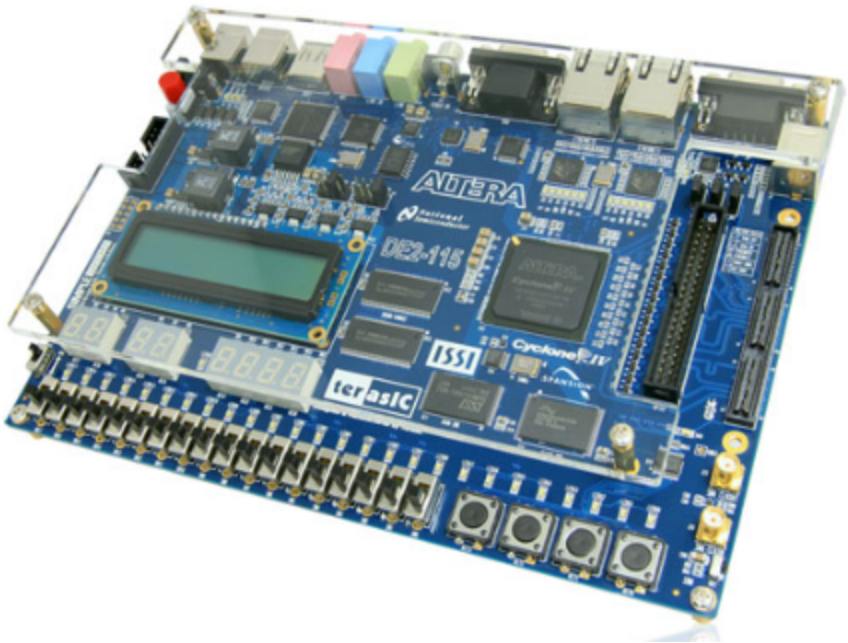
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Ανάπτυξη εργαστηριακού εκπαιδευτικού υλικού Ενσωματωμένων Συστημάτων με χρήση του επεξεργαστή Nios II

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ρίζος Β. Χρήστος



Επιβλέπων : Μ. Καλλίγερος, Επίκουρος Καθηγητής

Σάμος, Οκτώβριος 2019

ΑΝΆΠΤΥΞΗ ΕΡΓΑΣΤΗΡΙΑΚΟΎ ΕΚΠΑΙΔΕΥΤΙΚΟΎ ΥΛΙΚΟΎ ΕΝΣΩΜΑΤΩΜΈΝΩΝ ΣΥΣΤΗΜΆΤΩΝ ΜΕ ΧΡΉΣΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΉ NIOS II

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ρίζος Β. Χρήστος

Η Διπλωματική Εργασία παρουσιάστηκε ενώπιον του Διδακτικού Προσωπικού του Πανεπιστημίου Αιγαίου σε μερική εκπλήρωση των απαιτήσεων για την απόκτηση του μεταπτυχιακού διπλώματος του προγράμματος Μεταπτυχιακών Σπουδών με τίτλο Τεχνολογίες και Διοίκηση Πληροφοριακών και Επικοινωνιακών Συστημάτων, του τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων.

Η τριμελής επιτροπή διδασκόντων που επικυρώνει τη διπλωματική εργασία:

Μ. Καλλίγερος, Επίκουρος Καθηγητής (Επιβλέπων)
Χ. Γκουμόπουλος, Επίκουρος Καθηγητής
Γ. Κορμέτζας, Καθηγητής

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του Μεταπτυχιακού Προγράμματος Σπουδών του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων του Πανεπιστημίου Αιγαίου <<Τεχνολογίες και Διοίκηση Πληροφοριακών και Επικοινωνιακών Συστημάτων>>, υπό την επίβλεψη του Επίκουρου Καθηγητή κ. Εμμανουήλ Καλλίγερου.

Αρχικά θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου για τη στήριξη καθ' όλη τη διάρκεια των μεταπτυχιακών σπουδών μου, καθώς και για την δυνατότητα που μου έδωσε να πραγματοποιήσω τη διπλωματική μου εργασία.

Το μεγαλύτερο όμως ευχαριστώ το χρωστάω κυρίως στους γονείς μου Αριστέα και Βασίλειο, που μου έδωσαν τη δυνατότητα να σπουδάσω και να εξελιχθώ, που πάντα με στηρίζουν στις επιλογές που παίρνω, και που είναι δίπλα μου σε κάθε μου βήμα, καθώς επίσης και στους φίλους μου που ήταν εκεί όταν χρειάστηκε να με βοηθήσουν να μη χάσω τους στόχους μου.

Περίληψη

Οι έξυπνες συσκευές, ή αλλιώς ενσωματωμένα συστήματα, είναι πανταχού παρούσες και ο σχεδιασμός αυτών των ενσωματωμένων συστημάτων απαιτεί μια πολυεπιστημονική προσέγγιση. Είναι σημαντικό λοιπόν, οι φοιτητές της επιστήμης των υπολογιστών να μάθουν για αυτές τις διαφορετικές πτυχές του σχεδιασμού των ενσωματωμένων συστημάτων. Η διδασκαλία μόνο των θεωρητικών εννοιών στους φοιτητές όμως δεν αρκεί. Πρακτικά, εργαστηριακά μαθήματα είναι απαραίτητα για να αποκτηθούν όλες οι απαραίτητες δεξιότητες.

Στην παρούσα διπλωματική εργασία παρουσιάζεται μία προσέγγιση αυτών των εργαστηριακών μαθημάτων, στόχος της οποίας είναι να δημιουργήσει σχετικά μικρά προβλήματα, καθένα από τα οποία να επικεντρώνεται σε μία ενιαία πτυχή σχεδιασμού ενός ενσωματωμένου συστήματος. Η παρούσα διπλωματική εργασία μπορεί να χαρακτηριστεί και ως μία επισκόπηση του τρόπου σχεδίασης των ενσωματωμένων συστημάτων, με κύρια εστίαση στις βασικές έννοιες αυτών, προετοιμάζοντας τον φοιτητή μέσα από τέσσερις εργαστηριακές ασκήσεις, για μια πιο λεπτομερή παρακολούθηση των μαθημάτων της αντίστοιχης θεωρίας.

Η εργασία χωρίζεται σε δύο μέρη: το θεωρητικό και το εργαστηριακό-πρακτικό. Το θεωρητικό μέρος προκύπτει από τη μελέτη της βιβλιογραφίας και των ηλεκτρονικών πηγών, και στοχεύει στην απόκτηση του απαραίτητου θεωρητικού υπόβαθρου των προγραμματιζόμενων ψηφιακών κυκλωμάτων, καθώς επίσης και του λογισμικού Quartus Prime, που αποτελεί το βασικό εργαλείο σχεδίασης και σύνθεσης ψηφιακών κυκλωμάτων, όταν αυτά πρόκειται να τοποθετηθούν σε προγραμματιζόμενα ολοκληρωμένα (FPGA) της εταιρίας Intel.

Στο εργαστηριακό-πρακτικό μέρος παρουσιάζεται μια σειρά εφαρμογών με την αναπτυξιακή πλατφόρμα DE2-115 της εταιρίας Terasic, το οποίο βασίζεται στο προγραμματιζόμενο ολοκληρωμένο (FPGA) Cyclone IV της Intel, με αναφορά στις κυριότερες συσκευές που περιλαμβάνει. Όλοι οι σχεδιασμοί υλοποιήθηκαν χρησιμοποιώντας το λογισμικό Quartus Prime v18.1 της Intel.

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Περιεχόμενα

1. Εισαγωγικό Κεφάλαιο

Η παρούσα μεταπτυχιακή εργασία ασχολείται με τη σχεδίαση τεσσάρων, απλών και πιο σύνθετων, προγραμματιζόμενων ψηφιακών συστημάτων, με τη χρήση του εργαλείου Platform Designer στο περιβάλλον του Quartus Prime της Intel, καθώς επίσης και με την ανάπτυξη του λογισμικού που υποστηρίζει τις δυνατότητες των κυκλωμάτων αυτών.

Πρέπει να τονιστεί ότι σκοπός της κάθε άσκησης δεν είναι να καλύψει πλήρως την χρήση του περιβάλλοντος της Intel, καθώς κάτι τέτοιο δεν θα ήταν εφικτό μόνο μέσα από αυτές τις τέσσερις εργαστηριακές ασκήσεις. Αυτό που θέλει να πετύχει συγκεκριμένα αυτή η σειρά ασκήσεων, είναι αρχικά η εξοικείωση με το εργαλείο Platform Designer, και κατ' επέκταση, με τη χρήση της εκπαιδευτικής πλακέτας FPGA DE2-115 και του επεξεργαστή της NIOS II.

1.1 Παιδαγωγική κατεύθυνση της διπλωματικής εργασίας

Σε οποιοδήποτε είδος διδακτικού και μαθησιακού περιβάλλοντος, ο κύριος στόχος θα πρέπει να είναι η παρουσίαση της γνώσης με τέτοιο τρόπο, ώστε να έχει το καλύτερο μαθησιακό αποτέλεσμα σε αυτόν που την λαμβάνει.

Διαφοροποιώντας τον τρόπο με τον οποίο παρουσιάζεται και διδάσκεται η γνώση είναι δυνατό να έχουμε πολύ διαφορετικά αποτελέσματα σε σχέση με αυτά που θα καταλάβει και θα διδαχθεί ο φοιτητής. Κατά την ανάγνωση της διπλωματικής, η χρήση των εργαλείων που παρουσιάζονται, σε συνδυασμό με την εμπειρία τρίτων, με γνώση του αντικειμένου, μπορεί να βελτιώσει σημαντικά τον βαθμό κατανόησή της.

1.2 Ενσωματωμένα Συστήματα

Οι περισσότεροι άνθρωποι είναι εξοικειωμένοι με τον επιτραπέζιο υπολογιστή. Ο επιτραπέζιος ή, όπως καλείται στις μέρες μας, προσωπικός υπολογιστής, είναι μια πολύ ευέλικτη υπολογιστική μηχανή που μπορεί να χρησιμοποιηθεί για πολλές εφαρμογές, όπως για να παίξει κάποιος παιχνίδια, να ελέγξει την ηλεκτρονική του αλληλογραφία κ.α. Ένα ενσωματωμένο σύστημα μπορεί επίσης να οριστεί ως ένα σύστημα υπολογιστή, το οποίο όμως έχει σχεδιαστεί για να εκτελεί μία πολύ συγκεκριμένη εργασία. Αυτό βρίσκεται σε πλήρη αντίθεση με τη χρήση ενός προσωπικού ηλεκτρονικού υπολογιστή που έχει σχεδιαστεί με σκοπό η χρήση του να είναι πολύ πιο γενική.

Ως αποτέλεσμα, το ενσωματωμένο σύστημα μπορεί να κατασκευαστεί για να εκτελεί τις επιμέρους εργασίες με πολύ πιο αποτελεσματικό τρόπο, τόσο σε σχέση με το μέγεθός του, όσο και με την κατανάλωση ενέργειας και την απόδοσή του. Τα ενσωματωμένα συστήματα είναι στην πραγματικότητα πολύ πιο κοινά από τον επιτραπέζιο υπολογιστή, αν και δεν είναι πάντα τόσο εύκολο να εντοπιστούν. Περίπου το 99% όλων των μικροεπεξεργαστών που κατασκευάζονται χρησιμοποιούνται σε ενσωματωμένα συστήματα.

1.2.1 Τα Ενσωματωμένα Συστήματα στον κόσμο

Μπορεί κάποιος να βρει ενσωματωμένα συστήματα σε σχεδόν οποιοδήποτε είδος καταναλωτικών προϊόντων με ηλεκτρονικές λειτουργίες, όπως αυτοκίνητα, πλυντήρια, καφετιέρες, παιχνίδια, τηλεοράσεις, Blu-Ray players κ.α. Τα ενσωματωμένα συστήματα είναι επίσης πολύ κοινά και σημαντικά στη βιομηχανία και χρησιμοποιούνται συχνά για τον έλεγχο μηχανικών συστημάτων, πολλά από τα οποία έχουν αυξημένες απαιτήσεις σε χρόνο απόκρισης και ταχύτητα. Τα συστήματα αυτά συχνά αναφέρονται ως συστήματα πραγματικού χρόνου.

Πολλά από αυτά τα συστήματα έχουν απαιτήσεις που δεν θα ήταν δυνατόν να επιτευχθούν με ένα συμβατικό επιτραπέζιο υπολογιστή. Πιθανές αιτίες μπορεί να είναι η ανεπάρκεια επεξεργαστικής ισχύς, ή ο μη εξειδικευμένος χαρακτήρας των υπολογιστών αυτών. Τα ενσωματωμένα συστήματα αποτελούνται συνήθως από έναν συνδυασμό υλικού (π.χ. μνήμη, περιφερειακά, μικροελεγκτές) και λογισμικού (κάποιο είδος προγράμματος που υλοποιεί τη λειτουργία συστήματος). Αυτός ο συνδυασμός παρέχει ευελιξία και ευρείες δυνατότητες βελτιστοποίησης και προσαρμογής. Για να μπορεί ένα ενσωματωμένο σύστημα να εκτελεί λογισμικό απαιτείται και ένας μικροεπεξεργαστής ή μικροελεγκτής για την εκτέλεση των εντολών.

1.2.2 Τα Ενσωματωμένα Συστήματα στην εκπαίδευση

Υπάρχουν αρκετοί άνθρωποι που ερευνούν το πεδίο του τρόπου διδασκαλίας των ενσωματωμένων συστημάτων και των μικροεπεξεργαστών. Ένας από τους κοινούς τους στόχους είναι να βρουν ένα αποτελεσματικό τρόπο για να παρέχουν στους φοιτητές επαρκή γνώση επί του θέματος. Αρκετοί προτείνουν να αλλάξει το γενικό πρόγραμμα προπτυχιακών σπουδών των σχολών μηχανικών υπολογιστών, έτσι ώστε να περιλαμβάνουν στα πεδία τους μία εισαγωγή στα ενσωματωμένα συστήματα, τα οποία θεωρούνται ήδη αρκετά εξειδικευμένα για ένα μεταπτυχιακό φοιτητή.

Αναφέρεται επίσης ότι για να μπορέσουν οι φοιτητές να αναπτύξουν τις δεξιότητές τους στο κόσμο των ενσωματωμένων συστημάτων, δεν αρκεί να γίνονται μόνο μεμονωμένες παρουσιάσεις (π.χ. σεμινάρια), ή μονομερή μαθήματα (μόνο μαθήματα θεωρίας ή επαφή με εργαστηριακό εξοπλισμό χωρίς την απαραίτητη γνωσιακή υποστήριξη), αλλά χρειάζεται ένα πιο εκτεταμένο και εξειδικευμένο πρόγραμμα σπουδών για την κατάλληλη εκπαίδευση των φοιτητών στον σχεδιασμό των ενσωματωμένων συστημάτων. Για τον λόγο αυτό τα ενσωματωμένα συστήματα θεωρούνται ένα πολύ εκτεταμένο και μεταβαλλόμενο πεδίο διδασκαλίας.

Κατά τη διαδικασία διδασκαλίας των μαθημάτων που παρέχονται στο τομέα της επιστήμης των υπολογιστών, σημαντικές πληροφορίες έχουν συγκεντρωθεί σχετικά με την επίτευξη των εκάστοτε μαθησιακών στόχων. Τα μαθήματα στα οποία οι φοιτητές έδειξαν μεγαλύτερη ανταπόκριση εμπεριείχαν και πρακτικό μέρος, αποδεικνύοντας ότι μια θεωρητική προσέγγιση δεν αρκεί. Οι πρακτικές συνεδρίες είναι απαραίτητες για να μπορέσει ο φοιτητής να κατανοήσει πραγματικά τη θεωρία.

Παρόλο που το μάθημα των ενσωματωμένων συστημάτων μπορεί να επικεντρώνεται σε έναν σύνθετο σχεδιασμό, τα πρακτικά παραδείγματα θα πρέπει να διατηρούνται αρκετά μικρά και απλά για να μπορέσουν να εξηγήσουν τις βασικές έννοιες χωρίς να χαθεί η προσοχή και το ενδιαφέρων των φοιτητών λόγω της πολυπλοκότητας των προβλημάτων.

Γενικά είναι καλο να γίνεται εστίαση σε μια μικρή πτυχή του σχεδιασμού, με επιπλέον επεξήγηση του τρόπου με τον οποίο μικρά κομμάτια σχεδιασμού μπορούν να χρησιμοποιηθούν με πιο περίπλοκο τρόπο. Βασικοί παράγοντες για την απόκτηση αυτού του καλού μαθησιακού περιβάλλοντος είναι η ενθάρρυνση και το κίνητρο από τον καθηγητή, αλλά και η αλληλεπίδραση και τα παραδείγματα μέσω του πραγματικού κόσμου.

Ο επεξεργαστής Nios II, που χρησιμοποιείται στο FPGA της πλακέτας DE2-115 της Terasic, σε συνδυασμό με τον αναδιαμορφώσιμο χαρακτήρα του FPGA γενικά, παρέχει μεγάλη ευελιξία στο να επιδείξει πολλές διαφορετικές πτυχές του τομέα των ενσωματωμένων συστημάτων. Στην συγκεκριμένη διπλωματική εργασία γίνεται μία προσπάθεια παρουσίασης μερικών εκ των πτυχών αυτών, καθώς οι ασκήσεις που παρουσιάζονται χρησιμοποιούν πρακτικές προσεγγίσεις για την εκμάθηση του φοιτητή, παρέχοντάς του όπου είναι απαραίτητο την αντίστοιχη θεωρία, αλληλεπιδρώντας παράλληλα και με την εκπαιδευτική πλακέτα.

1.3 Εισαγωγή στις Συστοιχίες Πυλών Προγραμματιζόμενες από το Χρήστη (FPGA)

1.3.1 Συσκευές προγραμματίσιμης λογικής (PLD)

Μία συσκευή προγραμματίσιμης λογικής (Programmable Logic Device - PLD) είναι ένα ολοκληρωμένο κύκλωμα αποτελούμενο από διατάξεις πυλών AND και OR που υλοποιούν συναρτήσεις αθροίσματος γινομένων. Σε αντίθεση με τα ολοκληρωμένα κυκλώματα (IC) που επίσης αποτελούνται από λογικές πύλες, που έχουν όμως και μια σταθερή λειτουργία, ένα PLD έχει μη προσδιορισμένη λειτουργία κατά το χρόνο της κατασκευής του. Πριν το PLD χρησιμοποιηθεί σε ένα κύκλωμα, πρέπει να προγραμματιστεί (διαμορφωθεί) χρησιμοποιώντας κάποια εξειδικευμένη συσκευή.

Υπάρχουν τρεις απλοί τύποι συνδυαστικών PLD οι οποίοι διαφέρουν κυρίως στη διάταξη των πυλών AND/OR:

1. *PROM (Programmable ROM)*: Προγραμματίσιμη μνήμη ανάγνωσης-μόνο με σταθερή διάταξη πυλών AND που είναι υλοποιημένη ως αποκωδικοποιητής, και μια προγραμματίσιμη διάταξη πυλών OR που υλοποιούν τις συναρτήσεις Boole σε μορφή αθροίσματος ελαχιστόρων.

2. *PAL (Programmable Array Logic)*: Η PAL έχει μια προγραμματίσιμη διάταξη πυλών AND, οι οποίες παράγουν γινόμενα δυαδικών μεταβλητών, τα οποία αθροίζονται λογικά σε κάθε πύλη OR, οι οποίες βρίσκονται σε σταθερή διάταξη.

3. *PLA (Programmable Logic Array)*: Ο πιο ευέλικτος τύπος PLD είναι οι PLA, στις οποίες τόσο οι διατάξεις AND, όσο και οι διατάξεις OR, είναι προγραμματίσιμες. Τα γινόμενα που παράγονται από τη διάταξη των AND, εφαρμόζονται άμεσα σε οποιαδήποτε OR, καταλλήγοντας έτσι στη δυνατότητα υλοποίησης πολλών διαφορετικών αθροισμάτων γινομένων.

Επιπλέον των απλών αυτών τύπων συνδυαστικών PLD, υπάρχουν και κάποιες βελτιωμένες εκδοχές αυτών:

1. *GAL (Generic Array Logic)*: Μία βελτίωση των PAL είναι η γενική λογική διάταξη συστοιχιών (GAL). Αυτή η συσκευή έχει τις ίδιες λογικές ιδιότητες με μία PAL αλλά μπορεί να διαγραφεί και να επαναπρογραμματιστεί. Μία GAL είναι πολύ χρήσιμη στο στάδιο πρωτοτυποποίησης ενός σχεδιασμού, όταν οποιαδήποτε σφάλματα στη λογική μπορούν να διορθωθούν με επαναπρογραμματισμό. Οι GAL μπορούν να επαναπρογραμματιστούν χρησιμοποιώντας τις ίδιες συσκευές με τις PAL.

2. *CPLD (Complex Programmable Logic Array)*: Οι PAL και GAL είναι διαθέσιμες μόνο σε μικρά μεγέθη, επιτρέποντας την υλοποίηση σχεδιασμού με μερικές εκατοντάδες λογικές πύλες. Για μεγαλύτερα λογικά κυκλώματα, μπορούν να χρησιμοποιηθούν πιο σύνθετα PLD ή αλλιώς CPLD. Αυτά περιέχουν το ισοδύναμο πολλών PAL που συνδέονται με προγραμματιζόμενες διασυνδέσεις, όλα σε ένα ολοκληρωμένο κύκλωμα. Τα CPLD μπορούν να χρησιμοποιηθούν για την υλοποίηση κυκλωμάτων με χιλιάδες ή ακόμα και εκατοντάδες χιλιάδες λογικές πύλες.

