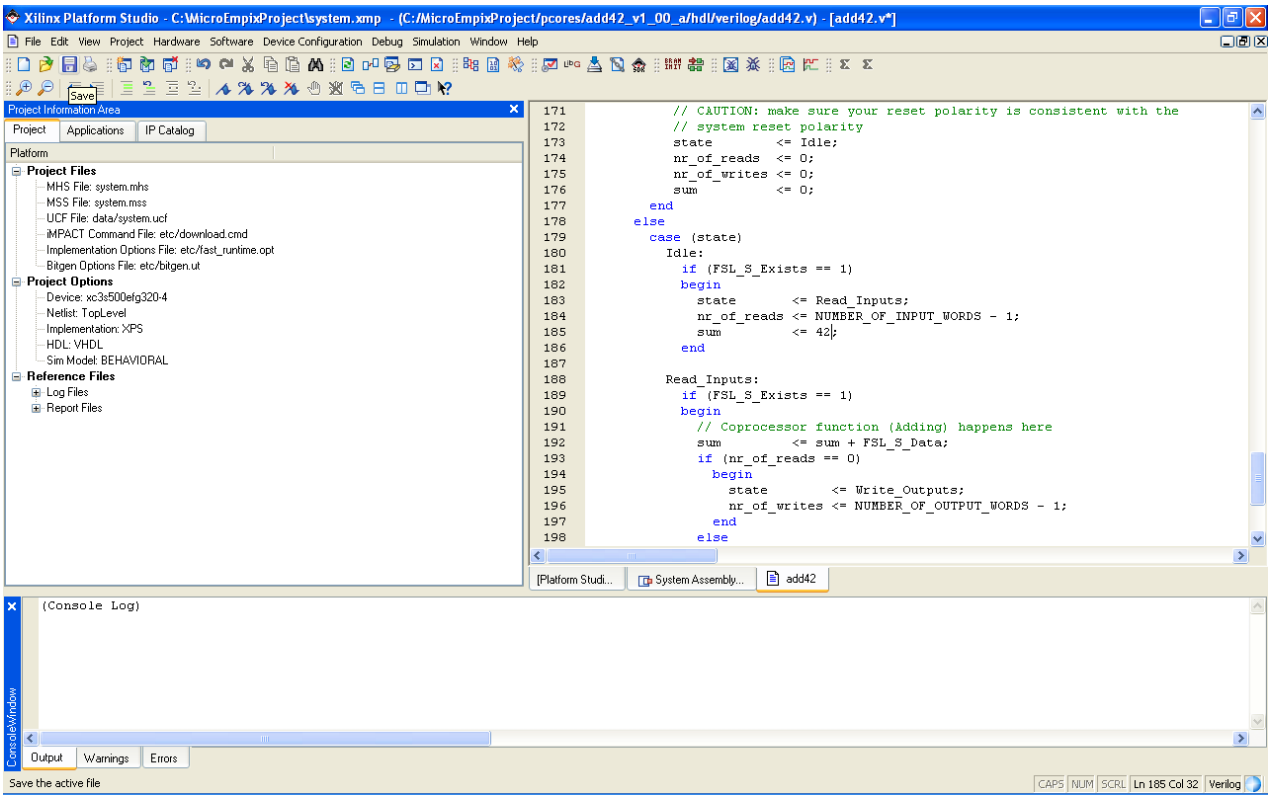


Σχήμα 5.52: Open File

Το αρχείο τότε ανοίγει στο δεξί τμήμα της οθόνης. Το example design έχει ένα συσσωρευτή που αρχικοποιείται στο 0 και σε αυτόν προστίθενται οι είσοδοι. Οι έξοδοι είναι το αποτέλεσμα της πρόσθεσης επαναλαμβανόμενο τόσες φορές όσες πρέπει να είναι οι έξοδοι. Επειδή ο χρήστης έχει επιλέξει μία είσοδο και μία έξοδο, σε αυτή την υλοποίηση του παραδείγματος σχεδίασης η έξοδος ταυτίζεται με την είσοδο. Για την επίτευξη της επιθυμητής λειτουργίας, δηλαδή η έξοδος να προκύπτει από την πρόσθεση της εισόδου με το 42, αρκεί η αρχικοποίηση του συσσωρευτή στο 42. Στο Σχήμα 5.53 και στο Σχήμα 5.54 φαίνεται η τροποποίηση που πρέπει να γίνει στον κώδικα ενώ η αποθήκευσή του γίνεται με κλικ στο εικονίδιο της δισκέτας.



Σχήμα 5.53: Example Design



Σχήμα 5.54: Final Design

5.6.5 Χρήση Περιφερειακού

Σε αυτό το σημείο ο χρήστης θα δημιουργήσει και θα εκτελέσει στο Spartan-3E Starter Board μία εφαρμογή του MicroEmpix/FPGA η οποία θα χρησιμοποιεί το στιγμιότυπο add42_0 του περιφερειακού add42, το οποίο συνδέεται στο FSL0. Αρχικά ο χρήστης δημιουργεί το αρχείο add42.c, το οποίο περιέχει τον εξής κώδικα:

```
#include "micro.h"

void P1()
{
        int value, status;

        value = 0;
        while (1)
        {
                PrintStr("==P1:");
                PrintInt(value);
                while (FSL0_WriteInt(&value, &status, 0) == 0);
                value++;
                V(2);
                P(1);
        }
}

void P2()
{
        int value, status;

        P(2);
        while (1)
        {
```

```
            PrintStr("==P2:");
            while (FSL0_ReadInt(&value, &status, 0) == 0);
            PrintInt(value);
            V(1);
            P(2);
        }
}
```

Στην εφαρμογή αυτή υπάρχουν δύο συναρτήσεις που πρόκειται να εκτελεστούν από δύο διεργασίες του MicroEmpix/FPGA. Οι δύο διεργασίες συγχρονίζονται με τη χρήση δύο σηματοφορέων. Η πρώτη διεργασία, η οποία έχει τον έλεγχο αρχικά, έχει μία μεταβλητή που αρχικοποιείται στο 0. Η διεργασία αυτή γράφει στη σειριακή θύρα την τιμή της μεταβλητής, την γράφει στο FSL0 και την αυξάνει κατά 1. Στη συνέχεια δίνει τον έλεγχο στη δεύτερη διεργασία. Η δεύτερη διεργασία διαβάζει από το FSL0 και γράφει την τιμή που διάβασε στη σειριακή θύρα. Στη συνέχεια δίνει τον έλεγχο στην πρώτη διεργασία. Αυτό επαναλαμβάνεται συνεχώς.

Για την ενσωμάτωση της εφαρμογής του χρήστη στο MicroEmpix/FPGA πρέπει να γίνουν δύο τροποποιήσεις στον κώδικα του micro.c. Η πρώτη τροποποίηση γίνεται στην αρχή του αρχείου και πρόκειται για τη δήλωση των εξωτερικών συναρτήσεων του χρήστη, δηλαδή των δύο συναρτήσεων που θα εκτελεστούν από τις δύο διεργασίες. Η προσθήκη είναι η εξής:

```
// Declaration of external user functions
extern void P1();
extern void P2();
// End of declaration of external user functions
```

Η δεύτερη προσθήκη γίνεται στο τέλος του αρχείου micro.c, μέσα στη συνάρτηση main, και πρόκειται για τη δημιουργία των δύο διεργασιών. Η προσθήκη είναι η εξής:

```
// User space for initialization and creation of processes
create(P1, 0, "P1", USER_PROCESS);
create(P2, 0, "P2", USER_PROCESS);
// End of user space for initialization and creation of processes
```