3. *FPGA (Field Programmable Gate Arrays)*: Ενώ οι PAL μετεξελίχθηκαν σε GAL και CPLD, ένα ξεχωριστό μονοπάτι ανάπτυξης προγραμματιζόμενων συσκευών ξεκίνησε. Έτσι εμφανίστηκε ένας νέος τύπος συσκευών, ο οποίος βασίστηκε στην τεχνολογία συστοιχίας πυλών και ονομάστηκε Συστοιχίες Πυλών Προγραμματιζόμενων από το Χρήστη (Field Programmable Gate Arrays - FPGA).

1.3.2 Συστοιχίες Πυλών Προγραμματιζόμενες από τον Χρήστη

Τα FPGA είναι κυκλώματα VLSI, τα οποία μπορούν να προγραμματιστούν στο εργαστήριο του χρήστη, κάτι το οποίο “πέρασε” και στον ίδιο τον όρο των συσκευών (“Συστοιχίες Πυλών *Προγραμματιζόμενες από το Χρήστη*”). Ένα τυπικό FPGA αποτελείται από διατάξεις εκατοντάδων ή χιλιάδων λογικών πυλών, περιβαλλόμενων από επαναπρογραμματίσιμα μπλοκ εισόδου/εξόδου, διασυνδεδεμένων μεταξύ τους μέσω προγραμματιζόμενων συνδέσεων.

Ένα τυπικό λογικό μπλοκ ενός FPGA αποτελείται από πίνακες αντιστοίχισης, πολυπλέκτες, πύλες και φλιπ-φλοπ. Οι λειτουργίες συνδυαστικού κυκλώματος του λογικού μπλοκ υλοποιούνται από τον πίνακα αντιστοίχισης για τον σχεδιασμό και την σύνθεση.

Για να φορτωθεί ο προγραμματισμός στο κύκλωμα υπάρχουν δύο τρόποι: ο πρώτος είναι με χρήση υπολογιστή, ενώ ο δεύτερος με την χρήση PROM ή FLASH πάνω στο κύκλωμα. Ο προγραμματισμός παραμένει στην SRAM έως ότου επαναπρογραμματιστει το FPGA, ή διακοπεί η τροφοδοσία. Εφαρμογές οι οποίες απαιτούν διαφορετικά λογικά κυκλώματα για τη λειτουργία τους, μπορούν κάλλιστα να βασιστούν στα FPGA, λόγω της ιδιότητας που έχουν να είναι εύκολα επαναπρογραμματίσιμα.

Ο λόγος για τον οποίο έχει επιλεγεί η χρήση μνήμης RAM και όχι ROM στα FPGA, είναι ότι η μνήμη RAM επιτρέπει τον προγραμματισμό με απλό γράψιμο απευθείας σε αυτή. Το μειονέκτημα το οποίο έχει είναι ότι στην περίπτωση διακοπής τροφοδοσίας χάνονται όλα τα αποθηκευμένα δεδομένα.

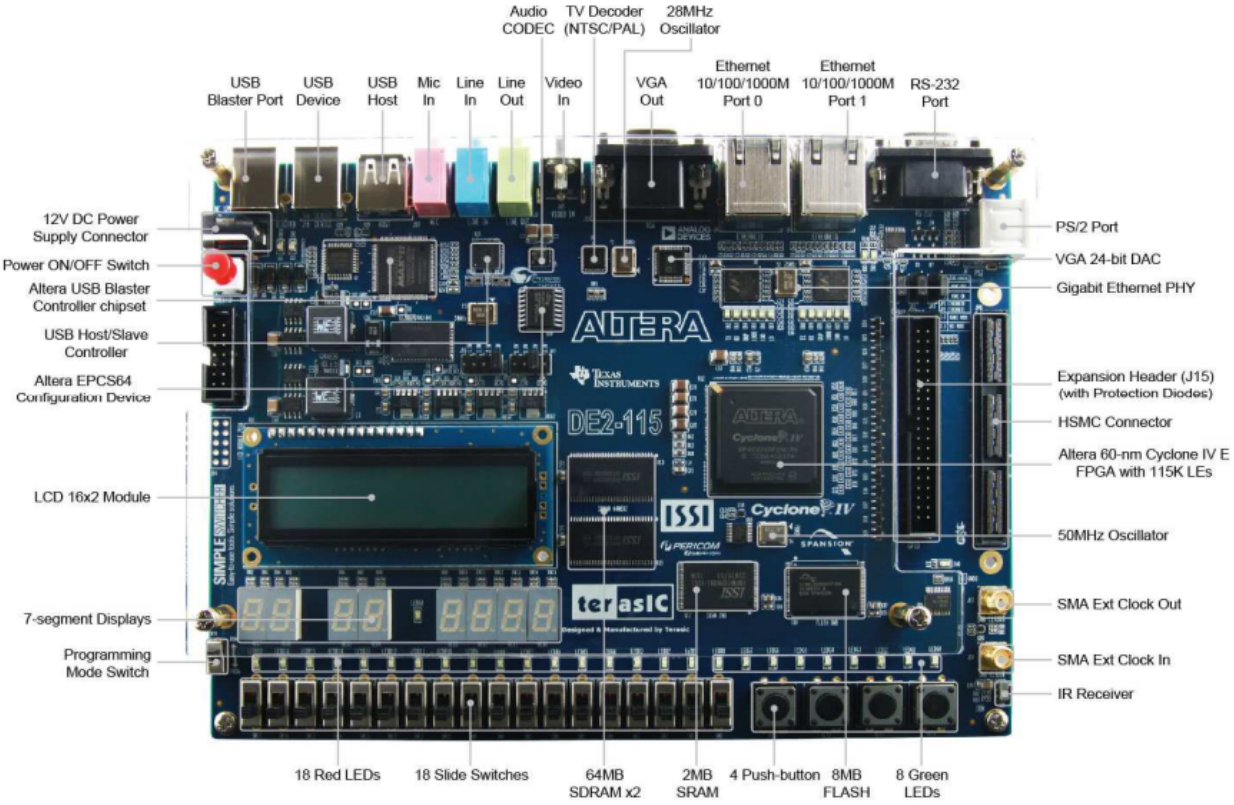
Για το σχεδιασμό και τη σύνθεση κυκλωμάτων με την χρήση CPLD ή FPGA είναι απαραίτητη η χρήση εργαλείων CAD (Computer-Aided Design). Υπάρχουν πολλά διαθέσιμα εργαλεία, καθώς και γλώσσες περιγραφής υλικού, όπως είναι η ABEL, η VHDL και η Verilog.

1.3.3 Η αναπτυξιακή πλακέτα DE2-115 της Terasic

1.3.3.1 Γνωρίζοντας την πλακέτα

Η αναπτυξιακή πλακέτα που θα χρησιμοποιήσουμε στην παρούσα διπλωματική εργασία είναι η DE2-115 της Terasic (Εικόνα 1.1). Αν και είναι σχεδιασμένη κατά βάση για ακαδημαϊκούς σκοπούς (διδασκαλία και έρευνα), εντούτοις δεν υπολείπεται ως προς τη λειτουργικότητά της και μπορεί να χρησιμοποιηθεί ακόμη και σε πραγματικές εφαρμογές.

Διαθέτει όλα εκείνα τα χαρακτηριστικά που επιτρέπουν στους χρήστες να σχεδιάσουν και να αναπτύξουν, μέσα από ένα ευρύ φάσμα προσχεδιασμένων κυκλωμάτων, από απλά ψηφιακά κυκλώματα και εφαρμογές, μέχρι διάφορες πολύπλοκες και απαιτητικές εφαρμογές πολυμέσων. Παράλληλα, παρέχει τη δυνατότητα επικοινωνίας με μια πληθώρα από περιφερειακά, όπως για παράδειγμα με πληκτρολόγιο, ηχεία και κάρτες μνήμης. Η Εικόνα 1.1 απεικονίζει τη βασική διάταξη της πλακέτας καθώς και τα κύρια περιφερειακά της.



Εικόνα 1.1 Η αναπτυξιακή πλακέτα DE2-115 [10]

1.3.3.2 Τεχνικά χαρακτηριστικά

Η πλακέτα DE2-115 διαθέτει ένα FPGA της οικογένειας Cyclone IV της Intel, και όλα τα περιφερειακά της είναι συνδεδεμένα με τους ακροδέκτες (pins) του FPGA αυτού, παρέχοντας έτσι τη δυνατότητα στον χρήστη να μπορεί να ρυθμίσει την σύνδεση πολλαπλών περιφερειακών με τον τρόπο που επιθυμεί. Για την εύκολη διεξαγωγή πειραμάτων, η πλακέτα αυτή περιλαμβάνει έναν ικανοποιητικό αριθμό διακοπτών, φωτεινών LED, καθώς και οθόνες 7 στοιχείων (7-segment displays). Για πιο προχωρημένες εφαρμογές, περιλαμβάνει μνήμες SRAM, SDRAM και επίσης μη πτητική μνήμη FLASH που διατηρεί τα δεδομένα της.

Για τα προβλήματα που απαιτούν χρήση επεξεργαστή και συσκευές εισόδου/εξόδου, μπορεί να χρησιμοποιηθεί ο επεξεργαστής Nios II, ο οποίος παρέχεται σε μορφή soft-core από την Intel, και μπορεί με τον κατάλληλο προγραμματισμό να εκμεταλλευτεί όλες τις διεπαφές της αναπτυξιακής πλακέτας. Παρακάτω αναφέρονται τα σημαντικότερα στοιχεία που περιλαμβάνονται στην πλακέτα DE2-115:

- FPGA
    - Cyclone IV 4CE115
    - EPCS64 serial configuration device
- I/O Διεπαφές
    - Θύρα USB-Blaster για την επαναδιαμόρφωση του FPGA
    - Είσοδος/Έξοδος γραμμής ήχου, είσοδος μικροφώνου (24-bit)
    - Έξοδος Video (VGA 3x8-bit DAC)
    - Είσοδος Video (NTSC/PAL/SECAM/TV-in connector)
    - Θύρα υπερύθρων
    - Σειριακή θύρα RS232 DB9
    - 10/100/1000 Ethernet (2 θύρες)
    - Δυο θύρες USB 2.0 Host (Type A/B)
    - PS/2 πληκτρολόγιο και mouse
    - Θύρες επέκτασης 40 ακροδεκτών (40-pin headers)
- Μνήμη
    - 128MB SDRAM, 2MB SRAM, 8MB Flash, 32Kbit EEPROM
    - Υποδοχή για MicroSD κάρτα μνήμης
- Απεικόνιση
    - 8 οθόνες 7-segment
    - 1 οθόνη LCD 2x16
- Switches και LEDs
    - 18 διακόπτες
    - 18 LEDs
    - 4 διακόπτες πίεσης
- Ρολόγια
    - 50 MHz clock (3 ρολόγια)
    - SMA υποδοχές για εξωτερική παροχή ρολογιού

Για να αναπτυχθεί ένα σύστημα σε οποιαδήποτε FPGA της Intel χρησιμοποιείται το λογισμικό Quartus. Υπάρχουν 2 εκδόσεις του προγράμματος, που μπορούν να μεταφορτωθούν δωρεάν από τον ιστοχώρο της Intel. Η δωρεάν έκδοση, η οποία είναι μεν πλήρης άλλα έχει κάποιους περιορισμούς ως προς τις δυνατότητες των εργαλείων, και η επαγγελματική έκδοση (subscription), η οποία απαιτεί ένα αρχείο με άδειες που έχει αγοράσει κάποιος από την Intel, και οι οποίες καθορίζουν ακριβώς τα εργαλεία, τους διαθέσιμους πυρήνες πνευματικής ιδιοκτησίας (IP), και όλες τις δυνατότητες που μπορεί να έχει στη διάθεσή του ο σχεδιαστής.

Στα πλαίσια των εργαστηριακών ασκήσεων θα χρησιμοποιήσουμε τη δωρεάν έκδοση, η οποία μας καλύπτει πλήρως. Κατά την εκπόνηση της διπλωματικής, η τελευταία διαθέσιμη έκδοση του Quartus ήταν η 18.1, η οποία υποστηρίζει 32bit και 64bit συστήματα, από Windows XP και άνω, και έτσι μπορεί να χρησιμοποιηθεί σε όλους τους υπολογιστές που κατασκευάστηκαν από το 2001 και μετά.

Κατά την αγορά της πλακέτας DE2-115 παρέχεται και το απαραίτητο λογισμικό σε CD, και έτσι δε χρειάζεται να μεταφορτωθεί από το Internet. Καλό όμως είναι να χρησιμοποιούμε πάντα τη τελευταία διαθέσιμη έκδοση, κάθως πάντα υπάρχει το ενδεχόμενο να έχουν διορθωθεί σφάλματα στο λογισμικό της εφαρμογής, σε σχέση με παλαιότερες εκδόσεις. Μαζί με το λογισμικό, υπάρχουν και αρκετοί σχεδιασμοί που εκμεταλλεύονται τις δυνατότητες της πλακέτας.

1.4 Ενσωματωμένοι Επεξεργαστές

Ένας "ενσωματωμένος" επεξεργαστής είναι απλώς ένα κύκλωμα επεξεργαστή τοποθετημένο μέσα σε ένα σύστημα που το ελέγχει. Ένας επεξεργαστής ενσωματωμένος σε ένα σύστημα χειρίζεται όλον τον υπολογισμό και τη λογική λειτουργία που απαιτεί αυτό το υπολογιστικό σύστημα. Ο ενσωματωμένος επεξεργαστής χειρίζεται επίσης διεργασίες όπως η αποθήκευση και η ανάκτηση δεδομένων από τη μνήμη, και αναλαμβάνει την επεξεργασία των δεδομένων από οποιεσδήποτε εισόδους ή προς οποιεσδήποτε εξόδους.

1.4.1 Γενικές πληροφορίες για τους επεξεργαστές

Η κύρια λειτουργία ενός επεξεργαστή είναι να χειρίζεται δεδομένα με τρόπο που προσδιορίζεται από μια ακολουθία οδηγιών. Η σειρά οδηγιών είναι αυτό από το οποίο αποτελείται το πρόγραμμα λογισμικού, το οποίο είναι γραμμένο σε κάποιο είδος γλώσσας προγραμματισμού (π.χ. C, C++ κ.τ.λ.). Το λογισμικό αποθηκεύεται σε μια μνήμη (π.χ. Flash, SRAM, SDRAM, HDD), και ο επεξεργαστής λαμβάνει τις οδηγίες διαδοχικά, χρησιμοποιώντας λογική για τη μετάφραση των οδηγιών σε σήματα ελέγχου, και έναν δείκτη εντολών, ο οποίος “δείχνει” την επόμενη εντολή.

Ένας μικροελεγκτής είναι ένας απλός επεξεργαστής που ενσωματώνει όλο το απαραίτητο υλικό για τη διασύνδεση διαφόρων περιφερειακών συσκευών. Αποτελείται από μία Κεντρική Μονάδα Επεξεργασίας (CPU) και από διάφορα κυκλώματα διασύνδεσης, συμπεριλαμβανομένης και μνήμης. Μερικά παραδείγματα πρωτοκόλλων επικοινωνίας με γενικές περιφερειακές συσκευές είναι το SPI (Serial Peripheral Interface), το RS232, το JTAG στο οποίο θα αναφερθούμε αργότερα, και το I2C.

Υπάρχουν δύο κύριοι τύποι ενσωματωμένων επεξεργαστών: Οι “μαλακοί” επεξεργαστές και οι “σκληροί” επεξεργαστές. Ο μαλακός επεξεργαστής είναι συνήθως κατασκευασμένος από λογικά στοιχεία που βρίσκονται σε FPGA και είναι σε μορφή κώδικα γλώσσας περιγραφής υλικού. Ο σκληρός επεξεργαστής, από την άλλη πλευρά, δεν είναι δυνατόν να αλλάξει αφού κατασκευαστεί. Αυτό οφείλεται στο γεγονός ότι ένας τέτοιος επεξεργαστής δεν αποτελείται από αναδιαμορφώσιμα στοιχεία όπως ο μαλακός επεξεργαστής. Οι σκληροί επεξεργαστές μπορούν να εμφανιστούν τόσο σε μορφή εξειδικευμένου ολοκληρωμένου κυκλώματος, όσο και ως μέρος ενός FPGA. Ένα παράδειγμα ενός FPGA που περιλαμβάνει έναν σκληρό επεξεργαστή είναι η οικογένεια Vertex-V FXT, της Xilinx, η οποία περιλαμβάνει την ενσωματωμένη αρχιτεκτονική επεξεργαστή PowerPC 440 της IBM.

Υπάρχουν και πλεονεκτήματα και μειονεκτήματα στη χρήση ενός μαλακού ή σκληρού ενσωματωμένου επεξεργαστή. Παρακάτω αναφέρονται τα τρία κυριότερα πλεονεκτήματα χρησιμοποίησης ενός μαλακού επεξεργαστή:

- Προσαρμοστικότητα
- Αυξημένη διάρκεια ζωής
- Μείωση στοιχείων και κόστους

Σε σύγκριση με έναν μαλακό επεξεργαστή το κύριο πλεονέκτημα του σκληρού επεξεργαστή είναι η υψηλή απόδοση και το χαμηλό κόστος.

Ο μαλακός επεξεργαστής είναι ευπροσάρμοστος και μπορεί εύκολα να αλλάξει για να ταιριάξει με τις ανάγκες κάθε εφαρμογής. Αυτό δίνει στον σχεδιαστή τη δυνατότητα να είναι ευέλικτος επιλέγοντας διάφορους συνδυασμούς από περιφερειακά και ελεγκτές. Ο σχεδιαστής μπορεί επίσης να υλοποιήσει μοναδικά περιφερειακά που μπορούν να συνδεθούν απευθείας στον επεξεργαστή, σε αντίθεση με έναν σκληρό επεξεργαστή, για τον οποίο ο σχεδιαστής πρέπει να επιλέξει μεταξύ των ενσωματωμένων επεξεργαστών που είναι σήμερα διαθέσιμοι, πολλοί από τους οποίους είτε είναι πολύ ισχυροί, ή πολύ <<αδύναμοι>> για αυτό που ο σχεδιαστής θέλει να κάνει.

Η Intel προσφέρει για τα FPGA της ένα ευρύ φάσμα πυρήνων IP δικής της πνευματικής ιδιοκτησίας, καθώς και από άλλους, τρίτους παρόχους. Για την υλοποίηση αυτών των πυρήνων, οι χρήστες μπορούν να χρησιμοποιήσουν τα εργαλεία Quartus και Platform Designer, τα οποία περιγράφονται σε επόμενα Κεφάλαια. Πυρήνες υλικού είναι επίσης διαθέσιμοι ως δωρεάν και ανοιχτού κώδικα λύσεις, υπό την ελεύθερη άδεια γενικής δημόσιας χρήσης (LGPL) στη διεύθυνση www.OpenCores.org.

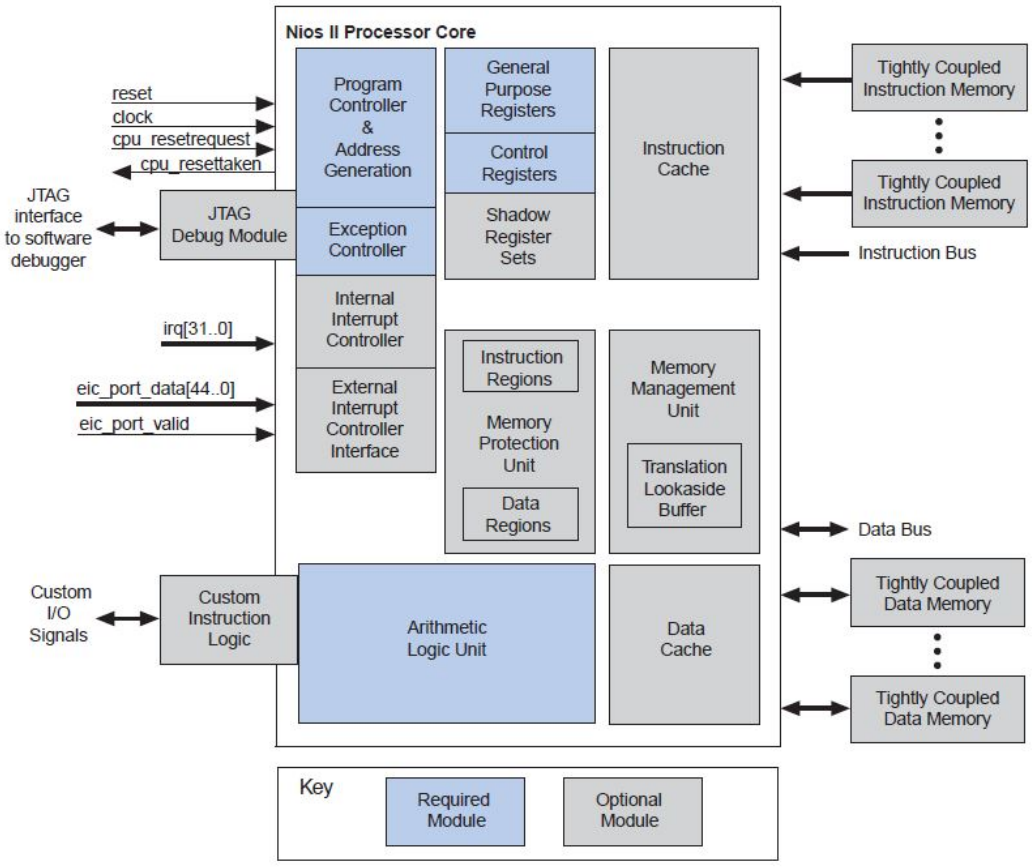
Δεδομένου ότι το υλικό ενός μαλακού επεξεργαστή περιγράφεται σε γλώσσα περιγραφής υλικού, η διάρκεια ζωής του είναι αρκετά μεγάλη. Μπορεί να εφαρμοστεί σε μελλοντικές πλατφόρμες FPGA, και μπορεί εύκολα να μεταβληθεί για να γίνει ακόμα καλύτερος. Ένα άλλο πλεονέκτημα της χρήσης ενός μαλακού επεξεργαστή είναι η μείωση χρήσης ολοκληρωμένων στοιχείων και, κατά συνέπεια, του κόστους υλοποίησης.