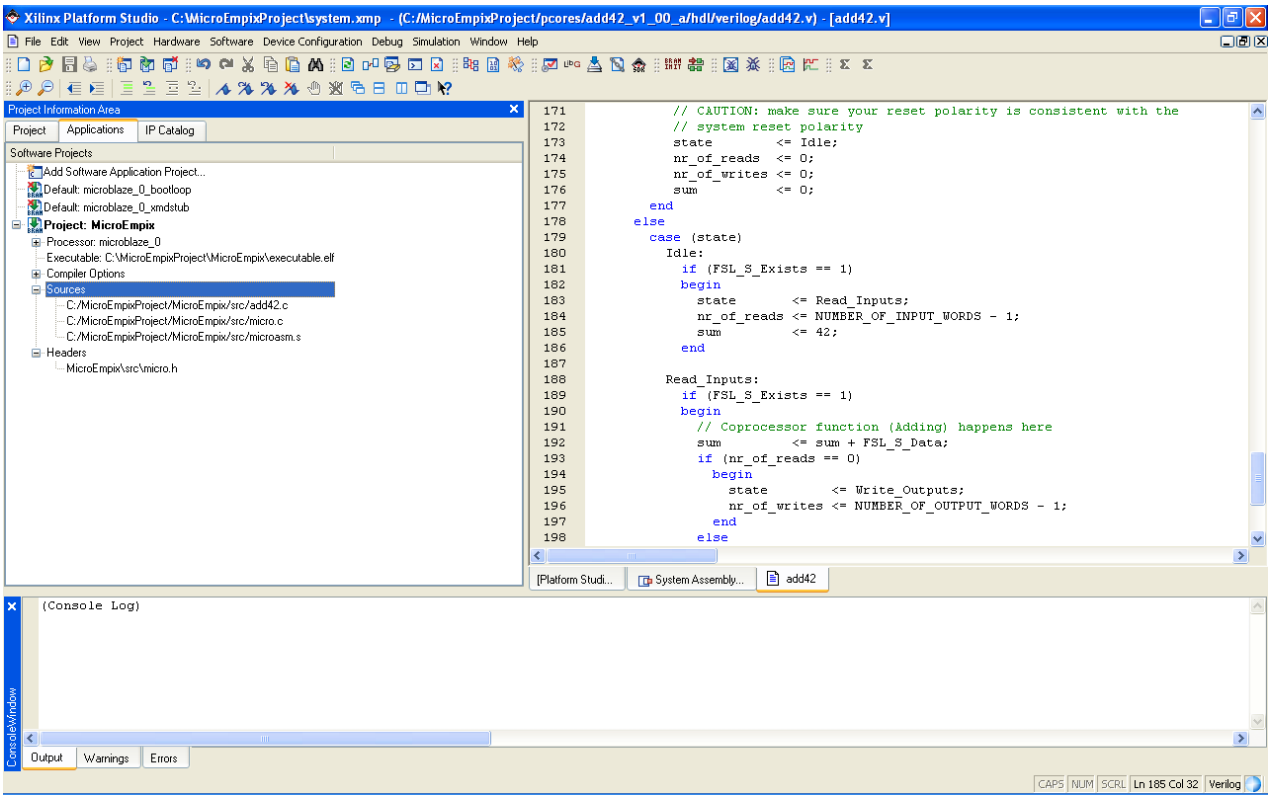
Να σημειωθεί, τέλος, ότι στην αρχή της main γίνεται η αρχικοποίηση του συστήματος με τον εξής τρόπο:

// System initialization
initialize(29452, UART_INT_MODE_OFF);

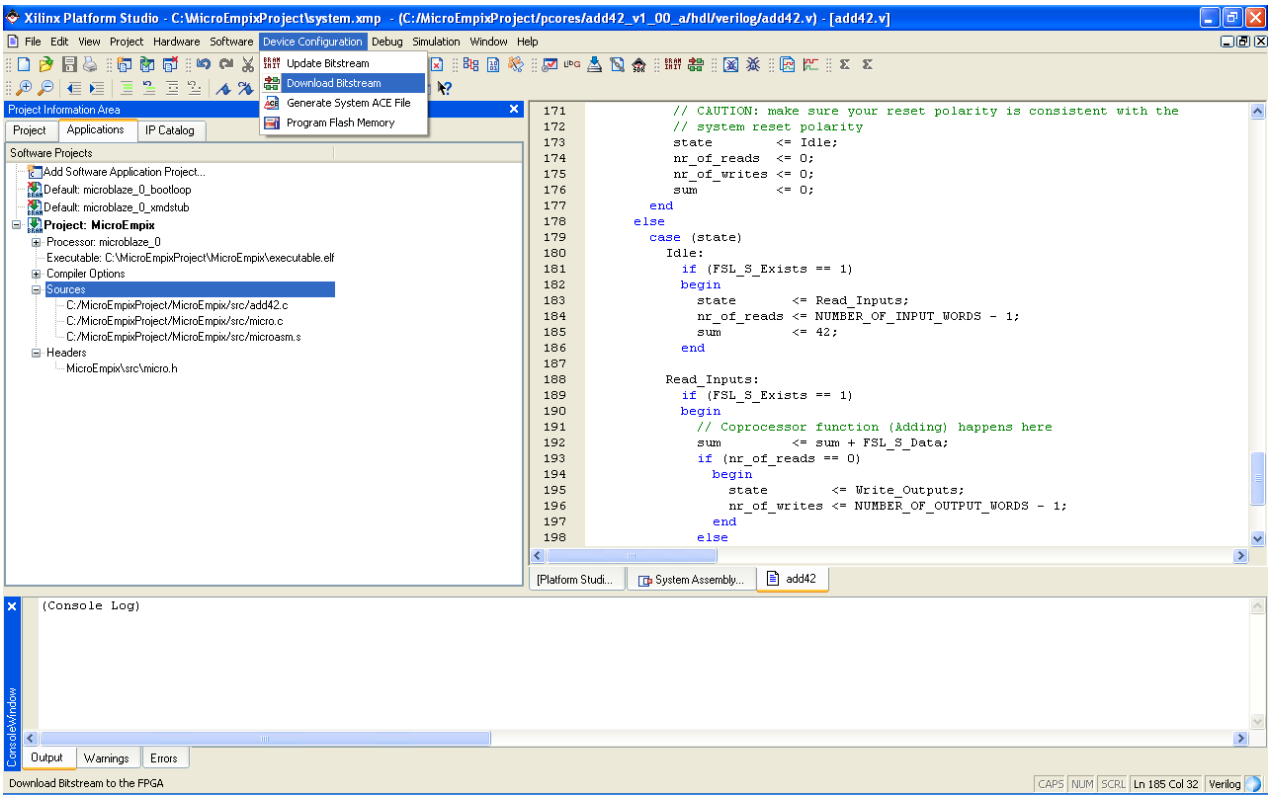
Εφόσον οι δύο διεργασίες του add42.c καλούν μόνο τις συναρτήσεις PrintInt και PrintStr όσον αφορά στις λειτουργίες της σειριακής θύρας, δε χρειάζεται ο χρήστης να αλλάξει το δεύτερο όρισμα στην παραπάνω εντολή. Αυτό συμβαίνει, γιατί οι συναρτήσεις αυτές ελέγχουν αν είναι ενεργοποιημένος ή όχι ο τρόπος λειτουργίας διακοπής της σειριακής θύρας και χρησιμοποιούν τις κατάλληλες μεθόδους. Αν ο χρήστης επιθυμεί να δοκιμάσει και τον τρόπο λειτουργίας με διακοπές της σειριακής θύρας, μπορεί να κάνει την αρχικοποίηση του συστήματος με τον εξής τρόπο:

// System initialization
initialize(29452, UART_INT_MODE_ON);

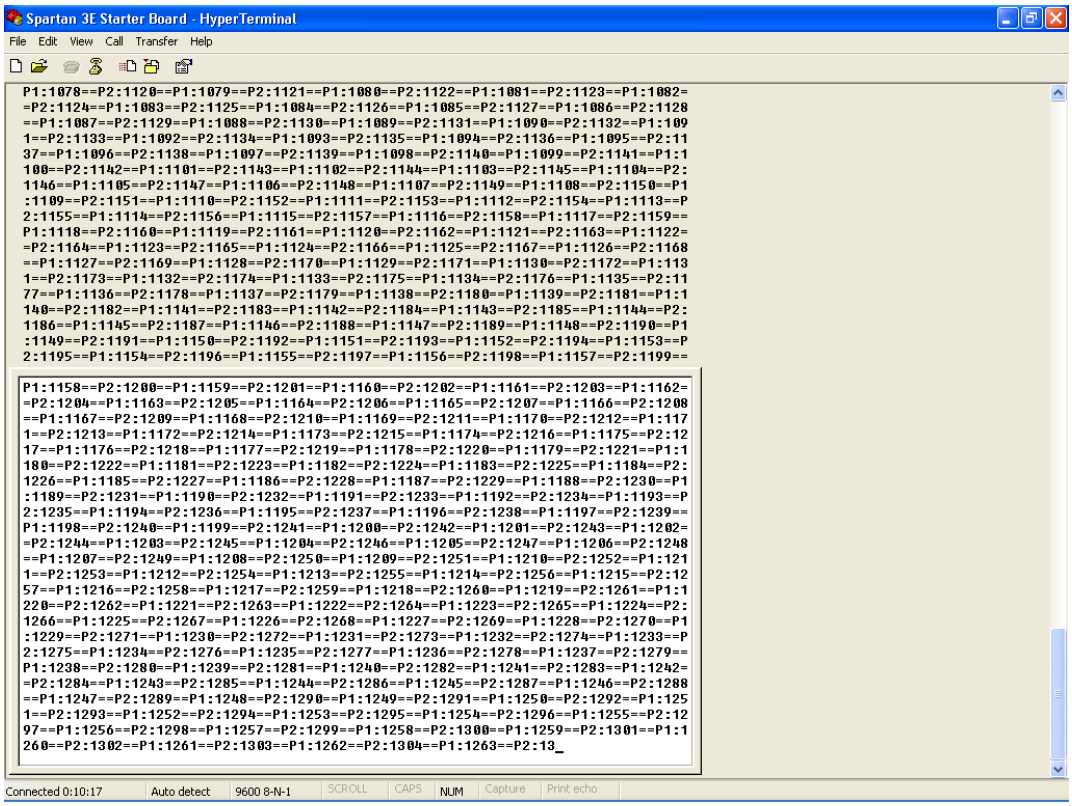
Στη συνέχεια ο χρήστης κάνει κλικ στο tab "Applications" από το πεδίο Project Information Area και φροντίζει ώστε στο πεδίο Sources του MicroEmpix να περιέχονται τα αρχεία micro.c, microasm.c και add42.c. Κάνοντας δεξί κλικ σε ένα αρχείο και επιλέγοντας "Remove" το αρχείο αφαιρείται από το project. Κάνοντας διπλό κλικ στο πεδίο Sources ο χρήστης μπορεί να προσθέσει αρχεία στο project. Τα αρχεία που απαρτίζουν τελικά το project φαίνονται στο Σχήμα 5.55. Σε αυτό το σημείο θα πρέπει ο υπολογιστής στον οποίο εκτελείται το Xilinx Platform Studio να έχει συνδεθεί μέσω της θύρας USB με το Spartan-3E Starter Board για τον προγραμματισμό της πλακέτας και το Spartan-3E Starter Board να έχει συνδεθεί μέσω της σειριακής θύρας με κάποιον υπολογιστή στον οποίο θα τρέχει το πρόγραμμα HyperTerminal. Η πλακέτα θα πρέπει να είναι, τέλος, σε λειτουργία. Από το μενού "Device Configuration" ο χρήστης επιλέγει, τελικά, το "Download Bitstream". Το παραπάνω φαίνεται στο Σχήμα 5.56. Θα εκτελεστούν αυτόματα όλα τα απαραίτητα ενδιάμεσα βήματα και τελικά η πλακέτα θα προγραμματιστεί και το MicroEmpix/FPGA θα αρχίσει να εκτελείται αμέσως. Τα αποτελέσματα της εκτέλεσης φαίνονται στο Σχήμα 5.57, το οποίο αποτελεί screenshot του HyperTerminal. Κάθε τιμή που γράφει η δεύτερη διεργασία στη σειριακή θύρα είναι κατά 42 μεγαλύτερη από την τιμή που τύπωσε η πρώτη διεργασία προηγουμένως. Το πρόγραμμα και το περιφερειακό έχουν, συνεπώς, την επιθυμητή λειτουργία.