Η ευελιξία του FPGA επιτρέπει στον σχεδιαστή να χρησιμοποιεί μόνο ένα για να αντικαταστήσει ένα σύστημα που απαιτεί πολλαπλά ολοκληρωμένα. Με τη μείωση του αριθμού των ολοκληρωμένων ενός συστήματος, ο σχεδιαστής μπορεί επίσης να μειώσει το μέγεθος του συστήματος ώστε να εξοικονομήσει χρόνο και κόστος.

1.4.2 Ο επεξεργαστής NIOS II

Ο Nios II, είναι μια αρχιτεκτονική μαλακού ενσωματωμένου επεξεργαστή 32 bit, που έχει σχεδιαστεί για τα FPGA της Intel. Ο παλαιότερος Nios επεξεργαστής εμφανίστηκε το 2001 και ήταν ο πρώτος επεξεργαστής που δημιουργήθηκε ειδικά για το σχεδιασμό συστημάτων σε FPGA. Ο Nios II παρουσίασε διάφορες βελτιώσεις σε σχέση με τον παλαιότερο Nios, καθιστώντας τον έτσι πιο κατάλληλο για ένα ευρύτερο φάσμα εφαρμογών.

Από την πρώτη εισαγωγή του Nios, δεκάδες χιλιάδες χρήστες FPGA έχουν χρησιμοποιήσει επεξεργαστές Nios ή Nios II από την Altera (νυν Intel). Ο Xilinx MicroBlaze είναι ένα παράδειγμα ενός πολύ παρόμοιου προϊόντος, το οποίο είναι επίσης ένας μαλακός ενσωματωμένος επεξεργαστής 32-bit, και έχει σχεδιαστεί για να χρησιμοποιείται στα FPGA της εταιρίας Xilinx.



Εικόνα 1.2 Πυρήνας επεξεργαστή NIOS II [17]

Η δομή του επεξεργαστή Nios II παρουσιάζεται στην Εικόνα 1.2. Παρέχει ένα πλήρες σύνολο εντολών 32-bit και πρόσβαση σε έως και 4 GB εξωτερικής μνήμης. Επιπλέον, ο πυρήνας έχει 32 καταχωρητές γενικής χρήσης, οι οποίοι μπορούν να χρησιμοποιηθούν για την αποθήκευση δεδομένων. Ο πυρήνας υποστηρίζει επίσης 32 εξωτερικές γραμμές διακοπής. Ο πυρήνας μπορεί επίσης να εκτελέσει σύνθετες εντολές, όπως πολλαπλασιασμό ή διαίρεση τελουμένων των 32 bit.

Ένα σημαντικό χαρακτηριστικό του πυρήνα NIOS II είναι ότι υποστηρίζει το πρωτόκολλο JTAG, το οποίο είναι ένα πρότυπο για την υποβοήθηση αποσφαλμάτωσης υλικού. Το JTAG επιτρέπει δυνατότητες όπως εκκίνηση, διακοπή και βήμα-προς-βήμα εκτέλεση, για τον εντοπισμό σφαλμάτων κατά την εκτέλεση του λογισμικού. Το περιβάλλον ανάπτυξης λογισμικού βασίζεται στην αλυσίδα εργαλείων GNU C/C++ και στο περιβάλλον ανάπτυξης λογισμικού Eclipse IDE.

Τέλος, ο πυρήνας Nios II είναι διαθέσιμος σε τρεις διαφορετικούς τύπους. Και οι τρεις εκδοχές υποστηρίζουν το σύνολο των εντολών Nios II, και είναι οι Nios II/e, Nios II/s και Nios II/f. Στόχος κάθε μίας από αυτές τις εκδόσεις είναι αντίστοιχα το ελάχιστο μέγεθος πυρήνα, το μικρό μέγεθος πυρήνα και η γρήγορη ταχύτητα εκτέλεσης. Στην ελεύθερη διανομή της πλατφόρμας Quartus δεν προσφέρεται μόνο η υλοποίηση του πυρήνα Nios II/s. Ο Πίνακας 1.1 παρουσιάζει μια επισκόπηση των χαρακτηριστικών των τριών εκδοχών του Nios II.

Πίνακας 1.1 Σύγκριση εφαρμογών πυρήνα του επεξεργαστή Nios II [17]

Feature		Core		
		Nios II/e	**Nios II/s**	**Nios II/f**
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz *(1)*	0.15	0.74	1.16
	Max. DMIPS *(2)*	31	127	218
	Max. f_{MAX} *(2)*	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	Without MMU or MPU: < 1800 LEs; < 900 ALMs With MMU: < 3000 LEs; < 1500 ALMs With MPU: < 2400 LEs; < 1200 ALMs
Pipeline		1 stage	5 stages	6 stages
External Address Space		2 GBytes	2 GBytes	2 GBytes without MMU 4 GBytes with MMU

2. Εισαγωγή στην πλατφόρμα της Intel, Quartus

Σκοπός των εργαλείων σχεδίασης, προσομοίωσης και σύνθεσης (CAD tools), είναι η προσαρμογή των κυκλωμάτων, καθώς και η δημιουργία των απαραίτητων αρχείων για τον προγραμματισμό του υλικού. Το προγραμματιζόμενο υλικό στο οποίο αναφερόμαστε είναι τα κυκλώματα CPLD και FPGA.

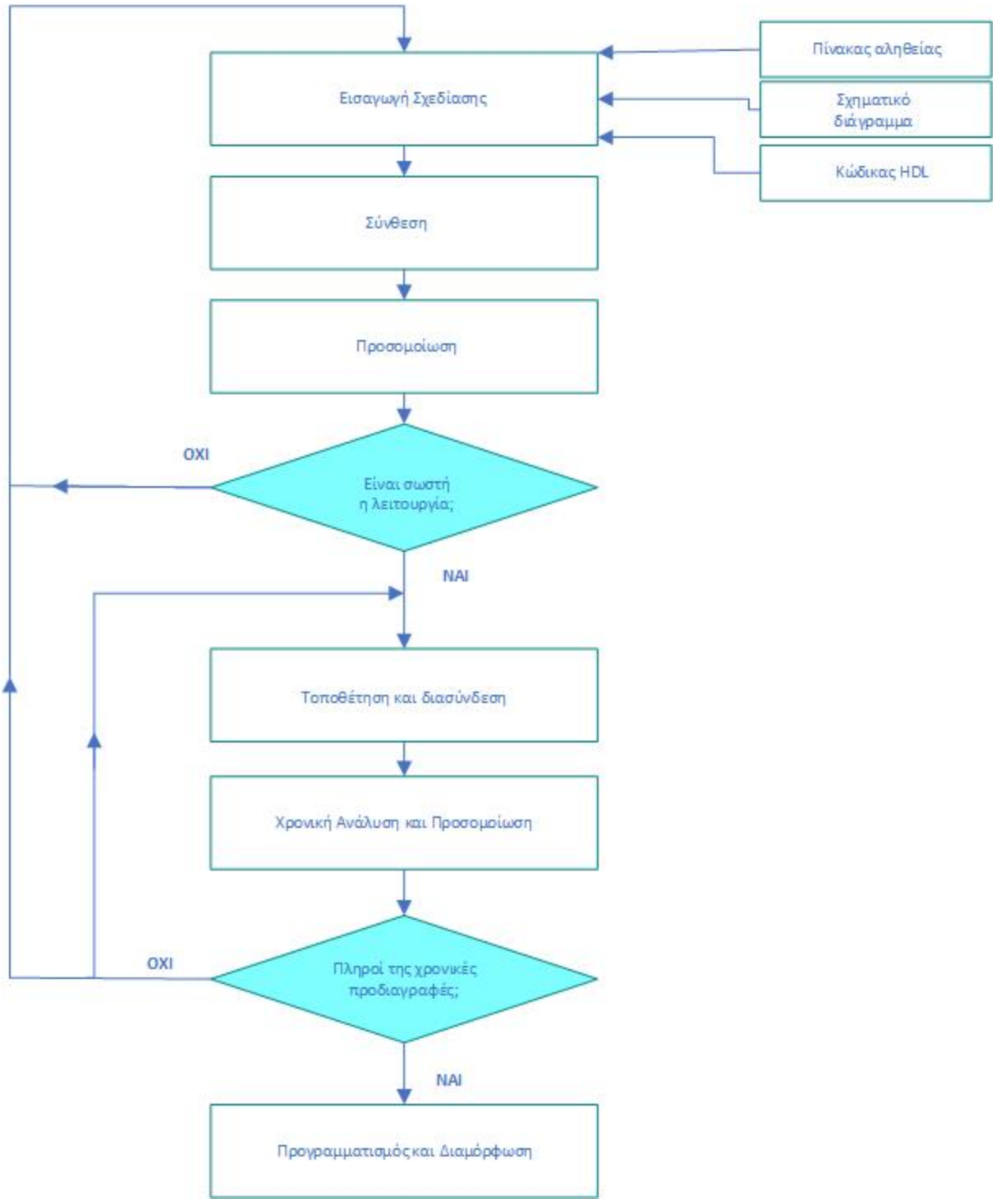
Η χρήση των εργαλείων CAD, παρέχει διάφορους τρόπους εισαγωγής της περιγραφής υλικού σε ένα κύκλωμα. Ο πρώτος τρόπος βασίζεται στην σχεδίαση των κυκλωμάτων με διατάξεις, τις οποίες λαμβάνουμε από σχετικές βιβλιοθήκες. Μια άλλη επιλογή είναι μέσω αρχείων γλώσσας περιγραφής υλικού. Παράδειγμα τέτοιας γλώσσας είναι η Verilog.

Υπάρχουν κάποια βασικά βήματα κατά τη χρήση των εργαλείων σχεδίασης, τα οποία παρατίθενται και στην Εικόνα 2.1. Τα βήματα αυτά είναι σε γενικές γραμμές όμοια για κάθε εργαλείο. Το πρώτο βήμα είναι η εισαγωγή του κυκλώματος, και ακολουθούν η σύνθεση (synthesis) και η τοποθέτηση και δρομολόγηση (place and route).

Κατά το βήμα της σύνθεσης, ακολουθείται μια αυτόματη διαδικασία σύμφωνα με την οποία το κύκλωμά μας επανασχεδιάζεται με βάση τους κανόνες της τεχνολογίας για την οποία προορίζεται. Αυτό σημαίνει ότι διαφορετικό αποτέλεσμα περιμένουμε να βγει από την διαδικασία αυτή στην περίπτωση που πρόκειται να προγραμματίσουμε ένα κύκλωμα CPLD, από την περίπτωση στην οποία πρόκειται να προγραμματίσουμε ένα κύκλωμα FPGA. Αυτό συμβαίνει γιατί διαφορετικά κυκλώματα περιέχουν διαφορετικές διατάξεις για την δημιουργία λογικών συναρτήσεων. Στην περίπτωση CPLD, οι λογικές συναρτήσεις υλοποιούνται με διατάξεις λογικων πυλών AND και OR, ενώ σε ένα FPGA γίνεται χρήση πινάκων αντιστοίχισης (Look-up Tables).

Είναι σημαντικό να σημειωθεί ότι μία προσομοίωση είναι πάντα λεπτομερής και ακριβής. Αν το αποτέλεσμα της προσομοίωσης δεν είναι ικανοποιητικό, τότε αυτό σημαίνει πως το κύκλωμα θα πρέπει να σχεδιαστεί ξανά από την αρχή. Στο σχήμα (Εικόνα 2.1) αυτό που περιγράψαμε μόλις αποτυπώνεται με το βέλος ανάδρασης.

Εάν η παρομοίωση είναι πετυχημένη, τότε λαμβάνει χώρα το επόμενο στην σειρά βήμα που είναι αυτό της δρομολόγησης του κυκλώματος (place and route ή fitting). Ο χρόνος που απαιτείται για την δρομολόγηση εξαρτάται από το μέγεθος του κυκλώματος. Σε πολύ μεγάλα κυκλώματα λοιπόν, η δρομολόγηση μπορεί να διαρκέσει πολλές ώρες. Στο βήμα αυτό γίνονται οι απαραίτητες συνδέσεις των λογικών στοιχείων, τα οποία είναι υπεύθυνα για την τέλεση των επιμέρους λειτουργιών του κυκλώματος. Οι διασυνδέσεις αυτές γίνονται στον πίνακα διασυνδέσεων του κυκλώματος (interconnection matrix).



Εικόνα 2.1 Τυπική Ροή Εργασιών σε ένα σύστημα CAD [13][14]

2.1 Το Quartus Prime

Ένα από τα γνωστότερα προγράμματα CAD είναι το Quartus Prime της Intel. Το λογισμικό αυτό ενσωματώνει όλες τις προαναφερθείσες λειτουργίες σε ένα ολοκληρωμένο περιβάλλον. Παρακάτω θα αναφερθούν όλα τα στάδια που απαιτούνται για την σχεδίαση με το λογισμικό Quartus Prime, χρησιμοποιώντας ένα παράδειγμα στο οποίο περιγράφονται αναλυτικά τα βήματα που αναφέραμε έως τώρα μόνο σε θεωρητικό επίπεδο.

2.1.1 Εισαγωγή Σχεδίασης

Το κύκλωμα που θέλει να υλοποιήσει ο χρήστης, περιγράφεται κατά το στάδιο της εισαγωγής σχεδίασης. Τα περισσότερα προγράμματα σχεδίασης υποστηρίζουν τουλάχιστον τους παρακάτω τρεις τρόπους περιγραφής ενός κυκλώματος: σχεδίαση με την χρήση πινάκων αληθείας, σχεδίαση με βάση σχηματικά διαγράμματα και σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής υλικού (HDL).

Κατά τη σχεδίαση με την χρήση πινάκων αληθείας, ο χρήστης εισάγει στο πρόγραμμα τις τιμές των εισόδων και τις επιθυμητές τιμές των εξόδων που θέλει να έχει το κύκλωμα του, και το πρόγραμμα αναλαμβάνει να δημιουργήσει το αντίστοιχο κύκλωμα. Αυτός ο τρόπος εισαγωγής συνήθως προτιμάται σε μικρά και απλά συνδυαστικά κυκλώματα.

Κατά την σχεδίαση με σχηματικό διάγραμμα, χρησιμοποιούνται σχεδιαστικά εργαλεία που παρέχει το ίδιο το πρόγραμμα, καθώς και βιβλιοθήκες οι οποίες μπορεί να περιέχουν από απλές πύλες μέχρι σύνθετα κυκλώματα. Πρόκειται για μία μορφή ιεραρχικής σχεδίασης, που δεν είναι τίποτα άλλο από την δημιουργία κυκλωμάτων από άλλα μικρότερα κυκλώματα.

Κατά τη σχεδίαση με HDL, ο χρήστης μπορεί να περιγράψει τη λειτουργία του κυκλώματος με την χρήση κατάλληλου κώδικα. Υπάρχουν αρκετές διαθέσιμες γλώσσες περιγραφής υλικού, όπως για παράδειγμα η Verilog, από τις οποίες κάποιες έχουν προτυποποιηθεί από το Ινστιτουτο Ηλεκτρολογων και Ηλεκτρονικων Μηχανικών (IEEE), ενώ κάποιες άλλες όχι.

Το πλεονέκτημα αυτού του τρόπου σχεδίασης έναντι των υπολοίπων που αναφέραμε, είναι ότι ένας σχεδιασμός μπορεί εύκολα να μεταφερθεί σε κάποιο άλλο σύστημα, χωρίς να χρειάζεται αλλαγή στον κώδικα. Παρακάτω παρατίθενται τα εργαλεία με την βοήθεια των οποίων μπορούμε να εισάγουμε περιγραφή κυκλώματος στο Quartus Prime.

- **Waveform Editor:** Χρησιμοποιείται για να δώσουμε είσοδο σε μορφή πίνακα αληθείας.
- **Block Editor:** Χρησιμοποιείται όταν θέλουμε να δώσουμε ως είσοδο ένα σχηματικό διάγραμμα.
- **Text Editor:** Χρησιμοποιείται όταν ως είσοδο χρησιμοποιούμε μία γλώσσα περιγραφής υλικού.

2.1.2 Σύνθεση (Analysis & Synthesis)

Το επόμενο βήμα μετά την Εισαγωγή Σχεδίασης είναι η Σύνθεση. Στο λογισμικό που εξετάζουμε η διαδικασία της σύνθεσης αναφέρεται ως “Analysis & Synthesis”. Κατά την διαδικασία αυτή, η είσοδος μετατρέπεται στις κατάλληλες λογικές συναρτήσεις, με βάση την τεχνολογία της συγκεκριμένης διάταξης που πρόκειται να διαμορφώσουμε.

Για παράδειγμα θα μπορούσε να έχει ως είσοδο ένα σχηματικό διάγραμμα, και σαν στόχο ένα κύκλωμα FPGA. Κατά την σύνθεση του διαγράμματος θα παίρναμε ως έξοδο λογικές εξισώσεις, ισοδύναμες με το κύκλωμα του σχηματικού διαγράμματος, οι οποίες μπορούν να υλοποιηθούν με πίνακες αντιστοίχισης. Όταν έχουμε ως είσοδο κώδικα σε γλώσσα περιγραφής υλικού, τότε κατά τη διαδικασία της σύνθεσης, τίθεται σε λειτουργία ο μεταφραστής, και ως έξοδο λαμβάνουμε την περιγραφή του κυκλώματος σε χαμηλό netlist, το οποίο μπορεί στην συνέχεια να υλοποιηθεί με την τεχνολογία που έχει οριστεί σαν στόχος.

Συμπερασματικά, η σύνθεση είναι ένα αυτοματοποιημένο εργαλείο μετατροπής του αρχικού κυκλώματος που δίνει ο σχεδιαστής, στην τελική τεχνολογία του στόχου. Κατά τη σύνθεση πραγματοποιείται επίσης και βελτιστοποίηση του κυκλώματος.

2.1.3 Τοποθέτηση και διασύνδεση (place and route)

Μια άλλη διαδικασία είναι η τοποθέτηση και διασύνδεση (place and route). Κατά τη διαδικασία αυτή το κύκλωμα τοποθετείται σε συγκεκριμένες περιοχές της προγραμματιζόμενης συσκευής και γίνεται η διασύνδεση των στοιχείων του.

Στο σημείο αυτό σημαντικό ρόλο παίζουν και οι χρονικές απαιτήσεις καθώς και οι απαιτήσεις κατανάλωσης που έχουν τεθεί από τον σχεδιαστή. Κατά τη συγκεκριμένη διαδικασία γίνεται ο καθορισμός των λογικών στοιχείων (logic elements-LE’s) του FPGA που θα χρησιμοποιηθούν. Επίσης επιλέγονται οι κατάλληλες γραμμές ώστε να διασυνδεθούν τα LE μεταξύ τους.

2.1.4 Χρονική ανάλυση και παραγωγή αρχείων προγραμματισμού

Σε συνέχεια της διαδικασίας τοποθέτησης και διασύνδεσης πραγματοποιείται η χρονική ανάλυση (timing analysis). Η χρονική ανάλυση υπολογίζει τους καλύτερους και τους χειρότερους χρόνους λειτουργίας ενός κυκλώματος και βασίζεται στις προβλεπόμενες καθυστερήσεις των μονοπατιών του κυκλώματος.

Οι καθυστερήσεις προκύπτουν από τα επίπεδα λογικής κάθε μονοπατιού και τις μεταξύ τους διασυνδέσεις. Η χρονική ανάλυση είναι καθοριστικής σημασίας για ένα κύκλωμα, καθώς οι χρόνοι που προβλέπει θα πρέπει να είναι σε συμφωνία με τους χρόνους που καθορίζονται από το επιθυμητό ρολόι του συστήματος. Στην περίπτωση μη συμφωνίας των χρόνων αυτών, θα πρέπει να τεθούν κατάλληλοι χρονικοί περιορισμοί (timing assignments) στο λογισμικό και να επαναληφθεί η διαδικασία τοποθέτησης και διασύνδεσης, για να βελτιωθούν οι χρόνοι απόκρισης του συστήματος.

Κατά την τελική φάση (assembling) δημιουργούνται τα αρχεία προγραμματισμού της συσκευής που θα χρησιμοποιηθούν για τους σκοπούς της τελικής διαμόρφωσης. Η διαδικασία της μετάφρασης (compilation) λοιπόν που αναφέρθηκε και προηγουμένως, αποτελείται από τα στάδια που αναφέραμε παραπάνω, και είναι τα εξής: η σύνθεση, η τοποθέτηση και προσαρμογή, η χρονική ανάλυση, και η διαδικασία παραγωγής του τελικών αρχείων προγραμματισμού της συσκευής.

Μέσω της διαδικασίας αυτής <<μεταφέρεται>> το επιθυμητό κύκλωμα σε μια προγραμματιζόμενη διάταξη (FPGA, CPLD) και δημιουργείται ο κώδικας για την προσομοίωση λειτουργίας (simulation) του κυκλώματος, και τον προγραμματισμό των διατάξεων (device programming).

2.1.5 Προσομοίωση

Η λειτουργία της προσομοίωσης μας επιτρέπει να ελέγξουμε εάν το κύκλωμα μας λειτουργεί σωστά. Δεν αρκεί μόνο η λειτουργία της σύνθεσης κατά την οποία γίνεται βελτιστοποίηση του κυκλώματος μας, για να είμαστε σίγουροι ότι το κύκλωμα που έχουμε σχεδιάσει λειτουργεί σωστά.

Ο έλεγχος της σωστής λειτουργία του κυκλώματος γίνεται στα πλαίσια της προσομοίωσης. Κατά τη λειτουργία της προσομοίωσης, ο χρήστης δίνει κάποιες τιμές στην είσοδο του κυκλώματος, έτσι ώστε να παρατηρήσει εάν στην έξοδο θα εμφανιστουν οι επιθυμητές τιμές.

Είναι σημαντικό στο σημείο αυτό να αναφερθούμε στους δυο δυνατούς τύπους προσομοίωσης, στη λειτουργική (functional), και στην προσομοίωση χρόνου (timing). Στη λειτουργική προσομοίωση δεν λαμβάνεται καθόλου υπόψη ο χρόνος παρά μόνο επαληθεύεται ότι η λογική που υλοποιεί το κύκλωμα είναι ορθή, ως προς το αποτέλεσμα που παράγει.

Στην περίπτωση της προσομοίωσης χρόνου, εξετάζουμε την ορθότητα του κυκλώματος με βάση τους χρονικούς περιορισμούς που έχουμε θέσει. Πρακτικά, η χρονική προσομοίωση είναι λειτουργική προσομοίωση στην οποία όμως χρησιμοποιείται και χρονική πληροφορία και, συγκεκριμένα, οι καθυστερήσεις των στοιχείων του κυκλώματος.

2.1.6 Προγραμματισμός και διαμόρφωση της συσκευής

Το τελευταίο βήμα που απαιτείται για τη δημιουργία ενός κυκλώματος είναι η διαδικασία του προγραμματισμού της συσκευής. Όπως ήδη γνωρίζουμε τα ολοκληρωμένα συστήματα CPLD και FPGA πρέπει να προγραμματιστούν κατάλληλα ώστε να υλοποιήσουν το κύκλωμα που έχουμε σχεδιάσει. Κατά τη διαδικασία assembling παράγονται τα απαιτούμενα αρχεία για τον προγραμματισμό και τη διαμόρφωση της συσκευής. Αυτό αποτελεί το τελευταίο βήμα της διαδικασίας μετάφρασης όπως αναφέραμε παραπάνω.

Για τον προγραμματισμό των ολοκληρωμένων της Intel υπάρχουν δύο επιλογές. Η πρώτη είναι μέσω του κυκλώματος διεπαφής JTAG και η άλλη είναι το Active Serial (AS) mode. Για τη μεταφορά των αρχείων διαμόρφωσης από τον υπολογιστή στην πλακέτα (board) απαιτείται συγκεκριμένη σύνδεση, μέσω κατάλληλου interface (USB-Blaster ή BYTE-BLASTER). Η σύνδεση της πλακέτας με τον υπολογιστή γίνεται μέσω καλωδίου, το οποίο συνδέεται σε θύρα USB του υπολογιστή και στην πλακέτα (board) όπου βρίσκεται το ολοκληρωμένο κύκλωμα CPLD ή FPGA.

Στην περίπτωση που χρησιμοποιείται η διεπαφή JTAG, τα δεδομένα πηγαίνουν απευθείας στο ολοκληρωμένο κύκλωμα. Αποτελεί δηλαδή μια απλή και άμεση επιλογή για την διαμόρφωση ενός κυκλώματος. Έτσι λοιπόν το ολοκληρωμένο, στην περίπτωση που είναι FPGA, διατηρεί την διαμόρφωση που του έχουμε δώσει για όσο διάστημα τροφοδοτείται με τάση. Εάν διακοπεί η τροφοδοσία τότε χάνεται η διαμόρφωση του. Αυτό δεν ισχύει στη περίπτωση που έχουμε επιλέξει ένα ολοκληρωμένο CPLD. Τα κυκλώματα αυτά διαμορφώνονται μόνιμα, δίνοντας όμως και την δυνατότητα επανεγγραφής τους, όπως στις μνήμες flash EEPROM.

Στην περίπτωση χρήσης του Active Serial mode, μια μνήμη (π.χ. FLASH) που βρίσκεται δίπλα στο προς προγραμματισμό FPGA χρησιμοποιείται για την αποθήκευση των αρχείων διαμόρφωσης. Στην περίπτωση αυτή και αφού μεταφερθούν τα αρχεία προγραμματισμού στη μνήμη, το σύστημα δεν επηρεάζεται από την ύπαρξη τροφοδοσίας ή μη. Όσον αφορά την πλακέτα DE2-115, ο διακόπτης RUN/PROG που βρίσκεται πάνω σε αυτή, μας βοηθά να καθορίσουμε ποιον από τους δύο τρόπους προγραμματισμού θα επιλέξουμε. Προεπιλεγμένη είναι η διαμόρφωση JTAG, ενώ η δεύτερη επιλογή είναι η Active Serial.

Για να ελέγξουμε εάν το σύστημα μας τελικά λειτουργεί σωστά, θα πρέπει να εφαρμόσουμε κατάλληλες εισόδους 0 και 1 με την χρήση των διακοπτών ή άλλων συσκευών που βρίσκονται πάνω στην πλακέτα. Το αποτέλεσμα αυτών μπορούμε να το δούμε στις συσκευές απεικόνισης της πλακέτας, αρκεί να κάνουμε τις σωστές συνδέσεις των ακροδεκτών εξόδου του FPGA με τις συσκευές αυτές.

2.2 Platform Designer

Ο Platform Designer είναι ένα εργαλείο ανάπτυξης ψηφιακών συστημάτων, βασισμένων στον επεξεργαστή Nios, τα οποία μπορούν να περιλαμβάνουν περιφερειακά, μνήμες και διεπαφές επικοινωνίας, όπως είναι το USB και το SPI, και χρησιμοποιείται σε συνδυασμό με το λογισμικό Quartus. Ο σχεδιαστής μπορεί να δημιουργήσει εύκολα ένα σύστημα επιλέγοντας απλά τις επιθυμητές μονάδες και παραμετροποιώντας τες κατάλληλα.

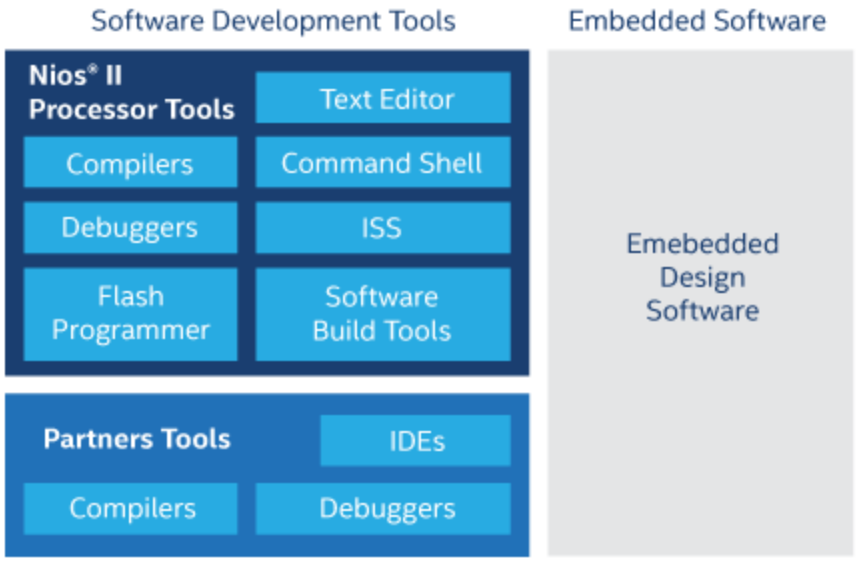
Ο Platform Designer αυτοματοποιεί πλήρως την διαδικασία ενσωμάτωσης μονάδων υλικού, ενώ χρησιμοποιώντας παραδοσιακές μεθόδους σχεδιασμού, ο σχεδιαστής μπορεί να περιγράψει (π.χ. μέσω HDL) και να ενσωματώσει και τις δικές του μονάδες, και να τις συνδέσει με τα υπόλοιπα τμήματα του συστήματος. Ο σχεδιαστής χρειάζεται μόνο να επιλέξει και να καθορίσει τα στοιχεία του συστήματος μέσα από ένα απλό γραφικό περιβάλλον, και ο Platform Designer δημιουργεί αυτόματα τις λογικές διασυνδέσεις.

Τέλος, ο Platform Designer παράγει αρχεία HDL που περιγράφουν όλα τα επιμέρους στοιχεία του συστήματος, και τη διασύνδεσή τους στο συνολικό σύστημα (είτε σε Verilog, είτε σε VHDL). Τα αρχεία αυτά χρησιμοποιούνται σαν είσοδος στο Quartus.

2.3 Eclipse Nios II Embedded Design Suite (EDS)

Η συγγραφή κώδικα για τον επεξεργαστή Nios II είναι παρόμοια με τη διαδικασία ανάπτυξης λογισμικού για μία οποιαδήποτε άλλη οικογένεια μικροεπεξεργαστών. Το Nios II EDS παρέχει ένα σταθερό περιβάλλον ανάπτυξης λογισμικού που μπορεί να χρησιμοποιηθεί για οποιοδήποτε σύστημα στηρίζεται στον επεξεργαστή Nios II.

Περιλαμβάνει πολλά ιδιόκτητα εργαλεία (Εικόνα 2.2), αλλά και εργαλεία ανοικτού κώδικα, όπως το GNU C/C++ toolchain, για τη δημιουργία προγραμμάτων για τους επεξεργαστές Nios II. Επιπλέον, το Nios II EDS διαθέτει ενσωματωμένο Layer Abstraction Hardware (HAL), το οποίο είναι λογισμικό που λειτουργεί ως διεπαφή μεταξύ του υλικού και του λογισμικού που γράφει ο χρήστης. Η διεπαφή HAL (ή διαφορετικά API), είναι ενσωματωμένη στη βασική βιβλιοθήκη ANSI C. Ο προγραμματισμός ενός συστήματος που στηρίζεται στον επεξεργαστή Nios II γίνεται μέσω της μονάδας JTAG αυτού.



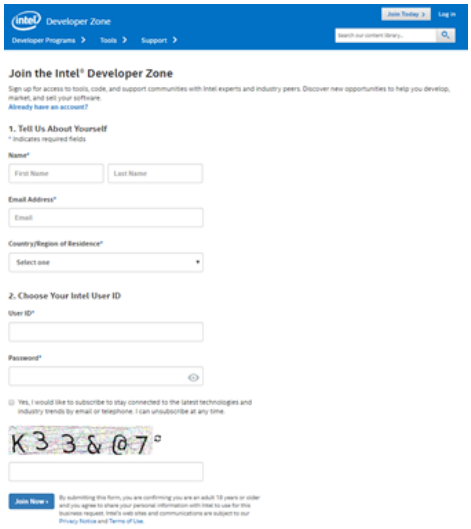
Εικόνα 2.2 Σουίτα εργαλείων σχεδίασης επεξεργαστών Nios II [21]

3. Άσκηση 0 - Εισαγωγική Άσκηση

Σε αυτό το κεφάλαιο θα περιγράψουμε, με τη μορφή μίας απλής άσκησης, τις βασικές λειτουργίες του εργαλείου σχεδίασης Quartus Prime, της Intel. Τα βήματα που θα ακολουθήσουμε ξεκινούν από την εγκατάσταση του εργαλείου και τη διαδικασία σχεδίασης ενός ψηφιακού κυκλώματος (μέσω του Platform Designer), και φτάνουν στον προγραμματισμό του FPGA της πλακέτας, ώστε να υλοποιεί το κύκλωμα που σχεδιάσαμε.

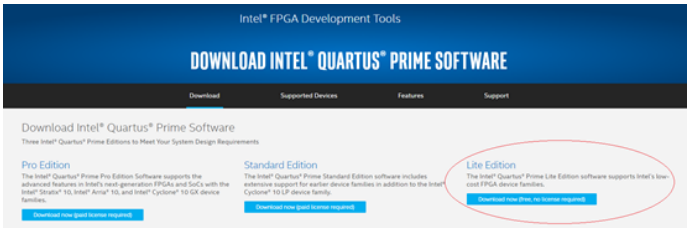
3.1 Εγκατάσταση Quartus Prime Lite Edition

Για να κατεβάσουμε επιτυχώς το εργαλείο θα πρέπει να δημιουργήσουμε έναν δωρεάν λογαριασμό και να κάνουμε εγγραφή στη σελίδα της Intel. Μεταβαίνουμε στη σελίδα https://software.intel.com/registration/, συμπληρώνουμε τα απαραίτητα στοιχεία και έπειτα κατεβάζουμε από τη σελίδα της Intel το δωρεάν εργαλείο Quartus Prime Lite Edition.



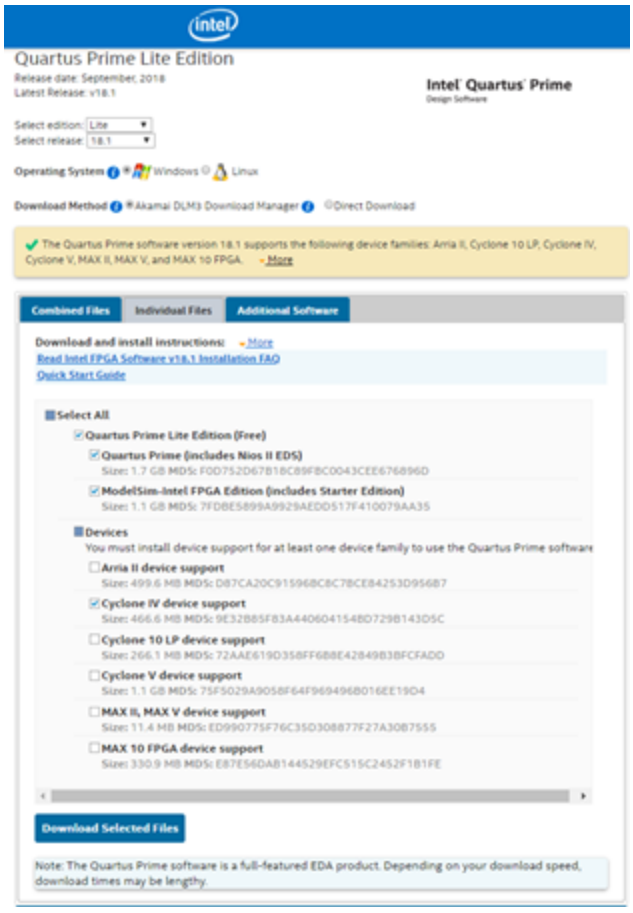
Εικόνα 3.1 Φόρμα εγγραφής [22]

Η σελίδα από την οποία μπορούμε να κατεβάσουμε το λογισμικό είναι η:
https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html



Εικόνα 3.2 Κατέβασμα εφαρμογής [22]

Στην σελίδα που θα μας ανοίξει (Εικόνα 3.3), αφού επιλέξουμε την έκδοση “Lite Edition”, θα πρέπει να ορίσουμε και κάποιες επιπλέον παραμέτρους για να μπορέσουμε να κατεβάσουμε το κατάλληλο για εμάς λογισμικό. Αφήνουμε τα πεδία “Select Edition” και “Select Release” ως έχουν καθώς αφορούν την επιλογή της έκδοσης, τα οποία είναι ήδη προεπιλεγμένα με τις καταχωρήσεις “Lite”, και την τελευταία έκδοση του λογισμικού αντίστοιχα. Στην προκειμένη περίπτωση, η έκδοση που έχει επιλεγεί είναι η 18.1. Στην παρούσα εργασία χρησιμοποιήθηκε υπολογιστής με λειτουργικό Windows 10 οπότε ακολούθως επιλέχθηκε το αντίστοιχο πεδίο. Επιλέγουμε για μέθοδο κατεβάσματος τον διαχειριστή λήψεων “Akamai DLM3 Download Manager” ο οποίος αποτελεί έναν ασφαλέστερο τρόπο λήψης και επιπλέον θα έχουμε λιγότερες πιθανότητες να κατεβάσουμε κατεστραμμένα ή κατακερματισμένα αρχεία.



Εικόνα 3.3 Επιλογή παραμέτρων λογισμικού [22]

Ακολουθούν τρεις καρτέλες από τις οποίες διαλέγουμε αυτή με το τίτλο “Individual Files” ώστε να κατεβάσουμε μόνο τα αρχεία που μας ενδιαφέρουν. Στη συνέχεια αφήνουμε επιλεγμένο το πρώτο πεδίο, που είναι το εργαλείο που μας ενδιαφέρει, και απενεργοποιούμε όλες τις υπόλοιπες επιλογές που μας δίνονται, εκτός από το πεδίο που αφορά τα συμπληρωματικά αρχεία της οικογένειας του ολοκληρωμένου της πλακέτας που έχουμε στην κατοχή μας, η οποία είναι η Cyclone IV.

Τέλος πατάμε το κουμπί για να κατέβουν τα αρχεία. Αφού τελειώσει το κατέβασμα των αρχείων χρησιμοποιούμε το αρχείο QuartusLiteSetup-18.1.X.XXX-windows ώστε να εγκαταστήσουμε το Quartus στον υπολογιστή μας. Προσέχουμε μόνο στην αρχική επιλογή να επιλέξουμε το πεδίο όπου μας ενημερώνει ότι για την εγκατάσταση θα χρησιμοποιηθεί η δωρεάν έκδοση του αρχείου.

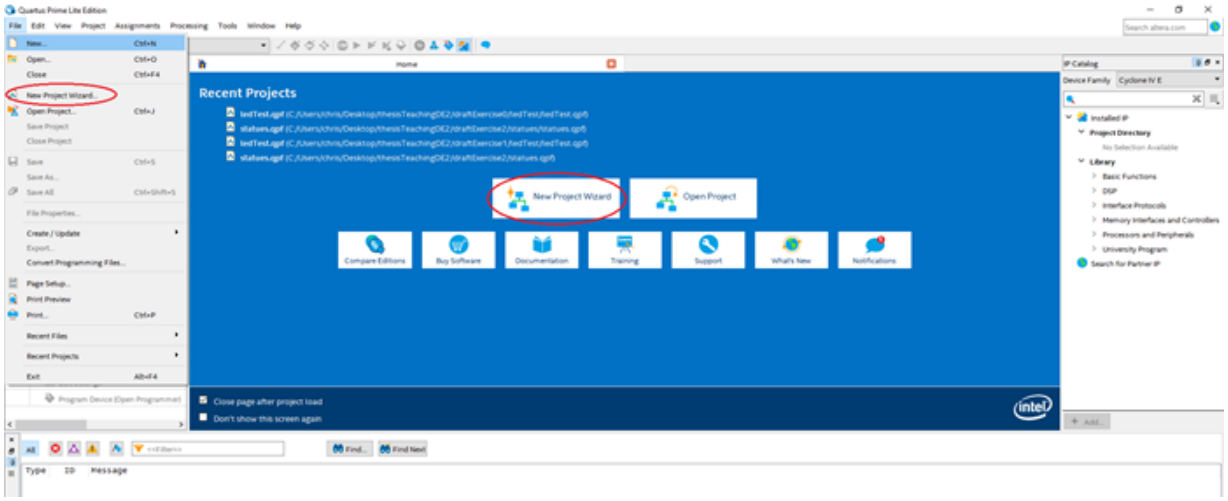
3.2 Δημιουργία νέου περιβάλλοντος εργασίας στο Quartus Prime

3.2.1 Ορισμός του έργου (project)

Η πρώτη άσκηση θα είναι εισαγωγική και θα αφορά την επιβεβαίωση της λειτουργίας της πλακέτας χρησιμοποιώντας ένα πολύ απλό παράδειγμα. Το παράδειγμα που θα υλοποιήσουμε αφορά τον έλεγχο των LED (λαμπάκια) της εκπαιδευτικής πλακέτας χρησιμοποιώντας τα SWITCHES (διακόπτες). Κάθε SWITCH θα αντιστοιχεί σε κάθε ένα από τα LED και θα μπορεί να το ελέγξει, δηλαδή να το ανάψει και να το σβήσει. Το λογισμικό του παραδείγματος μπορεί να βρεθεί στο εγχειρίδιο του εργαλείου “Quartus Prime Handbook” που υπάρχει σε διάφορες εκδόσεις στη σελίδα της Intel, καθώς επίσης και στα εγχειρίδια χρήσης που συνοδεύουν την πλακέτα.