Σχήμα 5.55: Τελικό Project



Σχήμα 5.56: Download Bitstream



Σχήμα 5.57: Αποτελέσματα Εκτέλεσης

Εκτός από τις εφαρμογές του MicroEmpix/FPGA που παρουσιάστηκαν σε αυτό το κεφάλαιο, υπήρξε και πλήθος άλλων εφαρμογών που χρησιμοποιήθηκαν για τον έλεγχο της σωστής λειτουργίας του MicroEmpix/FPGA και στις οποίες δοκιμάστηκαν επαρκώς όλες οι λειτουργίες του λειτουργικού συστήματος.

Κεφάλαιο 6

Συμπεράσματα – Μελλοντικές Επεκτάσεις

Σε αυτό το κεφάλαιο παρατίθενται τα συμπεράσματα που προκύπτουν από την εκπόνηση της παρούσας διπλωματικής εργασίας. Σκοπός αυτού του κεφαλαίου είναι η επισήμανση στον αναγνώστη των στοιχείων της εργασίας που μπορούν να αποτελέσουν αφορμή για ενδεχόμενη μελλοντική δουλειά και να τη διευκολύνουν.

Ένα πρόγραμμα ανοιχτού κώδικα, όπως είναι το αρχικό λειτουργικό σύστημα MicroEmpix αλλά και το νέο λειτουργικό σύστημα MicroEmpix/FPGA, παρέχει τη δυνατότητα στον προγραμματιστή να το τροποποιήσει και να το επεκτείνει. Το πρόγραμμα δεν είναι πλέον ένα μαύρο κουτί με το οποίο οι αλληλεπιδράσεις γίνονται με συγκεκριμένο τρόπο και η λειτουργικότητα του οποίου δεν μπορεί να αλλάξει. Ο προγραμματιστής μπορεί να αφαιρέσει στοιχεία του προγράμματος που δεν χρειάζεται για την εφαρμογή που του ζητείται και να προσθέσει επιπλέον στοιχεία που απαιτεί η εφαρμογή. Ο προγραμματιστής μπορεί με αυτό τον τρόπο να βελτιστοποιήσει το πρόγραμμα ως προς το μέγεθός του και την ταχύτητά του, καθώς και να το εκλεπτύνει και να το επεκτείνει ως προς τις λειτουργίες του. Τις δυνατότητες αυτές ενισχύει η αναγνωσιμότητα του κώδικα του προγράμματος, η οποία βασίζεται αφενός στη δομημένη υλοποίησή του και αφετέρου στην ύπαρξη αναλυτικών σχολίων που περιγράφουν τις λειτουργίες που ο κώδικας επιτελεί. Το λειτουργικό σύστημα MicroEmpix/FPGA διαθέτει τα παραπάνω στοιχεία και είναι, συνεπώς, ιδανικό για τροποποίηση και επέκταση αλλά και μεταφορά σε άλλα υπολογιστικά συστήματα.

Το MicroEmpix/FPGA προέκυψε από την τροποποίηση και την επέκταση του λειτουργικού συστήματος MicroEmpix. Ο κώδικας του αρχικού MicroEmpix είναι γραμμένος σε γλώσσα C και σε γλώσσα assembly του επεξεργαστή 8086. Σε πολλά τμήματα του κώδικα σε C του αρχικού MicroEmpix παρεμβάλλονται τμήματα κώδικα σε assembly (inline assembly), γεγονός που κάνει πιο δύσκολη την αναγνωσιμότητα του κώδικα. Στην υλοποίηση του MicroEmpix/FPGA δεν χρησιμοποιήθηκε inline assembly σε κανένα σημείο του κώδικα σε C. Διαχωρίστηκε, δηλαδή, πλήρως το τμήμα του κώδικα σε C από το τμήμα του κώδικα σε assembly του επεξεργαστή MicroBlaze, του επεξεργαστή που χρησιμοποιείται στα ενσωματωμένα συστήματα της εταιρίας Xilinx και προγραμματίζεται στα FPGAs αυτών. Η αναγνωσιμότητα γίνεται με αυτό τον τρόπο καλύτερη και η τροποποίηση και η επέκταση των λειτουργιών του κώδικα γίνονται κατ' επέκταση ευκολότερες. Η τροποποίηση του αρχικού MicroEmpix με σκοπό την υλοποίηση του MicroEmpix/FPGA αποτελείται

αφενός από αλλαγές που αφορούν την αναγνωσιμότητα, όπως είναι ο διαχωρισμός C από assembly, ο οποίος αναφέρθηκε προηγουμένως, ή η δημιουργία νέου κώδικα που επιτελεί την ίδια λειτουργία με τον παλιό αλλά είναι πιο ευανάγνωστος, και αφετέρου από αλλαγές που αφορούν τη λειτουργικότητα. Στις τελευταίες ανήκουν η εκλέπτυνση της λειτουργίας των οδηγών της σειριακής θύρας σχετικά με την καταγραφή λαθών κατά τη λήψη χαρακτήρων αλλά και η νέα υλοποίηση των σηματοφορέων, καθώς η αρχική υλοποίηση δεν εξασφάλιζε δικαιοσύνη στις λειτουργίες των σηματοφορέων. Τα υπολογιστικά συστήματα, επίσης, για τα οποία ήταν υλοποιημένο το αρχικό MicroEmpix, δηλαδή τα υπολογιστικά συστήματα που ακολουθούν την αρχιτεκτονική x86, ήταν, επίσης, διαφορετικά από το ενσωματωμένο σύστημα για το οποίο υλοποιήθηκε το MicroEmpix/FPGA, το οποίο βασίζεται στον επεξεργαστή MicroBlaze. Τέλος, προβλέφθηκαν δυνατότητες για συνεργασία και με το υπόλοιπο τμήμα του FPGA με την επέκταση της χρήσης του Fast Simplex Link. Η χρήση διαφορετικού επεξεργαστή και διαφορετικών περιφερειακών ήταν, προφανώς, η σημαντικότερη αιτία των τροποποιήσεων του αρχικού λειτουργικού συστήματος.

Η περίπτωση της μεταφοράς του MicroEmpix/FPGA σε κάποιο άλλο υπολογιστικό σύστημα αναφέρεται τόσο στην περίπτωση χρήσης του σε υπολογιστικό σύστημα ίδιας αρχιτεκτονικής, όπως για παράδειγμα σε μία πλακέτα AFX Virtex-II Pro fg456 Proto Board της εταιρείας Xilinx, όσο και στην περίπτωση χρήσης του σε υπολογιστικό σύστημα διαφορετικής αρχιτεκτονικής. Στην τελευταία περίπτωση ισχύουν ακριβώς αυτά που ίσχυαν για τη μεταφορά του MicroEmpix της αρχιτεκτονικής x86 στην αρχιτεκτονική του MicroBlaze, μεταφορά που οδήγησε στη δημιουργία του MicroEmpix/FPGA. Στην περίπτωση υπολογιστικού συστήματος ίδιας αρχιτεκτονικής με την πλακέτα Spartan-3E Starter Board, ενδέχεται να χρειαστεί αλλαγή σε ελάχιστα σημεία του κώδικα, με την προϋπόθεση, βέβαια, ότι χρησιμοποιούνται οι ίδιες συνιστώσες υλικού, δηλαδή ο επεξεργαστής MicroBlaze Processor, ο χρονιστής/μετρητής OPB Timer/Counter, ο ελεγκτής διακοπών OPB Interrupt Controller και η σειριακή θύρα OPB UART Lite. Για παράδειγμα, η υλοποίηση του MicroEmpix/FPGA ξεκίνησε στο αναπτυξιακό περιβάλλον Xilinx Platform Studio πάνω στην πλακέτα Spartan-3 Starter Board. Καθώς στη συγκεκριμένη πλακέτα υπάρχει μόνο μία σειριακή θύρα ενώ στην πλακέτα Spartan-3E Starter Board υπάρχουν δύο, υπήρξε ασυμβατότητα μεταξύ των ονομάτων που δόθηκαν αυτόματα από το αναπτυξιακό περιβάλλον στις σταθερές που χρησιμοποιούνται για τη χρήση των σειριακών θυρών. Κατά τη μεταφορά του MicroEmpix/FPGA από το Spartan-3 στο Spartan-3E έπρεπε να καθοριστεί ότι οι σταθερές έχουν την ίδια τιμή. Το σημαντικότερο είναι ότι και στις δύο πλακέτες χρησιμοποιήθηκαν οι ίδιες συνιστώσες υλικού, δηλαδή ο ίδιος επεξεργαστής, ο ίδιος ελεγκτής διακοπών, ο ίδιος χρονιστής/μετρητής και η ίδια σειριακή θύρα και, συνεπώς, η μεταφορά