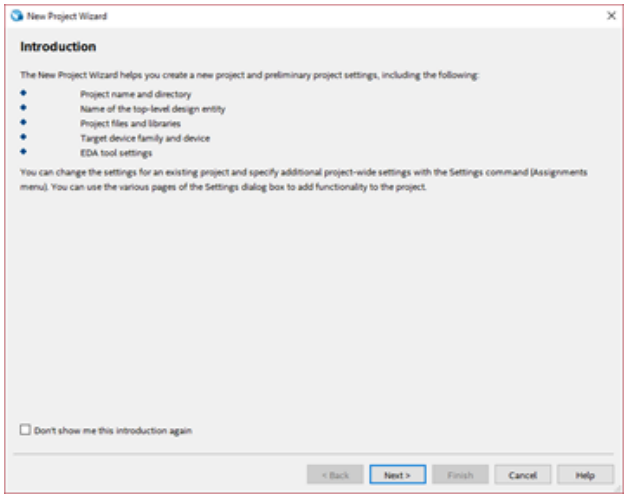
Ανοίγοντας το σχεδιαστικό περιβάλλον του Quartus Prime, θα πρέπει να δημιουργήσουμε ένα νέο project. Αυτό μπορεί να γίνει μέσω της γραμμής μενού πατώντας “File”->“New Project Wizard”. Για τη συγκεκριμένη λειτουργία υπάρχει και συντόμευση στην αρχική καρτέλα “Home” η οποία έχει ανοίξει μαζί με το εργαλείο.

Ένα project περιλαμβάνει το σύνολο των αρχείων που δημιουργούμε εμείς, καθώς επίσης και όποια αρχεία δημιουργεί το πρόγραμμα για να εκτελέσει τις λειτουργίες του. Με λίγα λόγια, σαν project θεωρούμε το σύνολο των αρχείων που δημιουργούνται για το κύκλωμα που υλοποιούμε.



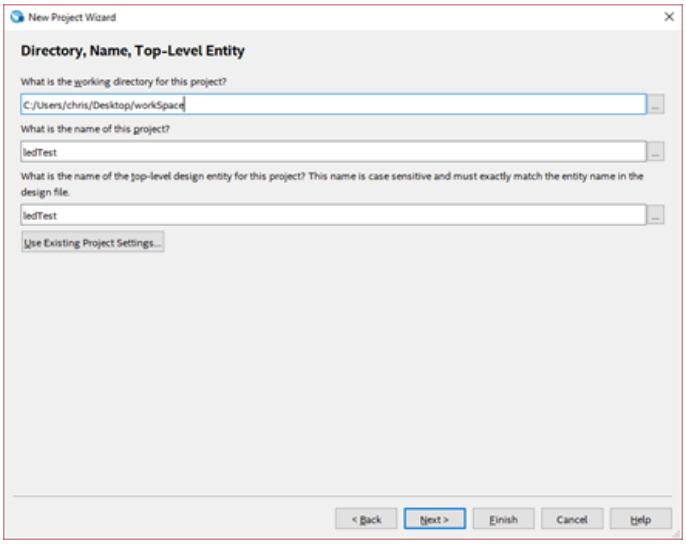
Εικόνα 3.4 Δημιουργία νέου project

➢ Αφού πατήσουμε πάνω στο κουμπί “New Project Wizard” ανοίγει ένα παράθυρο-πλοηγός για τη δημιουργία του project (Εικόνα 3.4). Το πρώτο παράθυρο που εμφανίζεται αφορά κάποιες εισαγωγικές πληροφορίες (Εικόνα 3.5), οι οποίες μας εξηγούν πως μπορεί να μας βοηθήσει ο πλοηγός και γενικά τι πληροφορίες θα χρειαστεί να εισάγουμε για να δημιουργήσουμε το project επιτυχώς.



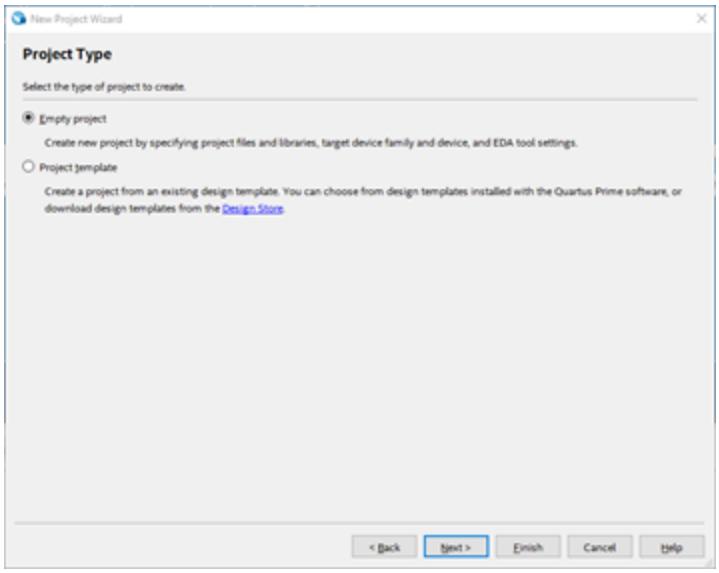
Εικόνα 3.5 Εισαγωγικές πληροφορίες πλοηγού “Project Wizard”

➢ Στη συνέχεια, πατώντας “Next” εμφανίζεται η καρτέλα όπου θα χρειαστεί να εισάγουμε το path (διαδρομή) στην οποία θα δημιουργηθεί το project, το όνομά του, καθώς επίσης και το όνομα του φακέλου που θα το περιέχει (Εικόνα 3.6). Καλό είναι να δημιουργήσουμε έναν διαφορετικό φάκελο, για κάθε ξεχωριστή εφαρμογή που υλοποιούμε και να μην αποθηκεύουμε στον ίδιο φάκελο δύο διαφορετικά σχέδια (projects). Μια καλή πρακτική είναι να χρησιμοποιούμε το ίδιο όνομα για τα πεδία της καρτέλας της Εικόνας 3.6. Στο παράδειγμά μας χρησιμοποιούμε το ledTest. Έπειτα πατάμε Next.



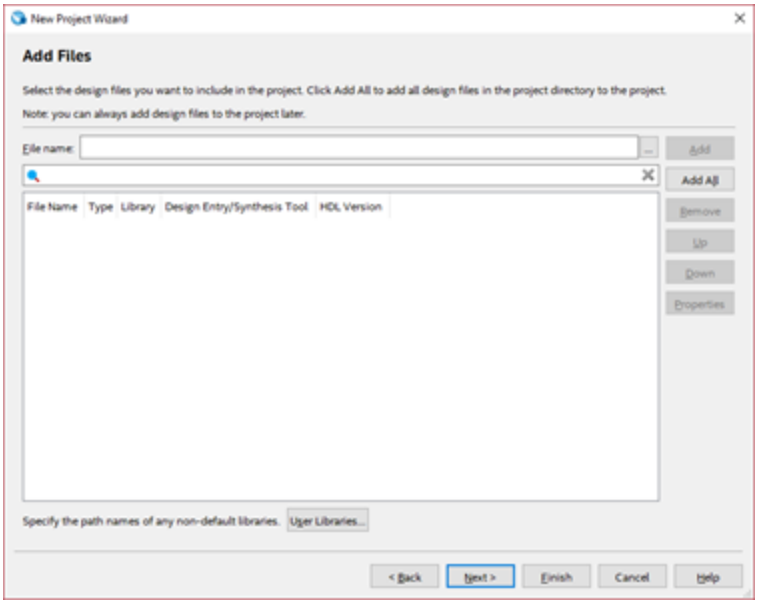
Εικόνα 3.6 Δημιουργία φακέλου και ονομασία του project

➢ Το νέο παράθυρο που θα ανοίξει (Εικόνα 3.7) αφορά τον τύπο του project. Υπάρχουν δύο επιλογές. Η πρώτη είναι να δημιουργήσουμε ένα νέο κενό project και η δεύτερη μας προτρέπει να χρησιμοποιήσουμε ένα υπάρχον ως πρότυπο (template). Επιλέγουμε την πρώτη από τις δύο επιλογές, δηλαδή να δημιουργήσουμε ένα κενό project και προχωράμε πατώντας Next.



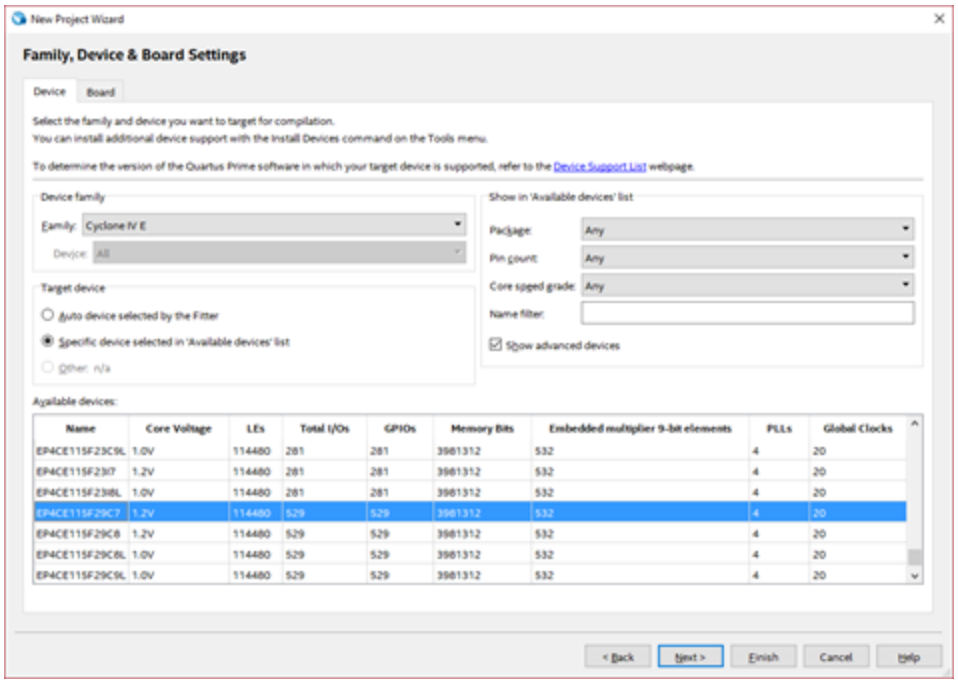
Εικόνα 3.7 Επιλογή του τύπου του project

➢ Στο επόμενο παράθυρο το οποίο φαίνεται στην Εικόνα 3.8, μπορούμε να εισάγουμε υπάρχοντα αρχεία που θέλουμε να συμπεριλάβουμε στο νέο μας project. Στην περίπτωσή μας δεν εισάγουμε κάτι και πατάμε Next.



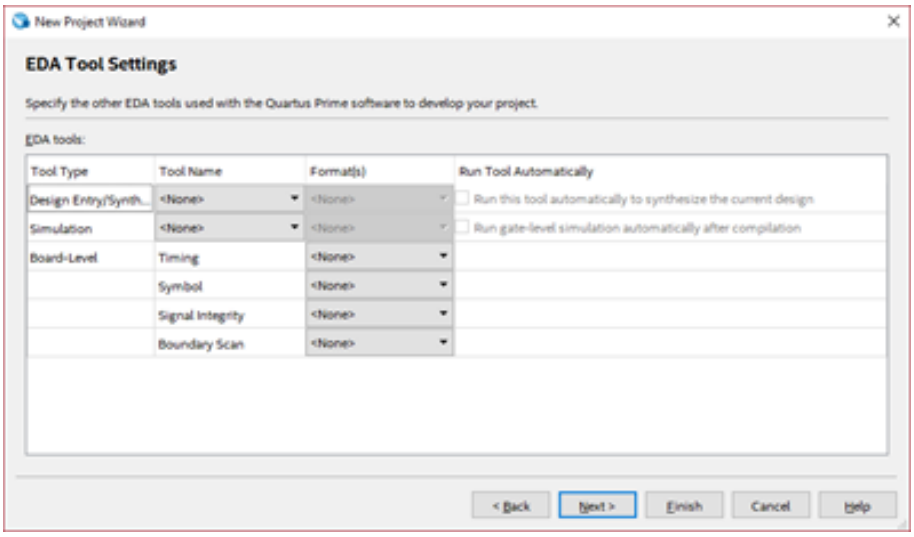
Εικόνα 3.8 Προσθήκη αρχείων στο project

➢ Στο επόμενο παράθυρο (Εικόνα 3.9) δίνουμε κάποια στοιχεία για το ολοκληρωμένο κύκλωμα FPGA της πλακέτας μας. Καθώς θα επανέλθουμε στο συγκεκριμένο παράθυρο δεν χρειάζεται να δώσουμε κάτι επιπλέον ή να αλλάξουμε κάποια από τις προεπιλεγμένες τιμές, εκτός από το να επιλέξουμε από τη λίστα το ακριβές chip που θα χρησιμοποιήσουμε. Είναι πολύ σημαντικό να επιλέξουμε σωστά τον κωδικό που θα βρούμε τυπωμένο πάνω στο chip της πλακέτας μας ώστε να μην έχουμε κάποια ασυμβατότητα στη συνέχεια, κατά το compilation του project, ή τον προγραμματισμό του ολοκληρωμένου. Ο κωδικός του chip της πλακέτας που χρησιμοποιήθηκε για την υλοποίηση της εργασίας είναι ο EP4CE115F29C7N, της οικογένειας Cyclone IV. Πατάμε Next και οδηγούμαστε στο προτελευταίο παράθυρο του πλοηγού, το οποίο φαίνεται στην Εικόνα 3.10.



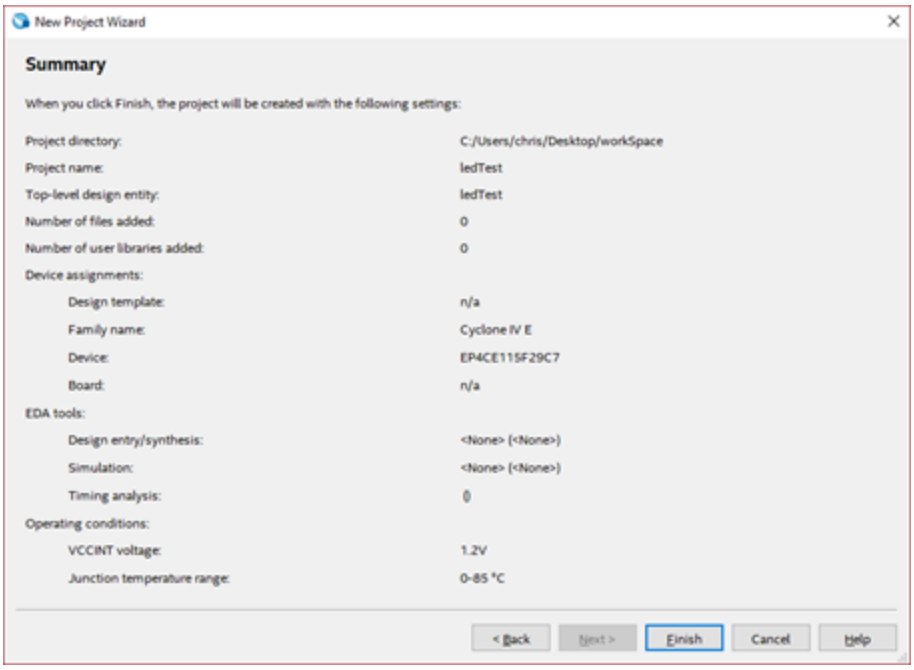
Εικόνα 3.9 Επιλογή του FPGA που θα προγραμματίσουμε

➢ Στο παράθυρο αυτό μπορούμε να δηλώσουμε αν θα χρησιμοποιηθεί κάποιο επιπλέον εργαλείο για την ανάπτυξη του project, όπως για παράδειγμα κάποιο εργαλείο προσομοίωσης. Αφήνουμε όλα τα πεδία ως έχουν και πατάμε Next.



Εικόνα 3.10 Επιλογή τρίτων εργαλείων

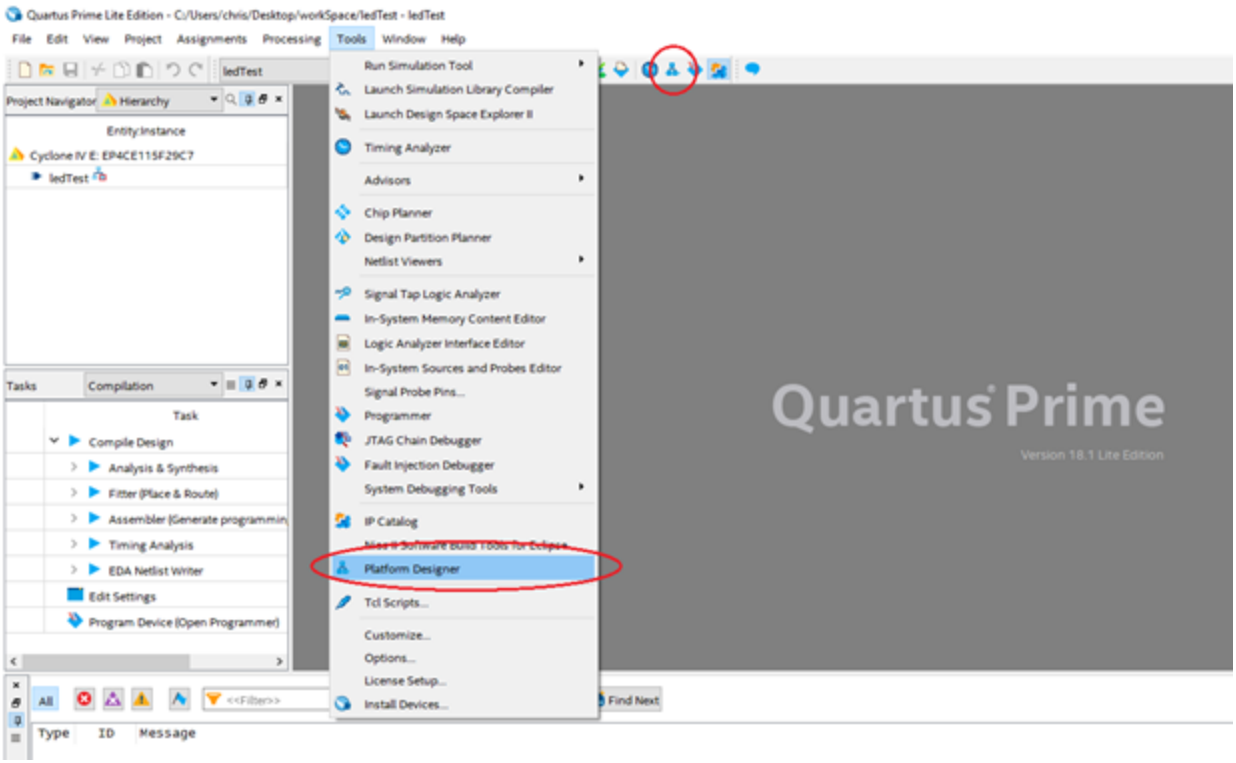
➢ Στο τελικό παράθυρο του πλοηγού (Εικόνα 3.11) βλέπουμε τις επιλογές που έχουμε κάνει, τις επιβεβαιώνουμε και πατάμε Finish. Σε κάθε βήμα μπορούμε να γυρίσουμε πίσω και να διορθώσουμε/αλλάξουμε κάποια από τις επιλογές μας πατώντας το κουμπί “Back”. Πλέον είμαστε έτοιμοι να ξεκινήσουμε να χτίζουμε το project μας.



Εικόνα 3.11 Επισκόπηση πληροφοριών του project

3.2.2 Σχεδιασμός συστήματος με χρήση του Platform Designer

Όπως αναφέρθηκε, οι ασκήσεις θα πραγματοποιηθούν με τη βοήθεια του Platform Designer. Για να ανοίξει υπάρχουν δύο τρόποι. Ο πρώτος είναι είτε μέσω της γραμμής του μενού, πατώντας την επιλογή “Tools->Platform Designer”, είτε μέσω της γραμμής εργαλείων απευθείας αφού εντοπίσουμε το αντίστοιχο εικονίδιο (Εικόνα 3.12).

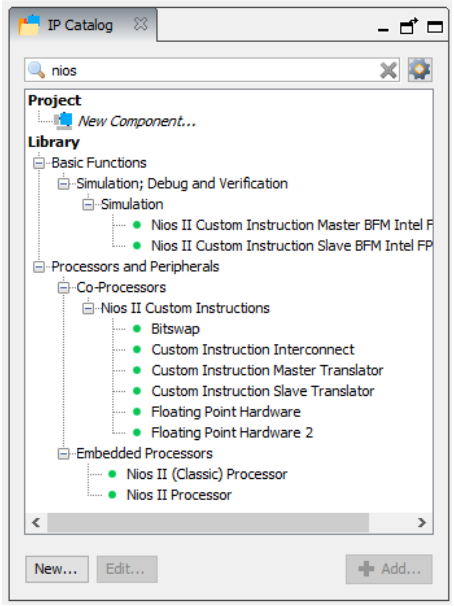


Εικόνα 3.12 Ξεκίνημα Platform Designer

Μόλις ανοίξει το παράθυρο είμαστε έτοιμοι να εντάξουμε στο project τα κατάλληλα components (στοιχεία) που θέλουμε να χρησιμοποιήσουμε. Τα στοιχεία που θα χρειαστούμε και θα πρέπει να περιλαμβάνονται στο project είναι τα παρακάτω:

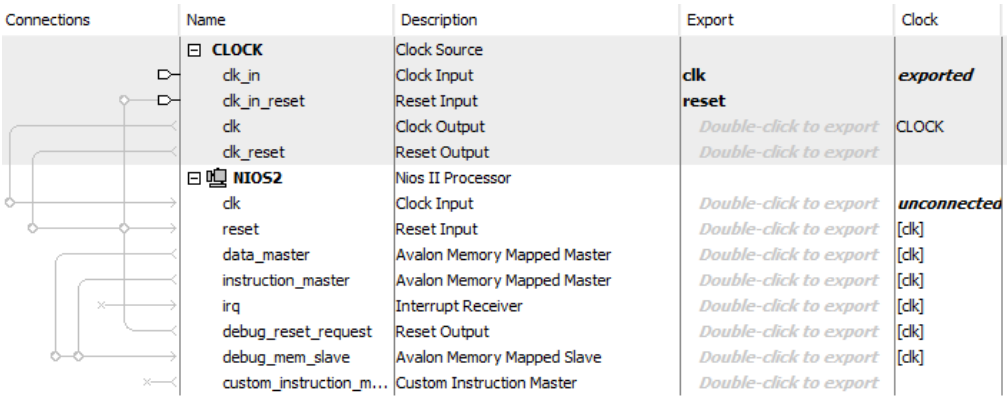
- Clock Source
- NIOS II processor (έκδοση e)
- On-Chip Memory (RAM μεγέθους 40K)
- Parallel I/O (PIO) εύρους 17 bit, ορισμένο σαν είσοδος για τα SWITCH
- Parallel I/O (PIO) εύρους 17 bit, ορισμένο σαν έξοδος για τα LED
- System ID Peripheral

Για την αναζήτηση των component χρησιμοποιούμε το Παράθυρο Αναζήτησης “IP Catalog”. Για παράδειγμα εάν θέλουμε να αναζητήσουμε το component του επεξεργαστή της πλακέτας πληκτρολογούμε τη λέξη “nios” και μας εμφανίζει μία λίστα με τις επιλογές που έχουμε (Εικόνα 3.13).



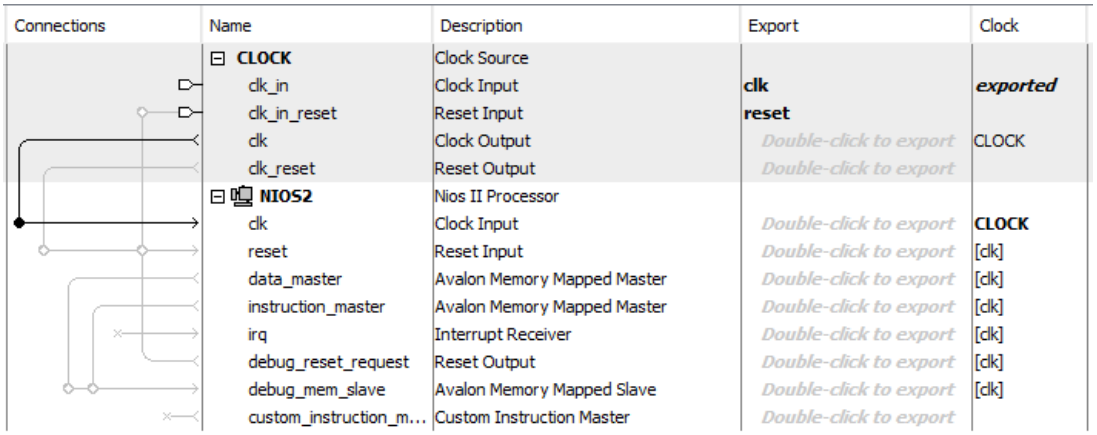
Εικόνα 3.13 Αναζήτηση component στον IP Catalog

Για να προσθέσουμε το component που μας ενδιαφέρει στο project μας κάνουμε διπλό κλικ επάνω του. Αφού αναζητήσουμε όλα τα component που μας ενδιαφέρουν και τα προσθέσουμε στο σύστημά μας πρέπει να τα συνδέσουμε καταλλήλως μεταξύ τους. Έστω ότι έχουμε αναζητήσει και προσθέσει ένα ρολόι και έναν επεξεργαστή Nios II στο σύστημά μας (Εικόνα 3.14α):



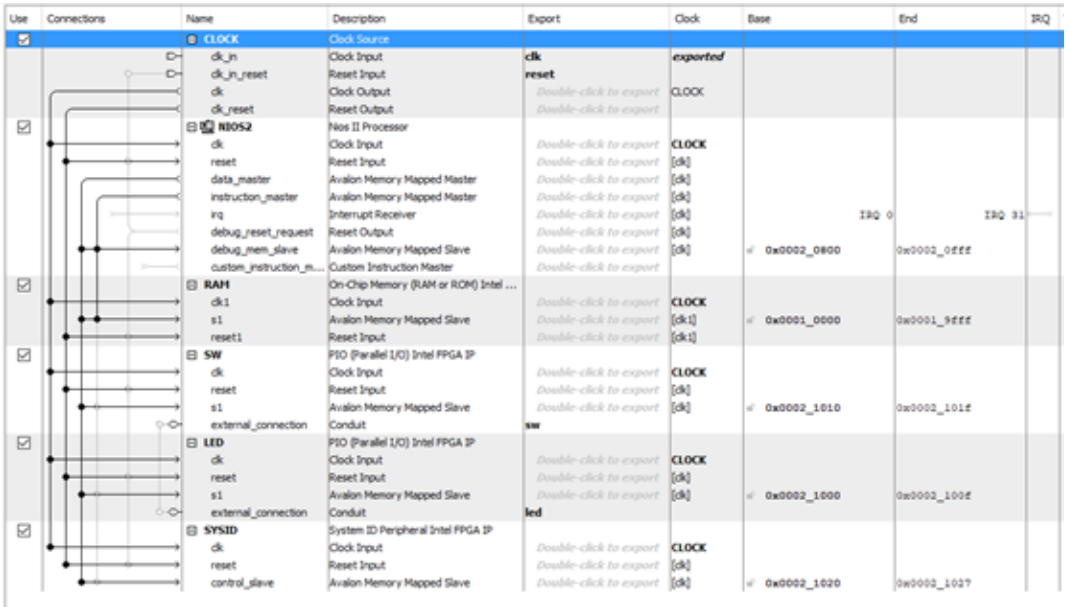
Εικόνα 3.14α Παράδειγμα μη συνδεδεμένης διάταξης ενός συστήματος

Παρατηρούμε στα αριστερά ποιές είναι οι πιθανές συνδεσμολογίες μεταξύ των δύο αυτών στοιχείων. Για την επιτυχή σύνδεση μεταξύ τους κλικάρουμε επάνω στις γραμμές με τους λευκούς κύκλους που βρίσκονται κάτω από τη στήλη “Connections”. Για παράδειγμα μετά το κλικ στον κύκλο επάνω στην γραμμή σύνδεσης του ρολογιού με τον επεξεργαστή, θα πρεπει να “μαυρίσει” ολόκληρη η γραμμή από την έξοδο του πρώτου στοιχείου εώς την είσοδο του δευτερου (Εικόνα 3.14β):



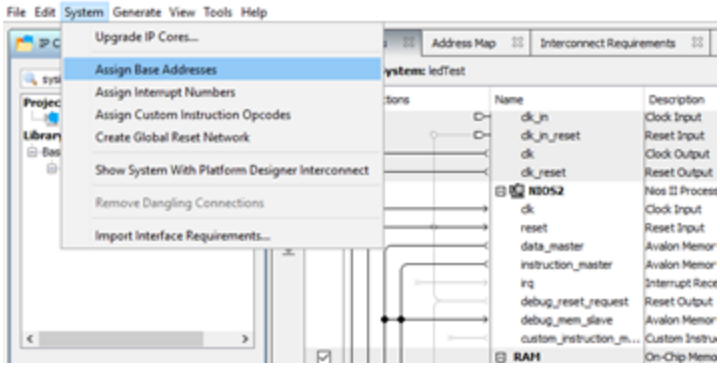
Εικόνα 3.14β Παράδειγμα συνδεδεμένης διάταξης ενός συστήματος

Αντίστοιχα, συνδέουμε και τις υπόλοιπες εισόδους και εξόδους των στοιχείων. Το τελικό αποτέλεσμα αφού προσθέσουμε όλα τα components θα πρέπει να είναι αντίστοιχο της Εικόνας 3.15.



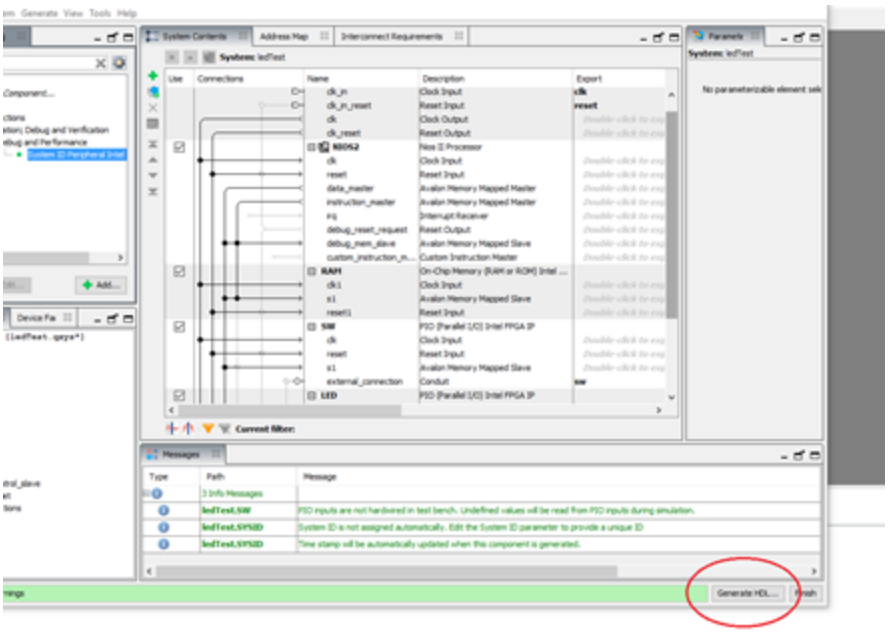
Εικόνα 3.15 Η τελική μορφή της διάταξη συστήματος της Άσκησης 0

Αφού προσθέσουμε ό,τι χρειαζόμαστε για το project μας πρέπει να κάνουμε ανάθεση διευθύνσεων στα components που επιλέξαμε από το χώρο διευθύνσεων του Nios II. Για να το πετύχουμε αυτό, από τη γραμμή μενού πατάμε “System->Assign Base Addresses” (Εικόνα 3.16).



Εικόνα 3.16 Ανάθεση διευθύνσεων στα components του συστήματος

Όταν τοποθετήσουμε και συνδέσουμε όλα τα τμήματα του συστήματός μας πατάμε το κουμπί “Generate HDL” (Εικόνα 3.17) και παράγουμε τα σχετικά αρχεία HDL. Τα αρχεία αυτά παράγονται στο path που δηλώσαμε κατά τη δημιουργία του περιβάλλοντός μας.



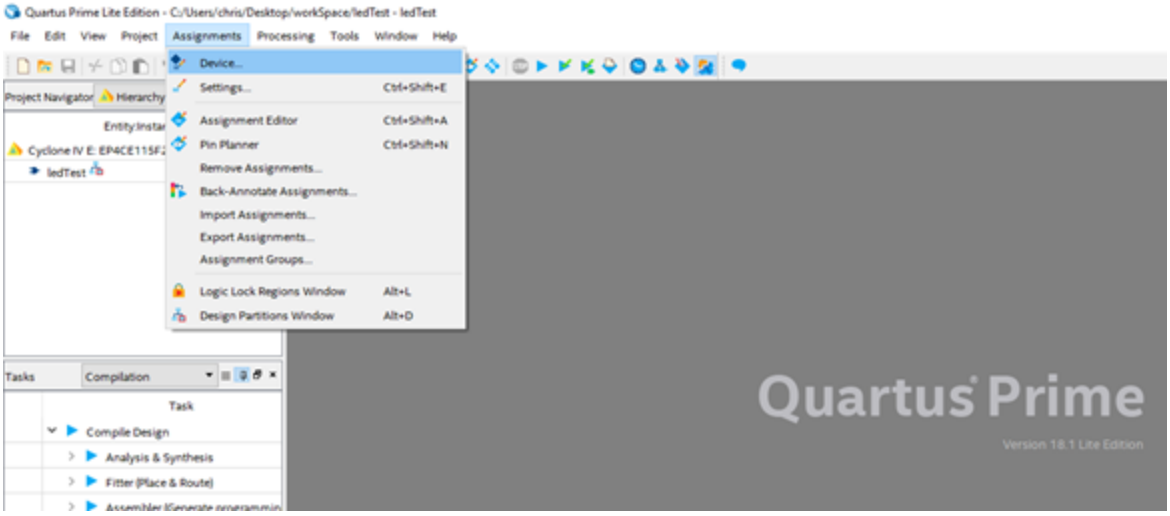
Εικόνα 3.17 Παραγωγή HDL κώδικα

Τέλος, κλείνουμε τον Platform Designer πατώντας “Finish” και επανερχόμαστε στο αρχικό παράθυρο του Quartus Prime.

3.2.3 Ορισμός ακροδεκτών (pin assignments)

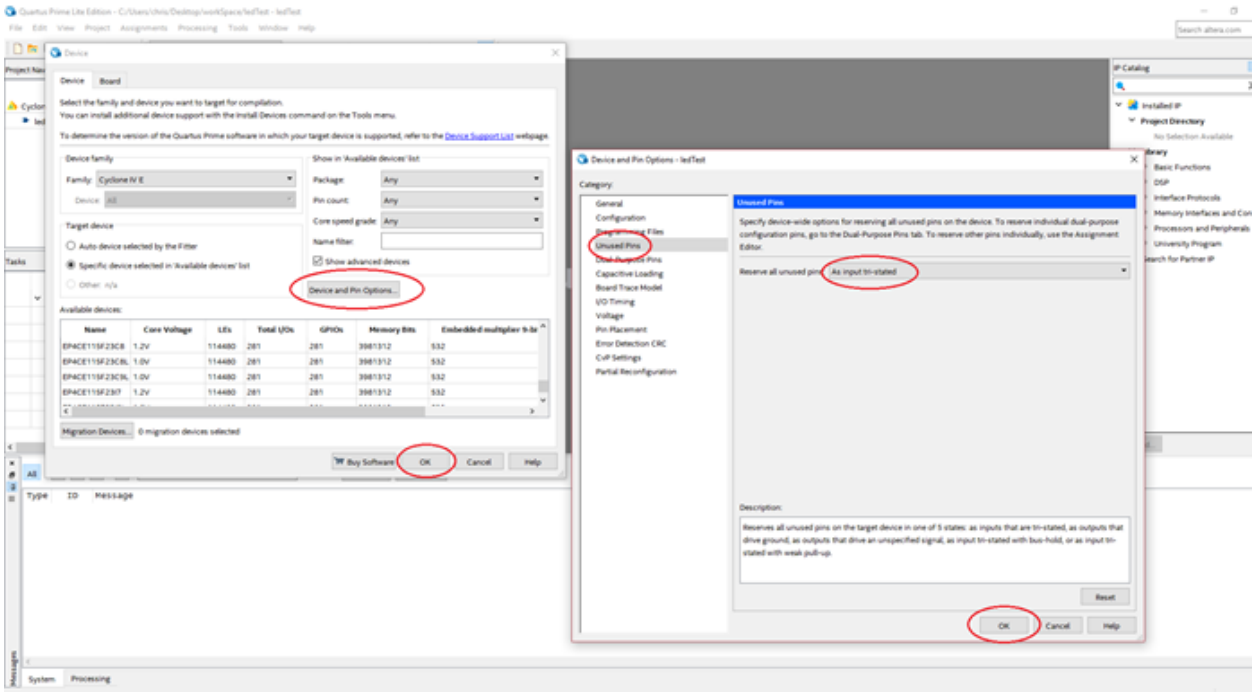
Το επόμενο βήμα είναι να αντιστοιχίσουμε τις εισόδους και τις εξόδους του συστήματός μας με συγκεκριμένους ακροδέκτες της συσκευής που πρόκειται να προγραμματίσουμε. Έχουμε ήδη αναφέρει στην προηγούμενη ενότητα ότι η συσκευή μας είναι ένα FPGA που ανήκει στην οικογένεια Cyclone IV, της εταιρίας Intel, και συγκεκριμένα το EP4CE115F29C7N (βλέπε Εικόνα 3.9). Προκειμένου να ορίσουμε σε ποιους ακροδέκτες του τσιπ θα συνδεθούν οι είσοδοι και οι έξοδοι του συστήματος που σχεδιάσαμε κάνουμε το εξής:

- Από τη γραμμή μενού επιλέγουμε “Assignments” και έπειτα πατάμε πάνω στην επιλογή “Device” (Εικόνα 3.18). Το παράθυρο που θα ανοίξει μας είναι γνώριμο. Είναι αυτό για το οποίο αναφέραμε ότι θα επανέλθουμε στην προηγούμενη ενότητα.



Εικόνα 3.18 Άνοιγμα πλοηγού ρυθμίσεων πλακέτας

- Περίπου στη μέση του παραθύρου, υπάρχει ένα νέο κουμπί, το “Device and Pin Options…”. Αφού το πατήσουμε θα μας ανοίξει ένα νέο παράθυρο στο οποίο θα πρέπει να αναζητήσουμε από το πλοηγό στα αριστερά του παραθύρου την επιλογή “Unused Pins”. Στη καρτέλα που θα ανοίξει στα δεξιά υπάρχει μία “drop-down” λίστα από την οποία διαλέγουμε την επιλογή “As input tri-stated”. Πατάμε “OK” και ξανά “OK” για να κλείσουμε τα παράθυρα που ανοίξαμε και επανερχόμαστε στο αρχικό (Εικόνα 3.19).



Εικόνα 3.19 Ρυθμίσεις ακροδεκτών πλακέτας

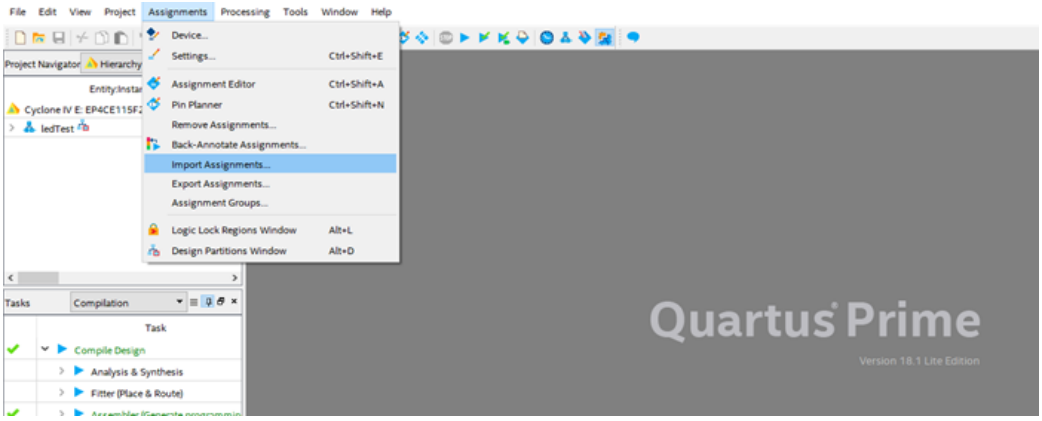
- Στα εγχειρίδια χρήσης που συνοδεύουν την πλακέτα περιέχεται ένα .csv αρχείο με όνομα “DE2_115_pin_assignments”, το οποίο θα πρέπει να αντιγράψουμε αρχικά στο path που δηλώσαμε κατά τη δημιουργία του project (βλέπε Εικόνα 3.6), και στη συνέχεια να το τροποποιήσουμε.

- Αφού το αντιγράψουμε στο path του project μας προαιρετικά αλλάζουμε το όνομα του σε κάτι που να αντιστοιχεί με το project. Για παράδειγμα στο δικό μας σενάριο μια καλή επιλογή είναι το “led_test_pin_assignments.csv” έτσι ώστε να περιλαμβάνει και το όνομα του project.