έγινε κάνοντας μόνο την παραπάνω αλλαγή. Η μεταφορά, εν γένει, του MicroEmpix/FPGA σε οποιαδήποτε πλακέτα της εταιρίας Xilinx μπορεί να γίνει άμεσα με ελάχιστες τροποποιήσεις. Αρκεί, βέβαια, να χρησιμοποιηθούν, όπως αναφέρθηκε, οι ίδιες συνιστώσες υλικού. Αν χρησιμοποιηθούν διαφορετικές συνιστώσες υλικού, ισχύει για τη νέα υλοποίηση ακριβώς αυτό που ισχύει και για τη μεταφορά σε μία διαφορετική αρχιτεκτονική υπολογιστικού συστήματος.

Το MicroEmpix/FPGA έχει υλοποιηθεί ώστε να έχει τη βασική λειτουργικότητα ενός λειτουργικού συστήματος που εκτελείται στο Spartan-3E Starter Board. Χρησιμοποιεί μόνο τη μία από τις δύο διαθέσιμες σειριακές θύρες και μόνο τη μία από τις οκτώ διαθέσιμες θύρες Fast Simplex Link. Είναι προφανές ότι η επέκτασή του για τη χρήση της δεύτερης σειριακής θύρας και των υπολοίπων επτά θυρών FSL είναι πολύ απλή, αφού υπάρχει σε κάθε περίπτωση ένα πρότυπο. Το Spartan-3E Starter Board διαθέτει, ωστόσο, και άλλα περιφερειακά, όπως LEDs, push buttons και DIP switches. Με τη μελέτη των data sheets αυτών των περιφερειακών μπορεί κάποιος να επεκτείνει το MicroEmpix/FPGA, ώστε να υποστηρίζει τα εν λόγω περιφερειακά, παράγοντας και ενσωματώνοντας νέο κώδικα στο MicroEmpix/FPGA. Υπάρχει, επίσης, η δυνατότητα προγραμματισμού στο FPGA και άλλων περιφερειακών που να συνδέονται στο On-chip Peripheral Bus, στο Processor Local Bus ή στο Fast Simplex Link, δηλαδή σε όλες τις διαπροσωπείες που παρέχει ο επεξεργαστής MicroBlaze. Αυτά τα περιφερειακά μπορούν είτε να παρέχονται από τη Xilinx είτε να τα υλοποιεί ο χρήστης του αναπτυξιακού περιβάλλοντος. Και στις δύο περιπτώσεις μπορούν να υποστηριχθούν από το MicroEmpix/FPGA με τις κατάλληλες προσθήκες και τροποποιήσεις στον κώδικά του. Εκτός από τη δυνατότητα μελλοντικής υποστήριξης και άλλων περιφερειακών, υπάρχει η δυνατότητα τροποποίησης των βασικότερων λειτουργιών του MicroEmpix/FPGA. Είναι δυνατές, για παράδειγμα, η υλοποίηση ενός νέου αλγορίθμου χρονοδρομολόγησης, όπως είναι ο RMS ή ο EDF, για αντικατάσταση του Round-Robin αλλά και η υποστήριξη της δημιουργίας και του τερματισμού διεργασιών κατά τη διάρκεια της εκτέλεσης του λειτουργικού συστήματος.

Κατά τη διάρκεια της ανάπτυξης του MicroEmpix/FPGA χρησιμοποιήθηκαν οι εκδόσεις 7.1i και 8.1i του Xilinx Platform Studio. Αν και η υλοποίηση του MicroEmpix/FPGA που δοκιμάστηκε σε επίπεδο προσομοίωσης στην πρώτη έκδοση πάνω στην πλακέτα Spartan-3 Starter Board λειτουργούσε όπως αναμενόταν, δε συνέβη το ίδιο και κατά τη δοκιμή της υλοποίησης σε επίπεδο προσομοίωσης στη δεύτερη έκδοση πάνω στην ίδια πλακέτα. Αυτό οφειλόταν σε προγραμματιστικό λάθος στην υλοποίηση του MicroEmpix/FPGA σχετικό με την έναρξη της μέτρησης του χρονιστή συστήματος, δηλαδή σχετικό με τη λειτουργία του περιφερειακού OPB Timer/Counter. Η ανάπτυξη του MicroEmpix/FPGA συνεχίστηκε πάνω στην πλακέτα Spartan-3E Starter Board, η οποία ήταν

διαθέσιμη για δοκιμή του πραγματικού συστήματος. Αν και η προσομοίωση λειτουργούσε κανονικά, το MicroEmpix/FPGA δεν έτρεχε κανονικά μετά τον προγραμματισμό της πλακέτας. Αυτή τη φορά το λάθος στην υλοποίηση του MicroEmpix/FPGA ήταν η χρήση δύο εντολών του επεξεργαστή MicroBlaze που είναι προαιρετικές στην υλοποίηση του επεξεργαστή και οι οποίες είχαν παραμετροποιηθεί ώστε να μην υπάρχουν στη συγκεκριμένη υλοποίηση. Όπως δεν έτρεξε, βέβαια, η υλοποίηση σε επίπεδο πραγματικού συστήματος δεν έπρεπε να τρέξει και στην εικονική πλατφόρμα αλλά, τελικά, τα αποτελέσματα δε συμβάδιζαν. Αυτό που προκύπτει ως συμπέρασμα είναι ότι μπορεί να χρειαστεί αποσφαλμάτωση τόσο σε επίπεδο προσομοίωσης όσο και σε επίπεδο πραγματικού συστήματος (με τη χρήση του περιφερειακού Microprocessor Debug Module) για τη σωστή υλοποίηση ενός προγράμματος. Κατά τα άλλα, ωστόσο, η προσομοίωση σε επίπεδο εικονικής πλατφόρμας βοήθησε για τη διόρθωση πολλών προγραμματιστικών λαθών και κρίθηκε, συνεπώς, πολύ ικανοποιητική.

Επισημαίνεται, τέλος, ότι αν και το MicroEmpix/FPGA δοκιμάστηκε σε πλήθος μικρών εφαρμογών, καθώς και σε ένα παράδειγμα ελέγχου της κίνησης μίας μάζας, θα πρέπει να γίνουν εκτεταμένοι έλεγχοι για τη διόρθωση ενδεχόμενων σφαλμάτων, προτού χρησιμοποιηθεί σε πρακτικές εφαρμογές.

Παράρτημα Α

Κώδικας του MicroEmpix/FPGA

micro.h

```
// MicroEmpix/FPGA Constants

#define QUANTUM 2               // The time quantum given to each process
#define MAXPROC 8               // The maximum number of processes allowed
                                        // including the System Idle Process
#define MAXSEM 16               // The maximum number of semaphores available
                                        // including the 2 UART semaphores
                                        // when UART interrupt mode is on
#define STACKSIZE 512   // The size of the stack for each process
                                        // except for the System Idle Process
                                        // that is executed on the program stack
// Process type
#define SYSTEM_PROCESS 0
#define USER_PROCESS 1
// Process state
#define RUNNING 'R'
#define BLOCKED 'B'
#define READY 'A'
#define PNMLEN 16           // Process name length
#define PNREGS 37           // Process number of registers
// When UART interrupt mode is on
#define OSEM MAXSEM - 1         // Output UART semaphore
#define ISEM MAXSEM - 2 // Input UART semaphore
#define BUFLEN 1024             // Buffer length for UART input
// State of UART interrupt mode
```

```
#define UART_INT_MODE_OFF 0
#define UART_INT_MODE_ON 1

// MicroEmpix/FPGA Application Programming Interface

// Semaphore functions
void P(int s);
void V(int s);
// UART functions
// When UART interrupt mode is off
int SER_Read(char *buf, int n);
int SER_Write(char *buf, int n);
// When UART interrupt mode is on
int QSR_ReadChar(char *c);
int QSR_WriteChar(char *c);
// In both cases
void PrintStr(char *str);
void PrintInt(int n);
int GetUartError();
// Counter functions
void StartCounter(int count);
int StopCounter();
// FSL0 functions
int FSL0_ReadInt(int *value, int *status, int control);
int FSL0_WriteInt(int *value, int *status, int control);
```