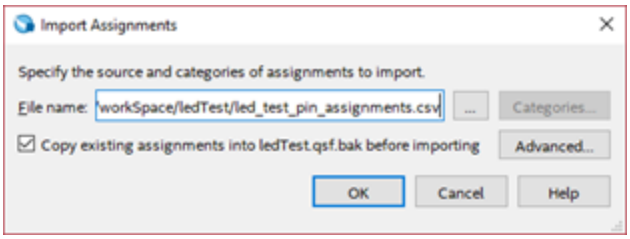
- Υπάρχουν αρκετά editors και document tools με τα οποία μπορούμε να επεξεργαστούμε .csv τύπους αρχείων. Ανοίγουμε με έναν text editor της επιλογής μας το αρχείο και προσεκτικά αρχίζουμε να παρατηρούμε ποιους ακροδέκτες θα κρατήσουμε και ποιους όχι. Στο συγκεκριμένο αρχείο υπάρχουν όλοι οι ακροδέκτες της πλακέτας. Εμείς για το σύστημα που σχεδιάσαμε θα χρειαστούμε ένα ρολόι, έναν διακόπτη για το reset, 17 εξόδους για τα LED καθώς και 17 εισόδους από τα SWITCHES. Τα περιεχόμενα του .csv αρχείου της άσκησης αυτής είναι τα ακόλουθα:

```
# Clock
clk_clk,Input,PIN_Y2,2,B2_N0,3.3-V LVTTL,
# Reset
reset_reset_n,Input,PIN_Y23,5,B5_N2,2.5 V,
# LEDS
led_export[16],Output,PIN_G16,7,B7_N2,2.5 V,
led_export[15],Output,PIN_G15,7,B7_N2,2.5 V,
led_export[14],Output,PIN_F15,7,B7_N2,2.5 V,
led_export[13],Output,PIN_H17,7,B7_N2,2.5 V,
led_export[12],Output,PIN_J16,7,B7_N2,2.5 V,
led_export[11],Output,PIN_H16,7,B7_N2,2.5 V,
led_export[10],Output,PIN_J15,7,B7_N2,2.5 V,
led_export[9],Output,PIN_G17,7,B7_N1,2.5 V,
led_export[8],Output,PIN_J17,7,B7_N2,2.5 V,
led_export[7],Output,PIN_H19,7,B7_N2,2.5 V,
led_export[6],Output,PIN_J19,7,B7_N2,2.5 V,
led_export[5],Output,PIN_E18,7,B7_N1,2.5 V,
led_export[4],Output,PIN_F18,7,B7_N1,2.5 V,
led_export[3],Output,PIN_F21,7,B7_N0,2.5 V,
led_export[2],Output,PIN_E19,7,B7_N0,2.5 V,
led_export[1],Output,PIN_F19,7,B7_N0,2.5 V,
led_export[0],Output,PIN_G19,7,B7_N2,2.5 V,
# SWITCHES
sw_export[16],Input,PIN_Y24,5,B5_N2,2.5 V,
sw_export[15],Input,PIN_AA22,5,B5_N2,2.5 V,
sw_export[14],Input,PIN_AA23,5,B5_N2,2.5 V,
sw_export[13],Input,PIN_AA24,5,B5_N2,2.5 V,
sw_export[12],Input,PIN_AB23,5,B5_N2,2.5 V,
sw_export[11],Input,PIN_AB24,5,B5_N2,2.5 V,
sw_export[10],Input,PIN_AC24,5,B5_N2,2.5 V,
sw_export[9],Input,PIN_AB25,5,B5_N1,2.5 V,
sw_export[8],Input,PIN_AC25,5,B5_N2,2.5 V,
sw_export[7],Input,PIN_AB26,5,B5_N1,2.5 V,
sw_export[6],Input,PIN_AD26,5,B5_N2,2.5 V,
sw_export[5],Input,PIN_AC26,5,B5_N2,2.5 V,
sw_export[4],Input,PIN_AB27,5,B5_N1,2.5 V,
sw_export[3],Input,PIN_AD27,5,B5_N2,2.5 V,
sw_export[2],Input,PIN_AC27,5,B5_N2,2.5 V,
sw_export[1],Input,PIN_AC28,5,B5_N2,2.5 V,
sw_export[0],Input,PIN_AB28,5,B5_N1,2.5 V,
```

➢ Αφού τροποποιήσουμε κατάλληλα το .csv αρχείο, θα πρέπει και να το εισάγουμε στο project μας. Αυτό μπορούμε να το κάνουμε πατώντας το Assignments στη γραμμή του μενού, και έπειτα επιλέγοντας “Import Assignments”. Στο παράθυρο που θα ανοίξει επιλέγουμε μέσω του πλοηγού το αρχείο led_test_pin_assignments.csv από το φάκελο του project μας και πατάμε OK (Εικόνα 3.20 και Εικόνα 3.21).

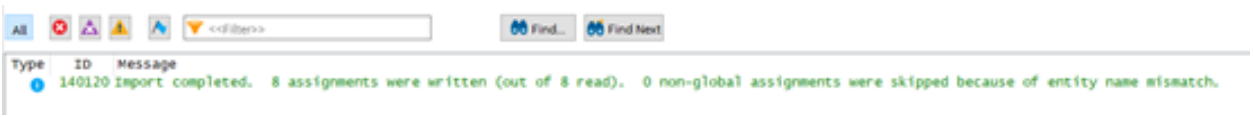


Εικόνα 3.20 Αντιστοίχιση ακροδεκτών



Εικόνα 3.21 Εισαγωγή αρχείου αντιστοίχισης

➢ Στο αρχικό παράθυρο θα πρέπει να μας εμφανίσει το ακόλουθο μήνυμα επιτυχίας στη καρτέλα “System” (Εικόνα 3.22). Προσέχουμε τα ονόματα που θα δώσουμε στους ακροδέκτες στο .csv αρχείο καθώς θα πρέπει να αντιστοιχούν με τα ονόματα τα οποία έχει δώσει ο Platform Designer κατά την παραγωγή του κώδικα του συστήματος στο αρχείο που ειδοποιεί όλα τα επιμέρους τμήματα ledTest.v, το οποίο βρίσκεται στη διαδρομή “./ledTest/ledTest/synthesis/”. Σε διαφορετική περίπτωση θα εμφανιστεί μήνυμα λάθους κατά την αντιστοίχιση των ακροδεκτών.

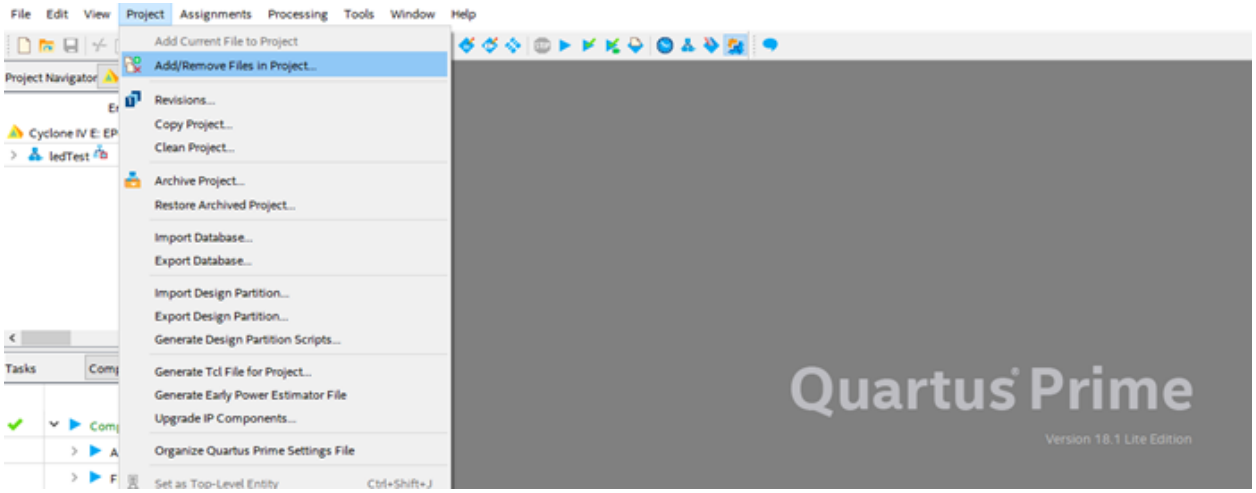


Εικόνα 3.22 Μήνυμα επιτυχημένης αντιστοίχισης ακροδεκτών

➢ Πλέον είμαστε έτοιμοι να εισάγουμε στο project μας τα αρχεία που παρήγαγε ο Platform Designer στην προηγούμενη ενότητα. Για να το πετύχουμε αυτό επιλέγουμε από τη γραμμή μενού “Project->Add/Remove Files in Project…”.

3.2.4 Διαμόρφωση του κυκλώματος

Αφού ολοκληρωθεί ο ορισμός των ακροδεκτών και πριν πραγματοποιηθεί η διαμόρφωση του FPGA, θα πρέπει να γίνει η Μεταγλώττιση (Compilation) ώστε να δημιουργηθεί το αρχείο .sof, το οποίο περιέχει τα bit διαμόρφωσης του FPGA. Πρέπει λοιπόν να εισάγουμε στο project μας όλα τα αρχεία που παρήγαγε ο Platform Designer. Για να το πετύχουμε αυτό επιλέγουμε από τη γραμμή μενού “Project->Add/Remove Files in Project…” (Εικόνα 3.23).



Εικόνα 3.23 Εισαγωγή των αρχείων διάταξης στο project

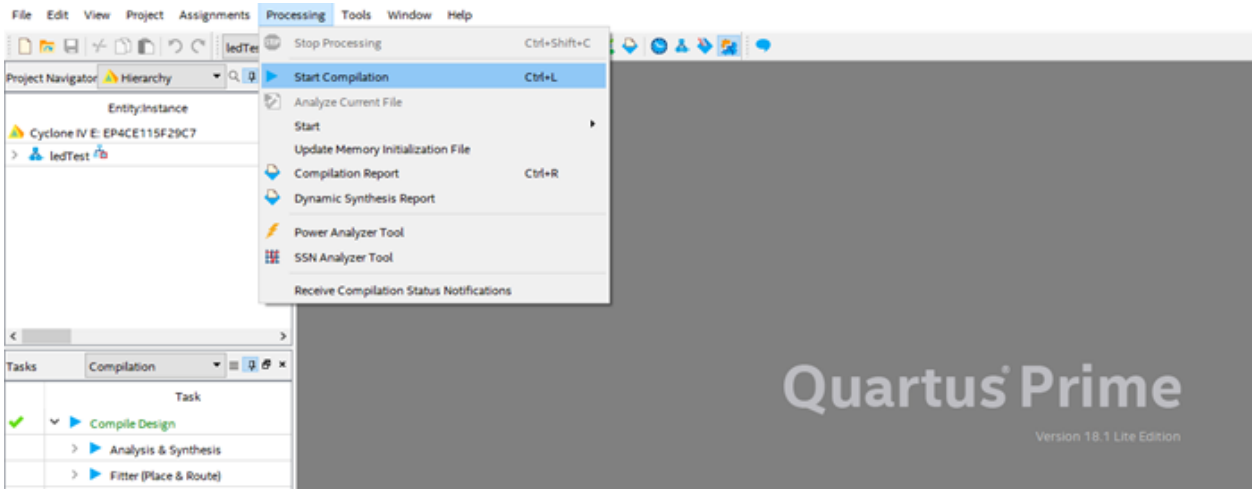
- Επιλέγοντας στα αριστερά την κατηγορία “Files” μπορούμε να πλοηγηθούμε στο path του project μας και επιλέγουμε να εισάγουμε από τη διαδρομή “./ledTest/ledTest/synthesis/” το αρχείο ledTest.v και από τη διαδρομή “./ledTest/ledTest/synthesis/submodules/” όλα τα υπάρχοντα σε αυτή αρχεία. Έπειτα πατάμε “OK” και επανερχόμαστε στο αρχικό παράθυρο, και πλέον είμαστε έτοιμοι για τη σύνθεση του υλικού.

- Επίσης, μπορούμε να δώσουμε και κάποιους σχεδιαστικούς περιορισμούς, ώστε η σύνθεση να γίνει με τρόπο που να τους ικανοποιεί. Οι σχεδιαστικοί περιορισμοί εισάγονται με το “File->New->Synopsis Design Constraints File”. Στο παράθυρο που ανοίγει τοποθετούμε τις παρακάτω γραμμές:

    - create_clock -period 20.000 -name clk_clk
    - derive_clocks -period 20.000
    - derive_clock_uncertainty

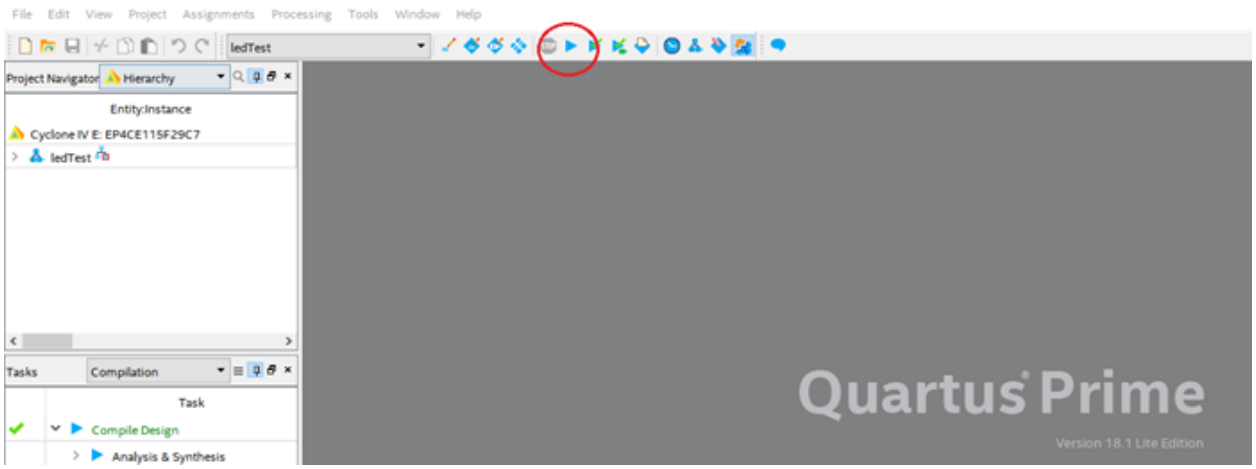
    όπου ορίζεται ότι το ρολόι θα πρέπει να έχει ελάχιστη περίοδο 50 MHz, και το αποθηκεύουμε στον κατάλογο του project με κατάληξη .sdc. Προσέχουμε το όνομα που θα δώσουμε στο ρολόι καθώς θα πρέπει να αντιστοιχεί με το όνομα που έχει δώσει ο Platform Designer κατά την παραγωγή του κώδικα του συστήματος στο αρχείο <ledTest>.v. Σε διαφορετική περίπτωση θα λάβουμε μήνυμα λάθους κατά τη μεταγλώττιση του συστήματος.

➢ Η μεταγλώττιση ξεκινά είτε επιλέγοντας από τη γραμμή του μενού “Processing” και έπειτα πατώντας την επιλογή “Start Compilation” (Εικόνα 3.24α),



Εικόνα 3.24α Μεταγλώττιση διάταξης από τη γραμμή μενού

είτε πατώντας απευθείας το αντίστοιχο κουμπί στη γραμμή εργαλείων. Η συντόμευση απεικονίζεται με το πράσινο εικονίδιο που έχει το σύμβολο του “play” (Εικόνα 3.24β). Επίσης, μπορούμε να ξεκινήσουμε τη μεταγλώττιση και μέσω του συνδυασμού των πλήκτρων του πληκτρολογίου “Ctrl+L”.



Εικόνα 3.24β Μεταγλώττιση του συστήματος με χρήση κουμπιού συντόμευσης

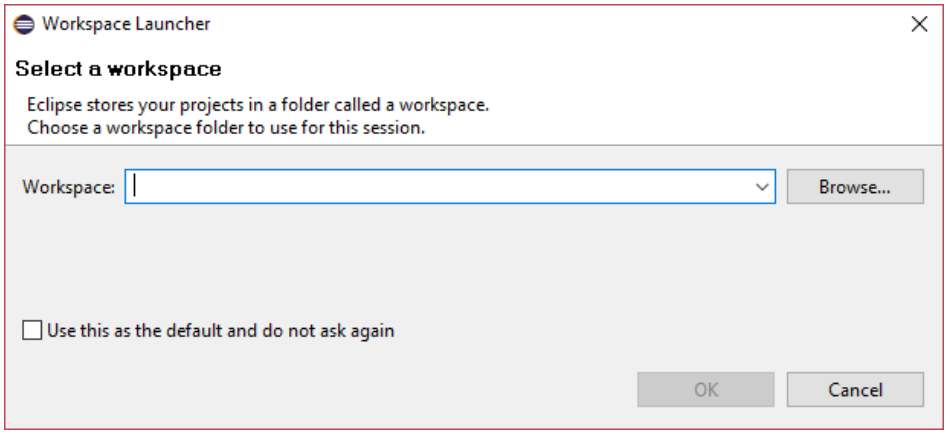
➢ Μόλις ολοκληρωθεί η σύνθεση του υλικού μας είμαστε έτοιμοι να αναπτύξουμε το λογισμικό που θα χρησιμοποιήσουμε.

Υπενθυμίζουμε ότι η Μεταγλώττιση περιλαμβάνει τις διαδικασίες της Ανάλυσης και Σύνθεσης, της Τοποθέτησης και Διασύνδεσης, της Χρονικής Ανάλυσης και της παραγωγής των αρχείων προγραμματισμού του FPGA. Μετά την επιτυχή ολοκλήρωση της Μεταγλώττισης μπορεί να γίνει η διαμόρφωση του FPGA με χρήση του εργαλείου Προγραμματιστής (Programmer), το οποίο επιλέγουμε να ανοίξουμε μέσα από το περιβάλλον ανάπτυξης λογισμικού του Nios II Software Build Tools (SBT) για το Eclipse, το οποίο περιγράφεται αναλυτικά στην επόμενη ενότητα.

3.3 Ανάπτυξη λογισμικού στο Nios II Software Build Tools (SBT)

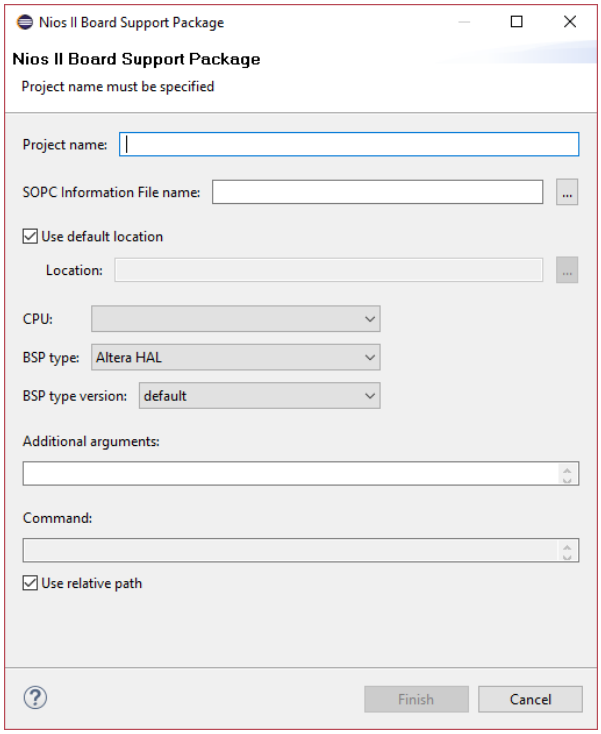
Μετά από την υλοποίηση του συστήματος στο Quartus Prime, για να προγραμματιστεί ο επεξεργαστής Nios II είναι απαραίτητη η συγγραφή κώδικα στο Nios II Embedded Design Suite (Nios II EDS) σε C ή σε C++. Η Intel δίνει την δυνατότητα αυτόματης παραγωγής μίας προσαρμοσμένης βιβλιοθήκης υποστήριξης της αναπτυξιακής πλακέτας (Board Support Package - BSP), για να επιταχύνει την ανάπτυξη του λογισμικού.

➢ Αφού ανοίξουμε το Nios II Software Build Tools (SBT) for Eclipse, το περιβάλλον μας ζητά να διαλέξουμε ένα περιβάλλον εργασίας. Προτείνεται να δημιουργείται ένας υποφάκελος στην τοποθεσία του κύριου έργου (Εικόνα 3.25):



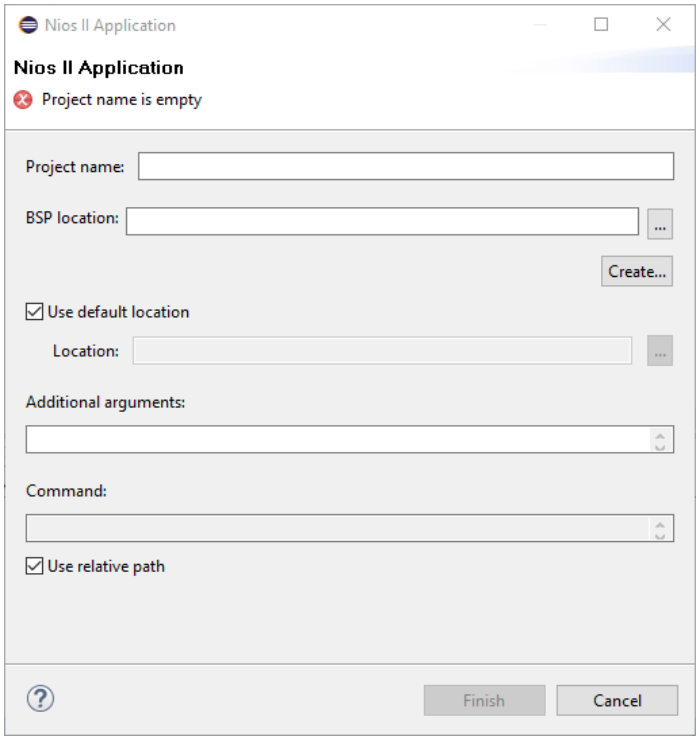
Εικόνα 3.25 Επιλογή καταλόγου εργασίας της εφαρμογής

➢ Μόλις ολοκληρωθεί το “άνοιγμα” του εργαλείου, είναι έτοιμο για την ανάπτυξη του λογισμικού μας. Είτε μέσω της γραμμής μενού πατώντας “File->New-> Nios II Board Support Package”, είτε κάνοντας δεξί κλικ στον κενό χώρο της καρτέλας “Project Explorer”, η οποία βρίσκεται στα αριστερά του κεντρικού παραθύρου, και επιλέγοντας “New->Nios II Board Support Package”, ανοίγουμε το πλοηγό για τη δημιουργία της βιβλιοθήκης υποστήριξης (Εικόνα 3.26).