micro.c

```
#include "xparameters.h"
#include "micro.h"

// For the Spartan-3E Starter Board
```

```
#define XPAR_RS232_BASEADDR XPAR_RS232_DTE_BASEADDR

// For the proper first-time execution of a process with interrupts enabled
extern void intro();

// Declaration of external user functions
extern void Initialize();
extern void P1();
extern void P2();
// End of declaration of external user functions

// Declaration of Interrupt Service Routine
void interruptServiceRoutine() __attribute__((interrupt_handler));

// UART error
int uartError;

// UART interrupt mode OFF or ON
int uartIntMode;

// Useful only if UART interrupt mode is ON
int uartWritten;

// Used only if UART interrupt mode is ON
char SER_InBuf[BUFLEN];
char *SER_InBufPtrRd, *SER_InBufPtrWr;

// Process Control Block
struct pcb
{
        char pstate;                    // Process state
        int pid;                        // Process identification number
```

```
        int pprio;                      // Process priority
        int pregs[PNREGS];          // Process registers
        char pname[PNMLEN];             // Process name
        int ptype;                      // Process type
} proctab[MAXPROC];

int current;                                    // The current process
int numproc;                                    // Total number of processes
int stacks[MAXPROC - 1][STACKSIZE];             // The stacks of the processes
int preempt;                                    // Used for rescheduling

int semaphores[MAXSEM][MAXPROC];   // The queues of the semaphores
int semavalue[MAXSEM];                          // The values of the semaphores

// Returns the distance between the Writer Pointer and the Reader Pointer
// Used for the detection of overflow
int distance()
{
        return ((int) (SER_InBufPtrWr >= SER_InBufPtrRd ?
                                        SER_InBufPtrWr - SER_InBufPtrRd :
                                        BUFLEN - (SER_InBufPtrRd - SER_InBufPtrWr)));
}

// Enable all interrupts
void enable_all_intr()
{
        int msr;

        storemsr(&msr);
        msr |= 0x2;
        loadmsr(&msr);
}
```

```
// Disable all interrupts
void disable_all_intr()
{
	int msr;

	storemsr(&msr);
	msr &= 0xFFFFFFFD;
	loadmsr(&msr);
}

// Process execution change
void reschedule()
{
	int previous, msr;

	// Disabling of interrupts due to critical section
	storemsr(&msr);
	disable_all_intr();
	// For the new process
	preempt = QUANTUM;
	// The current process becomes the previous one
	previous = current;
	// The next process that is going to be executed is chosen below
	// ------------------------------------------------------------
	do
	{
		// Check the next one
		current++;
		// If we have gone beyond the last process
		if (current == numproc)
			current = 0;
```

```
        }
        while (proctab[current].pstate == BLOCKED);
        // ------------------------------------------------------------
        if (proctab[previous].pstate == RUNNING)
                proctab[previous].pstate = READY;
        proctab[current].pstate = RUNNING;
        // Context switching
        ctxsw(proctab[previous].pregs, proctab[current].pregs);
        // Restoring previous state
        loadmsr(&msr);
}

// Create a new process
void create(void (*procaddr)(), int priority, char *name, int type)
{
        int i;

        // If no more processes are allowed
        if (numproc == MAXPROC)
                return;
        proctab[numproc].pid = numproc;                 // The id of the process
        proctab[numproc].pstate = READY; // The state of the process
        // The name of the process
        for (i = 0; (i < PNMLEN - 1) && (name[i] != '\0'); i++)
                proctab[numproc].pname[i] = name[i];
        proctab[numproc].pname[i] = '\0';
        proctab[numproc].pprio = priority;    // The priority of the process
        proctab[numproc].ptype = type;                  // The type of the process
        // Initialization of registers
        // Stack Pointer
        proctab[numproc].pregs[1] = (int) stacks[numproc];
        // Because of the rtid  r14, 0 instruction of the intro function
```

```
        proctab[numproc].pregs[14] = (int) procaddr;
        // Because of the rtsd r15, 8 instruction of the ctxsw function
        proctab[numproc].pregs[15] = ((int) intro) - 8;
        // Special purpose registers
        proctab[numproc].pregs[33] = 0;             // MSR
        proctab[numproc].pregs[34] = 0;             // EAR
        proctab[numproc].pregs[35] = 0;             // ESR
        proctab[numproc].pregs[36] = 0;             // FSR
        // Increase of the number of processes in the system
        numproc++;
}

// Initialize the system
void initialize(int count, int intMode)
{
        int i, j, value;

        // Initialize the semaphores
        for (i = 0; i < MAXSEM; i++)
                for (j = 0; j < MAXPROC; j++)
                {
                        semaphores[i][j] = 0;
                        semavalue[i] = 0;
                }
        numproc = 1;                    // Total number of processes is 1
        current = 0;                    // The current process is 0
        proctab[current].pstate = RUNNING;              // It is running
        preempt = QUANTUM;              // After QUANTUM timer interrupts
                                                // rescheduling occurs
        // Timer period is put in Load Register
        value = count - 2;
        move(XPAR_OPB_TIMER_1_BASEADDR + 4, &value);
```

```
// Load Value
value = 0x72;
move(XPAR_OPB_TIMER_1_BASEADDR, &value);
// No errors
uartError = 0;
// Clear FIFOs
value = 0x3;
move(XPAR_RS232_BASEADDR + 12, &value);
// If UART interrupt mode is OFF
if (intMode == UART_INT_MODE_OFF)
{
        uartIntMode = UART_INT_MODE_OFF;
        // Only Timer Interrupt Enable in Controller
        value = 0x1;
        move(XPAR_OPB_INTC_0_BASEADDR + 8, &value);
        // UART Interrupt disabled in UART
        value = 0x0;
        move(XPAR_RS232_BASEADDR + 12, &value);
}
// If UART interrupt mode is ON
else
{
        uartIntMode = UART_INT_MODE_ON;
        // Both Timer and UART Interrupts Enable
        value = 0x3;
        move(XPAR_OPB_INTC_0_BASEADDR + 8, &value);
        // UART Interrupt enabled in UART
        value = 0x10;
        move(XPAR_RS232_BASEADDR + 12, &value);
        uartWritten = 0;
        SER_InBufPtrRd = SER_InBuf;
        SER_InBufPtrWr = SER_InBuf;
```

```
            V(OSEM);
        }
        // For hardware interrupt generation
        value = 0x3;
        move(XPAR_OPB_INTC_0_BASEADDR + 28, &value);
        // Enable Timer
        value = 0xD2;
        move(XPAR_OPB_TIMER_1_BASEADDR, &value);
}

// Semaphore functions

// P function on semaphore s
void P(int s)
{
        int msr;

        // Disabling of interrupts due to critical section
        storemsr(&msr);
        disable_all_intr();
        // The value of the semaphore is decreased by 1
        semavalue[s]--;
        if (semavalue[s] < 0)
        {
                // The current process is blocked
                proctab[current].pstate = BLOCKED;
                // The current process enters the semaphore's waiting queue
                semaphores[s][current] = -semavalue[s];
                // Change of process execution
                reschedule();
        }
        // Restoring previous state
```

```
        loadmsr(&msr);
}

// V function on semaphore s
void V(int s)
{
        int i, msr;

        // Disabling of interrupts due to critical section
        storemsr(&msr);
        disable_all_intr();
        // The value of the semaphore is increased by 1
        semavalue[s]++;
        // If there are processes waiting on the semaphore s
        if (semavalue[s] <= 0)
        {
                // For all the processes
                for (i = 0; i < numproc; i++)
                        // If the process i is waiting on the semaphore s
                        if (semaphores[s][i] > 0)
                        {
                                // The first process becomes ready for execution
                                if (semaphores[s][i] == 1)
                                        proctab[i].pstate = READY;
                                // The queue moves forward one place
                                semaphores[s][i]--;
                        }
        }
        // Restoring previous state
        loadmsr(&msr);
}
```

```
// FSL0 functions

// Read an integer
int FSL0_ReadInt(int *value, int *status, int control)
{
        int msr;

        storemsr(&msr);
        disable_all_intr();
        if (control)
                fsl0ncget(value, status);
        else
                fsl0nget(value, status);
        loadmsr(&msr);
        *status = (((*status) >> 2) & 0x1) | (((*status) >> 3) & 0x2);
        if ((*status) & 0x1)
                return 0;
        else
                return 1;
}

// Write an integer
int FSL0_WriteInt(int *value, int *status, int control)
{
        int msr;

        storemsr(&msr);
        disable_all_intr();
        if (control)
                fsl0ncput(value, status);
        else
                fsl0nput(value, status);
```

```
	loadmsr(&msr);
	*status = (((*status) >> 2) & 0x1);
	if (*status)
		return 0;
	else
		return 1;
}

// UART functions

// When UART Interrupt Mode is OFF or in Interrupt Service Routine
// Returns the number of characters written to the buffer
// Returns when the receive FIFO is empty or n characters are read
int SER_Read(char *buf, int n)
{
	int value, status, msr, i;

	for (i = 0; i < n; i++)
	{
		// The status must not change from the moment the Status Register
		// is read until the Receive FIFO is read
		storemsr(&msr);
		disable_all_intr();
		move(&status, XPAR_RS232_BASEADDR + 8);  // Read from Status Register
		// New reception errors update the uartError
		uartError |= ((status >> 5) & 0x7);
		if ((status & 0x1) != 0)		// Receive FIFO is not empty
		{
			move(&value, XPAR_RS232_BASEADDR);	// Read from Receive FIFO
			buf[i] = value;
			loadmsr(&msr);
			continue;
```

```
                }
                else    // Receive FIFO is empty
                {
                        loadmsr(&msr);
                        break;
                }
        }
        return i;
}

// When UART Interrupt Mode is OFF or in QSR_WriteChar
// Returns the number of characters written to the Transmit FIFO
// Returns when the Transmit FIFO is full or n characters are written
int SER_Write(char *buf, int n)
{
        int value, status, msr, i;

        for (i = 0; i < n; i++)
        {
                value = buf[i];
                // The status must not change from the moment the Status Register
                // is read until the Transmit FIFO is written to
                storemsr(&msr);
                disable_all_intr();
                move(&status, XPAR_RS232_BASEADDR + 8);  // Read from Status Register
                // New reception errors update the uartError
                uartError |= ((status >> 5) & 0x7);
                if ((status & 0x8) == 0)        // Transmit FIFO is not full
                {
                        move(XPAR_RS232_BASEADDR + 4, &value);  // Write to the Transmit FIFO
                        // The Transmit FIFO has been written to
                        uartWritten = 1;
```

```
                loadmsr(&msr);
                continue;
            }
            else    // Transmit FIFO is full
            {
                loadmsr(&msr);
                break;
            }
        }
        return i;
}

// When UART Interrupt Mode is ON
int QSR_ReadChar(char *c)
{
        P(ISEM);
        *c = *SER_InBufPtrRd;
        SER_InBufPtrRd++;
        if (SER_InBufPtrRd - SER_InBuf >= BUFLEN)
                SER_InBufPtrRd = SER_InBuf;
        return 1;
}

// When UART Interrupt Mode is ON
int QSR_WriteChar(char *c)
{
        P(OSEM);
        return SER_Write(c, 1);
}

// UART interrupt mode ON or OFF
void PrintStr(char *str)
```

```
{
	while (*str != '\0')
	{
		if (uartIntMode == UART_INT_MODE_OFF)
			while (SER_Write(str, 1) == 0);
		else
			QSR_WriteChar(str);
		str++;
	}
}

// UART interrupt mode ON or OFF
void PrintInt(int n)
{
	char str[12], temp[10];
	int i, j;

	i = 0;
	j = 0;
	if (n < 0)
	{
		str[i] = '-';
		i++;
		n = -n;
	}
	do
	{
		temp[j] = '0' + (n % 10);
		j++;
	}
	while ((n = n / 10) != 0);
	while (j != 0)
```

```
        {
                j--;
                str[i] = temp[j];
                i++;
        }
        str[i] = '\0';
        PrintStr(str);
}

// UART interrupt mode ON or OFF
// Clears the uartError and returns its previous value
int GetUartError()
{
        int prevUartError, msr;

        storemsr(&msr);
        disable_all_intr();
        prevUartError = uartError;
        uartError = 0;
        loadmsr(&msr);
        return prevUartError;
}

// Counter functions

// Counter counts up starting from count
void StartCounter(int count)
{
        int value, msr;

        storemsr(&msr);
        disable_all_intr();
```

```
        value = count;
        move(XPAR_OPB_TIMER_1_BASEADDR + 20, &value);
        value = 0x20;
        move(XPAR_OPB_TIMER_1_BASEADDR + 16, &value);
        value = 0x80;
        move(XPAR_OPB_TIMER_1_BASEADDR + 16, &value);
        loadmsr(&msr);
}

// Counter stops and returns count value
int StopCounter()
{
        int value, msr;

        storemsr(&msr);
        disable_all_intr();
        value = 0x0;
        move(XPAR_OPB_TIMER_1_BASEADDR + 16, &value);
        move(&value, XPAR_OPB_TIMER_1_BASEADDR + 24);
        loadmsr(&msr);
        return value;
}

// Interrupt Service Routine
void interruptServiceRoutine()
{
        int value, status, enable;
        char c;

        // As long as interrupts may be pending
        while (1)
        {
```

```
// Read from Interrupt Status Register
move(&status, XPAR_OPB_INTC_0_BASEADDR);
// Read from Interrupt Enable Register
move(&enable, XPAR_OPB_INTC_0_BASEADDR + 8);
// No interrupts are pending
if ((status & enable) == 0)
        break;
if ((status & enable & 0x1) != 0)       // Timer interrupt has occurred
{
        // Acknowledge the interrupt
        value = 0x1D2;
        move(XPAR_OPB_TIMER_1_BASEADDR, &value);
        value = 0x1;
        move(XPAR_OPB_INTC_0_BASEADDR + 12, &value);
        // Another process may be scheduled
        preempt--;
        if (preempt == 0)
                reschedule();
}
else
        if ((status & enable & 0x2) != 0)       // UART interrupt has occurred
        {
                // Acknowledge the interrupt
                value = 0x2;
                move(XPAR_OPB_INTC_0_BASEADDR + 12, &value);
                // Get the status of the UART
                move(&status, XPAR_RS232_BASEADDR + 8);
                // New reception errors update the uartError
                uartError |= ((status >> 5) & 0x7);
                // Transmit FIFO was written to but it is empty now
                if ((uartWritten == 1) && ((status & 0x4) != 0))
                {
```

```
                        // Transmit FIFO can be written to again
                        uartWritten = 0;
                        V(OSEM);
                    }
                    // Reading from Receive FIFO
                    while (1)
                    {
                        if (SER_Read(&c, 1) == 0)   // Receive FIFO is empty
                            break;
                        else    // New character was read
                        {
                            if (distance() >= BUFLEN - 1)       // Buffer is full
                                uartError |= 0x8;       // uartError is updated
                                                        // New character is discarded
                            else    // Buffer is not full
                            {
                                // New character is written to the buffer
                                *SER_InBufPtrWr = c;
                                SER_InBufPtrWr++;
                                if (SER_InBufPtrWr - SER_InBuf >= BUFLEN)
                                    SER_InBufPtrWr = SER_InBuf;
                                // It can be read from the buffer
                                V(ISEM);
                            }
                        }
                    }
                }
            }
        }

// Program starts to run from here
// Interrupts are initially disabled
```

```
int main()
{
        // System initialization
        initialize(29452, UART_INT_MODE_OFF);
        // User space for initialization and creation of processes
        Initialize();
        create(P1, 0, "P1", USER_PROCESS);
        create(P2, 0, "P2", USER_PROCESS);
        // End of user space for initialization and creation of processes
        // Interrupts are now enabled
        enable_all_intr();
        // Turns into an idle process that loops forever
        while (1);
}
```

microasm.s

```
# void move(int *destination, int *source)
# Ο καταχωρητής R5 περιέχει τη διεύθυνση destination, στην οποία θα
# αποθηκευτεί το περιεχόμενο της διεύθυνσης source, η οποία περιέχεται
# στον καταχωρητή R6.

        .globl  move
move:
        lw              r7, r6, r0              # Φορτώνω το περιεχόμενο της διεύθυνσης
                                                        # source στον καταχωρητή R7 και
        sw              r7, r5, r0              # αποθηκεύω το περιεχόμενο του καταχωρητή
                                                        # R7 στη διεύθυνση destination.
        rtsd    r15, 8
        nop

# void ctxsw(int *prevpregs, int *currpregs)
```

```
# Η συνάρτηση ctxsw εκτελεί τη μεταγωγή περιεχομένου (context switching).
# Καλείται μόνο μέσα από τη reschedule και δεν πρέπει να διακοπεί.
# Παίρνει δύο παραμέτρους, τη διεύθυνση των καταχωρητών της προηγούμενης
# διεργασίας και τη διεύθυνση των καταχωρητών της τρέχουσας διεργασίας.
# Αυτές οι παράμετροι περιέχονται αντίστοιχα στους καταχωρητές R5 και R6,
# όπου τις έχει τοποθετήσει η reschedule πριν μας στείλει στην ctxsw.
# Ο καταχωρητής R15 περιέχει τη διεύθυνση της εντολής brlid της reschedule
# που μας έστειλε στην ctxsw.
# Ο καταχωρητής R1 περιέχει τη διεύθυνση του τελευταίου στοιχείου που η
# reschedule έχει τοποθετήσει στη στοίβα και το οποίο είναι η διεύθυνση
# επιστροφής της ίδιας της reschedule.

        .globl  ctxsw
ctxsw:
                                        # Αποθήκευση καταχωρητών προηγούμενης διεργασίας.
        swi     r1, r5, 4               # Αποθηκεύω τον Rn στη διεύθυνση R5 + 4n.

        swi     r14, r5, 56
        swi     r15, r5, 60

        swi     r19, r5, 76
        swi     r20, r5, 80
        swi     r21, r5, 84
        swi     r22, r5, 88
        swi     r23, r5, 92
        swi     r24, r5, 96
        swi     r25, r5, 100
        swi     r26, r5, 104
        swi     r27, r5, 108
        swi     r28, r5, 112
        swi     r29, r5, 116
        swi     r30, r5, 120
```

```
swi     r31, r5, 124

swi     r17, r5, 68
mfs     r7, rmsr
swi     r7, r5, 132
mfs     r7, rear
swi     r7, r5, 136
mfs     r7, resr
swi     r7, r5, 140
mfs     r7, rfsr
swi     r7, r5, 144
                                    # Φόρτωση καταχωρητών τρέχουσας διεργασίας.
lwi     r1, r6, 4           # Φορτώνω τον Rn από τη διεύθυνση R6 + 4n.

lwi     r14, r6, 56
lwi     r15, r6, 60

lwi     r19, r6, 76
lwi     r20, r6, 80
lwi     r21, r6, 84
lwi     r22, r6, 88
lwi     r23, r6, 92
lwi     r24, r6, 96
lwi     r25, r6, 100
lwi     r26, r6, 104
lwi     r27, r6, 108
lwi     r28, r6, 112
lwi     r29, r6, 116
lwi     r30, r6, 120
lwi     r31, r6, 124

lwi     r17, r6, 68
```

```
        lwi     r7, r6, 132
        mts     rmsr, r7
        lwi     r7, r6, 136
        mts     rear, r7
        lwi     r7, r6, 140
        mts     resr, r7
        lwi     r7, r6, 144
        mts     rfsr, r7

        rtsd    r15, 8
        nop

# void intro()

        .globl  intro
intro:
        rtid    r14, 0
        nop

# void storemsr(int *destination)

        .globl  storemsr
storemsr:
        mfs     r6, rmsr
        sw      r6, r5, r0
        rtsd    r15, 8
        nop

# void loadmsr(int *source)

        .globl  loadmsr
loadmsr:
```

```
        lw              r6, r5, r0
        mts             rmsr, r6
        rtsd    r15, 8
        nop

# FSL0 functions used in micro.c

# void fsl0nget(int *value, int *status)

        .globl  fsl0nget
fsl0nget:
        nget    r7, rfsl0
        mfs             r8, rmsr
        sw              r7, r5, r0
        sw              r8, r6, r0
        rtsd    r15, 8
        nop

# void fsl0ncget(int *value, int *status)

        .globl  fsl0ncget
fsl0ncget:
        ncget   r7, rfsl0
        mfs             r8, rmsr
        sw              r7, r5, r0
        sw              r8, r6, r0
        rtsd    r15, 8
        nop

# void fsl0nput(int *value, int *status)

        .globl  fsl0nput
```

```
fsl0nput:
        lw              r7, r5, r0
        nput    r7, rfsl0
        mfs             r8, rmsr
        sw              r8, r6, r0
        rtsd    r15, 8
        nop

# void fsl0ncput(int *value, int *status)

        .globl  fsl0ncput
fsl0ncput:
        lw              r7, r5, r0
        ncput   r7, rfsl0
        mfs             r8, rmsr
        sw              r8, r6, r0
        rtsd    r15, 8
        nop
```

Παράρτημα Β

Εντολές του MicroBlaze

Symbol	Description
Ra	R0 - R31, General Purpose Register, source operand a
Rb	R0 - R31, General Purpose Register, source operand b
Rd	R0 - R31, General Purpose Register, destination operand
MSR	Machine Status Register
ESR	Exception Status Register
EAR	Exception Address Register
FSR	Floating Point Unit Status Register
PC	Execute stage Program Counter
x[*y*]	Bit *y* of register *x*
x[*y*:*z*]	Bit range *y* to *z* of register *x*
$\overline{x}$	Bit inverted value of register *x*
Imm	16 bit immediate value
Imm*x*	*x* bit immediate value
FSL*x*	3 bit Fast Simplex Link (FSL) port designator where *x* is the port number
C	Carry flag, MSR[29]
Sa	Special Purpose Register, source operand
Sd	Special Purpose Register, destination operand
s(*x*)	Sign extend argument *x* to 32-bit value
*Addr	Memory contents at location Addr (data-size aligned)
&	Concatenate. E.g. "0000100 & Imm7" is the concatenation of the fixed field "0000100" and a 7 bit immediate value.
signed	Operation performed on signed integer data type
unsigned	Operation performed on unsigned integer data type
float	Operation performed on floating point data type

Σχήμα Β.1: Ονοματολογία Συνόλου Εντολών του MicroBlaze

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00000000000	Rd := Rb + Ra
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{\text{Ra}}$ + 1
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{\text{Ra}}$ + C
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00000000000	Rd := Rb + Ra

Σχήμα Β.2: Σύνολο Εντολών του MicroBlaze

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + 1
ADDKC Rd,Ra,Rb	000110	Rd	Ra	Rb	00000000000	Rd := Rb + Ra + C
RSUBKC Rd,Ra,Rb	000111	Rd	Ra	Rb	00000000000	Rd := Rb + $\overline{Ra}$ + C
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000001	Rd := Rb + $\overline{Ra}$ + 1(signed)
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00000000011	Rd := Rb + $\overline{Ra}$ + 1(unsigned)
ADDI Rd,Ra,Imm	001000	Rd	Ra	Imm		Rd := s(Imm) + Ra
RSUBI Rd,Ra,Imm	001001	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + 1
ADDIC Rd,Ra,Imm	001010	Rd	Ra	Imm		Rd := s(Imm) + Ra + C
RSUBIC Rd,Ra,Imm	001011	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + C
ADDIK Rd,Ra,Imm	001100	Rd	Ra	Imm		Rd := s(Imm) + Ra
RSUBIK Rd,Ra,Imm	001101	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + 1
ADDIKC Rd,Ra,Imm	001110	Rd	Ra	Imm		Rd := s(Imm) + Ra + C
RSUBIKC Rd,Ra,Imm	001111	Rd	Ra	Imm		Rd := s(Imm) + $\overline{Ra}$ + C
MUL Rd,Ra,Rb	010000	Rd	Ra	Rb	00000000000	Rd := Ra * Rb
BSRL Rd,Ra,Rb	010001	Rd	Ra	Rb	00000000000	Rd : = Ra >> Rb
BSRA Rd,Ra,Rb	010001	Rd	Ra	Rb	01000000000	Rd := Ra[0], (Ra >> Rb)
BSLL Rd,Ra,Rb	010001	Rd	Ra	Rb	10000000000	Rd := Ra << Rb
MULI Rd,Ra,Imm	011000	Rd	Ra	Imm		Rd := Ra * s(Imm)
BSRLI Rd,Ra,Imm	011001	Rd	Ra	00000000000 & Imm5		Rd : = Ra >> Imm5
BSRAI Rd,Ra,Imm	011001	Rd	Ra	00000010000 & Imm5		Rd := Ra[0], (Ra >> Imm5)
BSLLI Rd,Ra,Imm	011001	Rd	Ra	00000100000 & Imm5		Rd := Ra << Imm5
IDIV Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000000	Rd := Rb/Ra, signed
IDIVU Rd,Ra,Rb	010010	Rd	Ra	Rb	00000000010	Rd := Rb/Ra, unsigned
FADD Rd,Ra,Rb	010110	Rd	Ra	Rb	00000000000	Rd := Rb+Ra, float[1]
FRSUB Rd,Ra,Rb	010110	Rd	Ra	Rb	00010000000	Rd := Rb-Ra, float[1]
FMUL Rd,Ra,Rb	010110	Rd	Ra	Rb	00100000000	Rd := Rb*Ra, float[1]
FDIV Rd,Ra,Rb	010110	Rd	Ra	Rb	00110000000	Rd := Rb/Ra, float[1]
FCMP.UN Rd,Ra,Rb	010110	Rd	Ra	Rb	01000000000	Rd := 1 when (Rb = NaN or Ra = NaN, float[1]) else Rd := 0
FCMP.LT Rd,Ra,Rb	010110	Rd	Ra	Rb	01000010000	Rd := 1 when (Rb < Ra, float[1]) else Rd := 0

Σχήμα Β.3: Σύνολο Εντολών του MicroBlaze (συνέχεια)

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
FCMP.EQ Rd,Ra,Rb	010110	Rd	Ra	Rb	01000100000	Rd :– 1 when (Rb – Ra, float[1]) else Rd :– 0
FCMP.LE Rd,Ra,Rb	010110	Rd	Ra	Rb	01000110000	Rd :– 1 when (Rb <– Ra, float[1]) else Rd :– 0
FCMP.GT Rd,Ra,Rb	010110	Rd	Ra	Rb	01001000000	Rd :– 1 when (Rb > Ra, float[1]) else Rd :– 0
FCMP.NE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001010000	Rd :– 1 when (Rb !– Ra, float[1]) else Rd :– 0
FCMP.GE Rd,Ra,Rb	010110	Rd	Ra	Rb	01001100000	Rd :– 1 when (Rb >– Ra, float[1]) else Rd :– 0
GET Rd,FSLx	011011	Rd	00000	0000000000000 & FSLx		Rd :– FSLx (blocking data read) MSR[FSL] :– 1 if (FSLx_S_Control – 1)
PUT Ra,FSLx	011011	00000	Ra	1000000000000 & FSLx		FSLx :– Ra (blocking data write)
NGET Rd,FSLx	011011	Rd	00000	0100000000000 & FSLx		Rd :– FSLx (non-blocking data read) MSR[FSL] :– 1 if (FSLx_S_Control – 1) MSR[C] :– not FSLx_S_Exists
NPUT Ra,FSLx	011011	00000	Ra	1100000000000 & FSLx		FSLx :– Ra (non-blocking data write) MSR[C] :– FSLx_M_Full
CGET Rd,FSLx	011011	Rd	00000	0010000000000 & FSLx		Rd :– FSLx (blocking control read) MSR[FSL] :– 1 if (FSLx_S_Control – 0)
CPUT Ra,FSLx	011011	00000	Ra	1010000000000 & FSLx		FSLx :– Ra (blocking control write)
NCGET Rd,FSLx	011011	Rd	00000	0110000000000 & FSLx		Rd :– FSLx (non-blocking control read) MSR[FSL] :– 1 if (FSLx_S_Control – 0) MSR[C] :– not FSLx_S_Exists
NCPUT Ra,FSLx	011011	00000	Ra	1110000000000 & FSLx		FSLx :– Ra (non-blocking control write) MSR[C] :– FSLx_M_Full
OR Rd,Ra,Rb	100000	Rd	Ra	Rb	00000000000	Rd :– Ra or Rb
AND Rd,Ra,Rb	100001	Rd	Ra	Rb	00000000000	Rd :– Ra and Rb
XOR Rd,Ra,Rb	100010	Rd	Ra	Rb	00000000000	Rd :– Ra xor Rb
ANDN Rd,Ra,Rb	100011	Rd	Ra	Rb	00000000000	Rd :– Ra and $\overline{Rb}$
PCMPBF Rd,Ra,Rb	100000	Rd	Ra	Rb	10000000000	Rd :– 1 when (Rb[0:7] – Ra[0:7]) else Rd :– 2 when (Rb[8:15] – Ra[8:15]) else Rd :– 3 when (Rb[16:23] – Ra[16:23]) else Rd :– 4 when (Rb[24:31] – Ra[24:31]) else Rd :– 0
PCMPEQ Rd,Ra,Rb	100010	Rd	Ra	Rb	10000000000	Rd :– 1 when (Rd – Ra) else Rd :– 0

Σχήμα Β.4: Σύνολο Εντολών του MicroBlaze (συνέχεια)

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
PCMPNE Rd,Ra,Rb	100011	Rd	Ra	Rb	10000000000	Rd :– 1 when (Rd !– Ra) else Rd :– 0
SRA Rd,Ra	100100	Rd	Ra	0000000000000001		Rd :– Ra[0], (Ra >> 1) C :– Ra[31]
SRC Rd,Ra	100100	Rd	Ra	0000000000100001		Rd :– C, (Ra >> 1) C :– Ra[31]
SRL Rd,Ra	100100	Rd	Ra	0000000001000001		Rd :– 0, (Ra >> 1) C :– Ra[31]
SEXT8 Rd,Ra	100100	Rd	Ra	0000000001100000		Rd[0:23] :– Ra[24] Rd[24:31] :– Ra[24:31]
SEXT16 Rd,Ra	100100	Rd	Ra	0000000001100001		Rd[0:15] :– Ra[16] Rd[16:31] :– Ra[16:31]
WIC Ra,Rb	100100	00000	Ra	Rb	01101000	ICache_Tag :– Ra, ICache_Data :– Rb
WDC Ra,Rb	100100	00000	Ra	Rb	01100100	DCache_Tag :– Ra, DCache_Data :– Rb
MTS Sd,Ra	100101	00000	Ra	1100000000000 & Sd		Sd :– Ra, where Sd–001 is MSR, and Sd–111 is FSR
MFS Rd,Sa	100101	Rd	00000	1000000000000 & Sa		Rd :– Sa, where Sa–000 is PC, 001 is MSR, 011 is EAR, 101 is ESR, and 111 is FSR
MSRCLR Rd,Imm	100101	Rd	00001	00 & Imm14		Rd :– MSR MSR :– MSR and $\overline{\text{Imm14}}$
MSRSET Rd,Imm	100101	Rd	00000	00 & Imm14		Rd :– MSR MSR :– MSR or Imm14
BR Rb	100110	00000	00000	Rb	00000000000	PC :– PC + Rb
BRD Rb	100110	00000	10000	Rb	00000000000	PC :– PC + Rb
BRLD Rd,Rb	100110	Rd	10100	Rb	00000000000	PC :– PC + Rb Rd :– PC
BRA Rb	100110	00000	01000	Rb	00000000000	PC :– Rb
BRAD Rb	100110	00000	11000	Rb	00000000000	PC :– Rb
BRALD Rd,Rb	100110	Rd	11100	Rb	00000000000	PC :– Rb; Rd :– PC
BRK Rd,Rb	100110	Rd	01100	Rb	00000000000	PC :– Rb; Rd :– PC MSR[BIP] :– 1
BEQ Ra,Rb	100111	00000	Ra	Rb	00000000000	if Ra – 0: PC :– PC + Rb
BNE Ra,Rb	100111	00001	Ra	Rb	00000000000	if Ra /– 0: PC :– PC + Rb
BLT Ra,Rb	100111	00010	Ra	Rb	00000000000	if Ra < 0: PC :– PC + Rb
BLE Ra,Rb	100111	00011	Ra	Rb	00000000000	if Ra <– 0: PC :– PC + Rb
BGT Ra,Rb	100111	00100	Ra	Rb	00000000000	if Ra > 0: PC :– PC + Rb

Σχήμα Β.5: Σύνολο Εντολών του MicroBlaze (συνέχεια)

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

Σχήμα Β.6: Σύνολο Εντολών του MicroBlaze (συνέχεια)

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
BGEI Ra,Imm	101111	00101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
BEQID Ra,Imm	101111	10000	Ra	Imm		if Ra = 0: PC := PC + s(Imm)
BNEID Ra,Imm	101111	10001	Ra	Imm		if Ra /= 0: PC := PC + s(Imm)
BLTID Ra,Imm	101111	10010	Ra	Imm		if Ra < 0: PC := PC + s(Imm)
BLEID Ra,Imm	101111	10011	Ra	Imm		if Ra <= 0: PC := PC + s(Imm)
BGTID Ra,Imm	101111	10100	Ra	Imm		if Ra > 0: PC := PC + s(Imm)
BGEID Ra,Imm	101111	10101	Ra	Imm		if Ra >= 0: PC := PC + s(Imm)
LBU Rd,Ra,Rb	110000	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd[0:23] := 0, Rd[24:31] := *Addr
LHU Rd,Ra,Rb	110001	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd[0:15] := 0, Rd[16:31] := *Addr
LW Rd,Ra,Rb	110010	Rd	Ra	Rb	00000000000	Addr := Ra + Rb Rd := *Addr
SB Rd,Ra,Rb	110100	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr := Rd[24:31]
SH Rd,Ra,Rb	110101	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr := Rd[16:31]
SW Rd,Ra,Rb	110110	Rd	Ra	Rb	00000000000	Addr := Ra + Rb *Addr := Rd
LBUI Rd,Ra,Imm	111000	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:23] := 0, Rd[24:31] := *Addr
LHUI Rd,Ra,Imm	111001	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd[0:15] := 0, Rd[16:31] := *Addr
LWI Rd,Ra,Imm	111010	Rd	Ra	Imm		Addr := Ra + s(Imm) Rd := *Addr
SBI Rd,Ra,Imm	111100	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr := Rd[24:31]
SHI Rd,Ra,Imm	111101	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr := Rd[16:31]
SWI Rd,Ra,Imm	111110	Rd	Ra	Imm		Addr := Ra + s(Imm) *Addr := Rd

1. Due to the many different corner cases involved in floating point arithmetic, only the normal behavior is described.

Σχήμα Β.7: Σύνολο Εντολών του MicroBlaze (συνέχεια)

Βιβλιογραφία

[1] A. S. Tanenbaum and A. S. Woodhull, "Operating Systems: Design and Implementation", Prentice Hall, 2006.

[2] Γ. Κ. Παπακωνσταντίνου, Ν. Α. Μπιλάλης και Π. Δ. Τσανάκας, "Λειτουργικά Συστήματα Μέρος Ι: Αρχές Λειτουργίας", Συμμετρία, 1999.

[3] Γ. Παπακωνσταντίνου, Χ. Δελλαρόκας και Π. Τσανάκας, "Το Λειτουργικό Σύστημα EMPIX", Σ. Αθανασόπουλος – Σ. Παπαδάμης & Σία Ε.Ε., 1993.

[4] Γ. Παπακωνσταντίνου και Ι. Παναγόπουλος, "Το Λειτουργικό Σύστημα MICRO-EMPIX", 2004. Διαθέσιμο από: http://www.cslab.ece.ntua.gr/courses/compsyslab/files/Micro/MICRO%20NOTES%201.0.pdf.

[5] F. Vahid and T. Givargis, "Embedded System Design: A Unified Hardware/Software Introduction", Wiley, 2001.

[6] Xilinx Inc., "Embedded System Tools Reference Manual", 2005. Available from: http://www.xilinx.com/ise/embedded/est_rm.pdf.

[7] Xilinx Inc., "MicroBlaze Processor Reference Guide", 2006. Available from: http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

[8] Xilinx Inc., "OPB Interrupt Controller", 2005. Available from: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_intc.pdf.

[9] Xilinx Inc., "OPB Timer/Counter", 2005. Available from: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_timer.pdf.

[10] Xilinx Inc., "OPB UART Lite", 2005. Available from: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_uartlite.pdf.