Εικόνα 3.26 Επιλογή συστήματος και δημιουργία BSP

➢ Αφού συμπληρώσουμε όλα τα απαραίτητα στοιχεία που ζητούνται, πατάμε το κουμπί “Finish”. Επειτα, ακολουθούμε μία από τις διαδρομές που αναφέρθηκαν στο προηγούμενο βήμα, και επιλέγουμε “Nios II Application”. Στο παράθυρο που ανοίγει προσθέτουμε όλες τις απαραίτητες πληροφορίες που μας ζητούνται και πατάμε “Finish” (Εικόνα 3.27).



Εικόνα 3.27 Επιλογή BSP και δημιουργία έργου (project) λογισμικού για το σύστημά μας

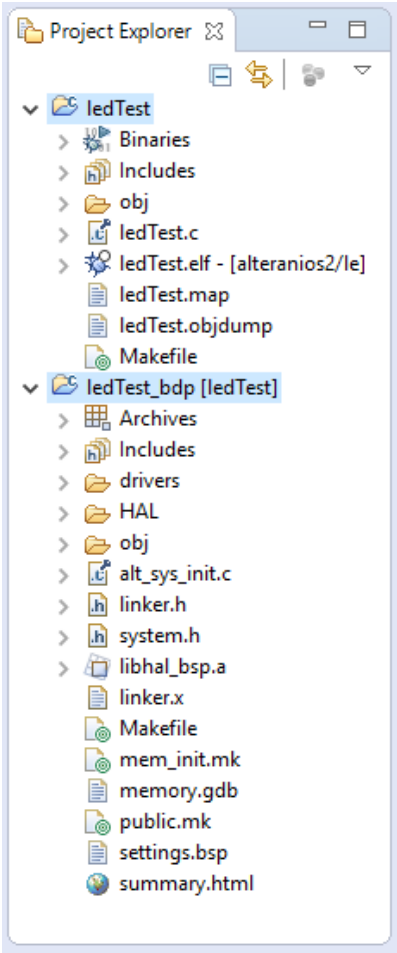
➢ Στη συνέχεια δημιουργούμε ένα αρχείο πηγαίου κώδικα στο project που μόλις δημιουργήσαμε, και προσθέτουμε τον κώδικα της εφαρμογής μας. Δίνεται ο παρακάτω κώδικας C (Εικόνα 3.28) που υλοποιεί το ζητούμενο της παρούσας εισαγωγικής Άσκησης 0. Ο κώδικας αυτός παρέχεται μαζί με το εγχειρίδιο χρήσης της αναπτυξιακής πλακέτας:

```
1  #include "system.h"
2
3  int main(void){
4      //initialization
5      int *leds     = (int *)LED_BASE;
6      volatile int *switches = (int *)SW_BASE;
7      while(1){//infinite loop
8          *leds = *switches;
9      }
10     return 0;
11 }
```

Εικόνα 3.28 Παράδειγμα υλοποίησης ζητούμενου Άσκησης 0

- Έπειτα επιλέγουμε και τα δύο project που δημιουργήσαμε (BSP και Application), και από τη γραμμή μενού πατάμε “Project->Build All”, είτε κάνουμε δεξί κλικ πάνω τους και πατάμε “Build”.

- Στην Εικόνα 3.29 φαίνονται τα περιεχόμενα της καρτέλας “Project Explorer” μετά την επιτυχημένη μεταγλώττιση του προγράμματος.



Εικόνα 3.29 Περιεχόμενα εφαρμογής

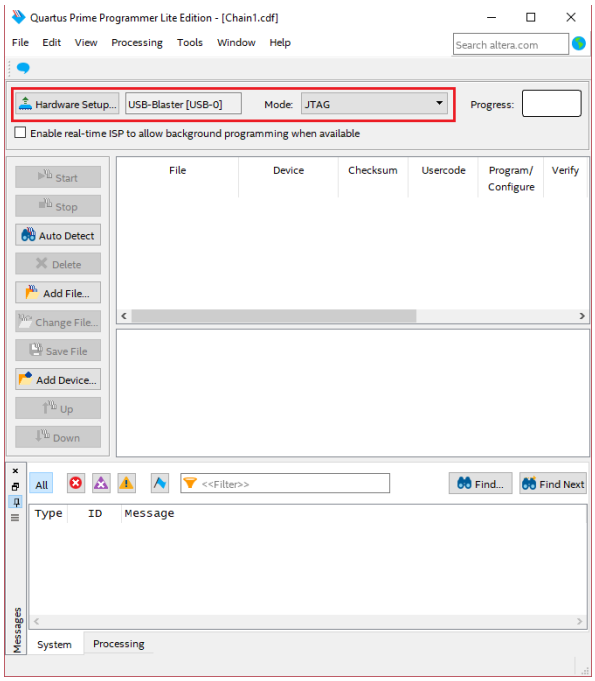
- ΔΕΝ κλείνουμε το παράθυρο του εργαλείου και προχωράμε στην επόμενη, και τελευταία, ενότητα, στην οποία θα μεταφορτώσουμε υλικό και λογισμικό στην πλακέτα, και θα παρακολουθήσουμε το αποτελέσματα αυτών, ως συνολική λειτουργία, χρησιμοποιώντας την αναπτυξιακή πλακέτα DE2-115.

3.4 Έλεγχος σωστής λειτουργίας της πλακέτας με χρήση των LED

Με το πέρας της επιτυχημένης μεταγλώττισης του προγράμματος είμαστε έτοιμοι να μεταφορτώσουμε υλικό και λογισμικό στην αναπτυξιακή πλακέτα. Όπως ορίστηκε στο .csv αρχείο του Quartus, για reset (επαναφορά) του συστήματος θα χρησιμοποιήσουμε το τελευταίο διακόπτη της πλακέτας (SW17).

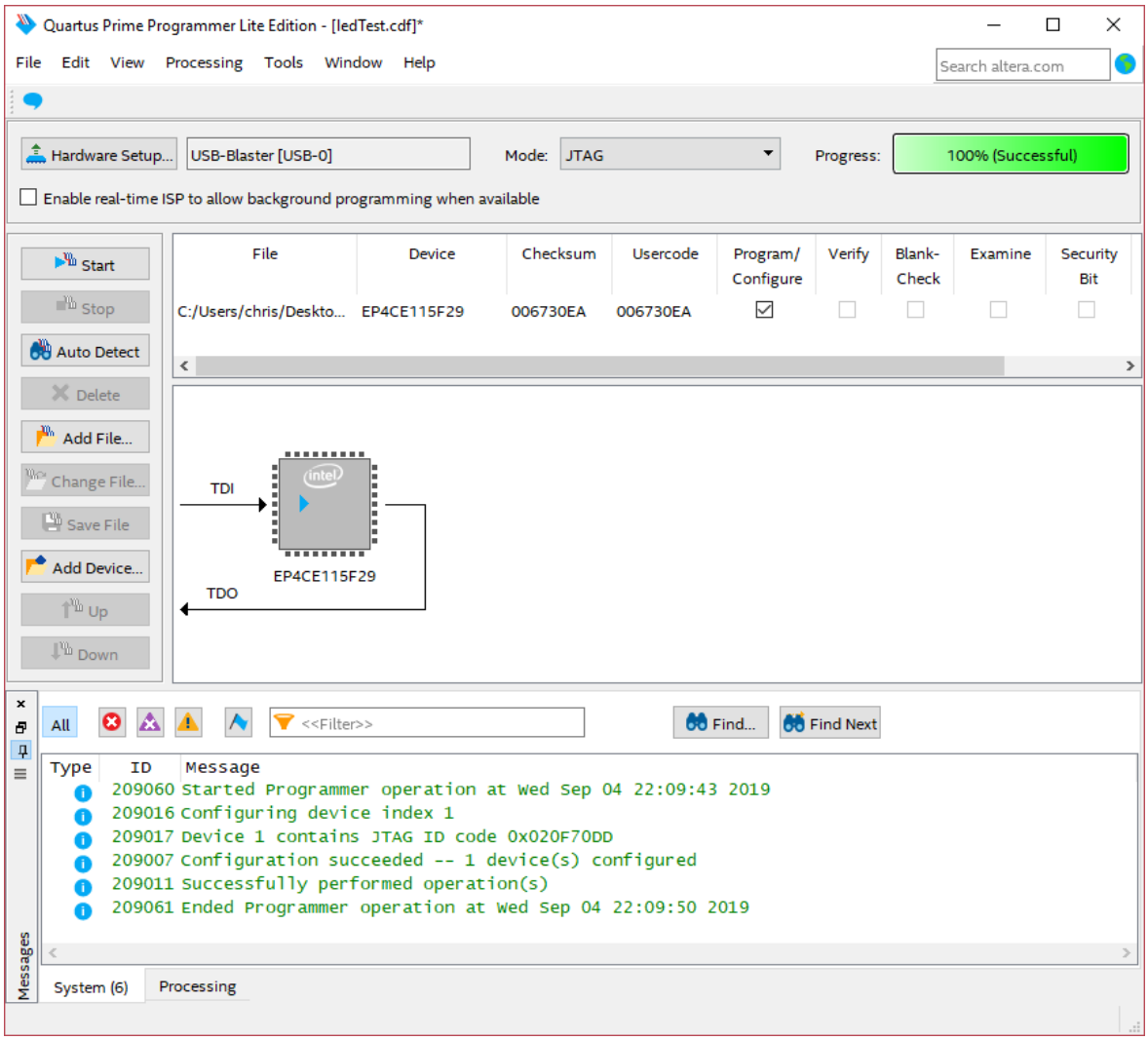
Θα πρέπει να δοθεί προσοχή σε αυτό το σημείο καθώς σε ένα σύστημα που υλοποιείται με τον Platform Designer, το reset είναι active low (ενεργοποιείται με την τιμή 0). Το τελευταίο σχετίζεται με τη σωστή θέση στην οποία πρέπει να βρίσκεται ο διακόπτης κατά τη λειτουργία της πλακέτας (θέση λογικού 1). Σε περίπτωση που βρίσκεται στη θέση λογικού 0 το σύστημά μας δεν θα ανταποκρίνεται στο “κατέβασμα” του λογισμικού μας στη πλακέτα, καθώς και ότι έχουμε ήδη μεταφορτώσει στη πλακέτα δεν θα “τρέχει”, αφού ο επεξεργαστής θα βρίσκεται σε κατάσταση reset.

- Ενεργοποιούμε την αναπτυξιακή πλακέτα DE2-115, και τη συνδεουμε στον υπολογιστή με το καλώδιο USB.
- Το επόμενο βήμα είναι να ξεκινήσουμε το εργαλείο Quartus Prime Programmer, το οποίο θα μας βοηθήσει να μεταφορτώσουμε το υλικό μας στην αναπτυξιακή πλακέτα. Επιλέγουμε από τη γραμμή μενού “Nios II->Quartus Prime Programmer”. Αφού ανοίξει το παράθυρο του εργαλείου, επιβεβαιώνουμε ότι στην επιλογή “Hardware Setup” έχει προεπιλεγεί το “USB-Blaster [USB-#]” και στο “Mode” έχει επιλεχθεί το “JTAG”, όπως φαίνεται και στην Εικόνα 3.30. Σε διαφορετική περίπτωση (αν η επιλογή είναι “No Hardware”) ελέγχουμε τη σύνδεση της πλακέτας με τον υπολογιστή, όπως επίσης και το αν έχει τροφοδοσία η πλακέτα.



Εικόνα 3.30 Quartus Prime Programmer

➢ Από τη γραμμή μενού πατάμε “File->Open”, και στο αναδυόμενο παράθυρο πλοηγούμαστε στον φάκελο του project μας. Εκεί έχει δημιουργηθεί από το Quartus ο φάκελος “output_files”, ο οποίος περιέχει ένα αρχείο με την κατάληξη .sof, το οποίο προέκυψε στο τέλος της Μεταγλώττισης, και περιέχει όλα τα bit προγραμματισμού του FPGA. Το επιλέγουμε και πατάμε “Open”. Έπειτα, στο αρχικό παράθυρο του εργαλείου, από τη γραμμή επιλογών στα αριστερά πατάμε “Start”. Στο δεξί μέρος του παραθύρου υπάρχει η μπάρα προόδου “Progress” η οποία μας ενημερώνει για το αν η μεταφόρτωση του υλικού ήταν επιτυχής ή όχι.

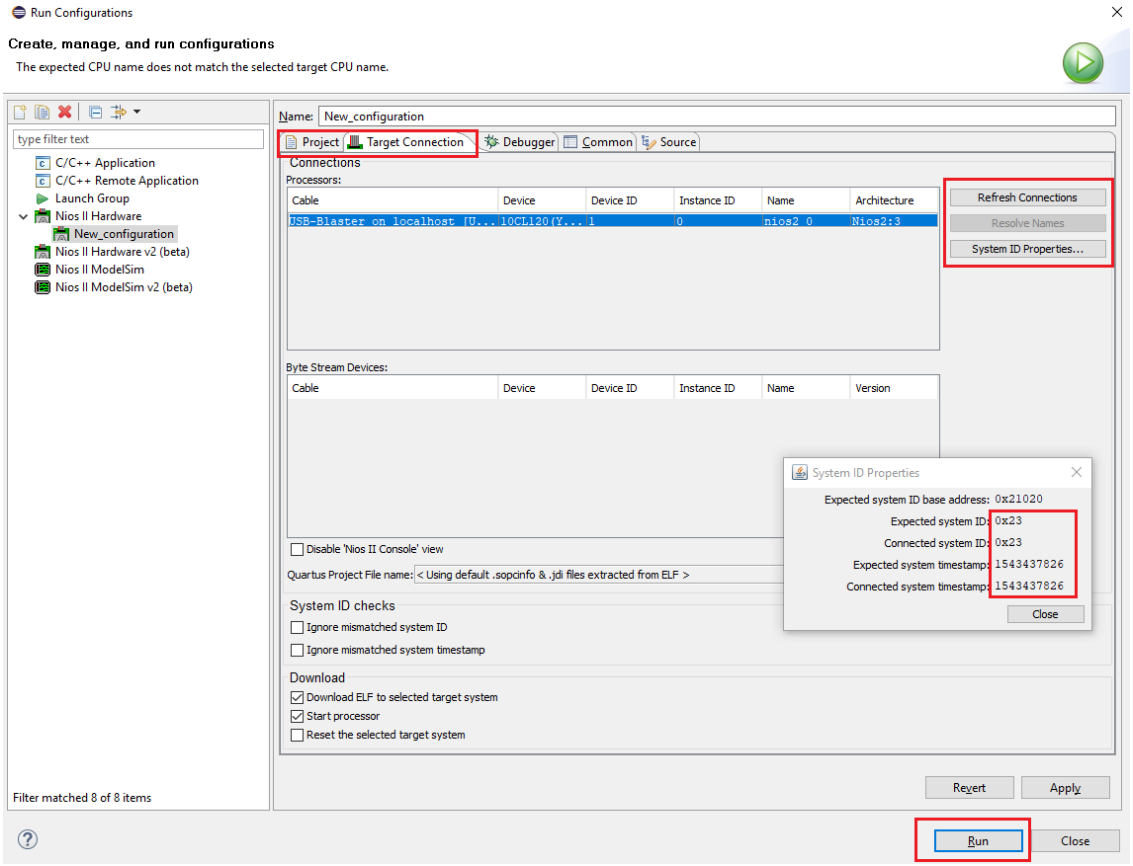


Εικόνα 3.31 Επιτυχημένη μεταφόρτωση υλικού στην αναπτυξιακή πλακέτα

➢ Με την επιτυχημένη μεταφόρτωση του υλικού μας στο FPGA (Εικόνα 3.31) παρατηρούμε ότι πλέον έχει σταματήσει η λειτουργίας επίδειξης, η οποία ξεκινά κάθε φορά που η πλακέτα συνδεθεί στο ρεύμα. Εάν διακοπεί η τροφοδοσία τότε χάνεται και η διαμόρφωση του FPGA, και επανέρχεται η αρχική λειτουργία επίδειξης.

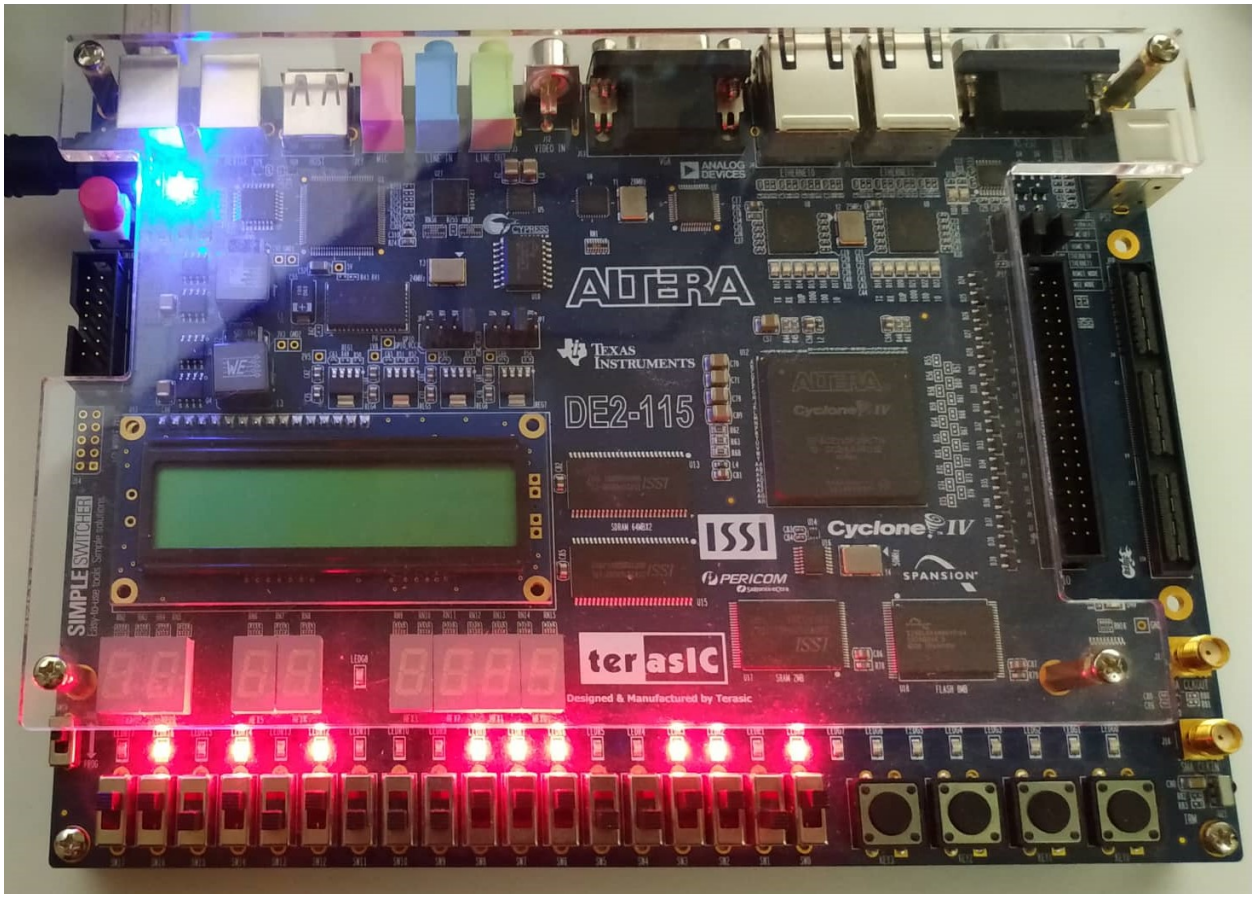
➢ Επιστρέφουμε στο εργαλείο Eclipse Nios II EDS, και από τη γραμμή μενού επιλέγουμε “Run->Run Configurations”. Στο παράθυρο που εμφανίζεται (Εικόνα 3.32), ελέγχουμε στην πρώτη καρτέλα “Project” ότι έχει επιλεγεί το “Project name” με το οποίο έχουμε ονομάσει το έργο μας, και προχωράμε στη δεύτερη καρτέλα “Target Connection”.

➢ Στο δεξί μέρος του παραθύρου πατάμε “Refresh Connections”, και αφού εμφανιστεί η σύνδεσή μας με τη συσκευή στη λίστα πατάμε “System ID Properties...”. Επιβεβαιώνουμε ότι τα “Expected” και “Connected”, “System ID” και “System timestamp” έχουν τις ίδιες τιμές και πατάμε “Close”, και τέλος “Run”.



Εικόνα 3.32 Ενεργοποίηση σύνδεσης Nios II EDS με την πλακέτα

➢ Μόλις ολοκληρωθεί και η μεταφόρτωση του λογισμικού, είμαστε έτοιμοι να δούμε τα αποτελέσματα στην πλακέτα. Σύμφωνα και με το ζητούμενο της εισαγωγικής Άσκησης 0, θα πρέπει η κατάσταση κάθε διακόπτη να αντικατοπτρίζεται στο αντίστοιχο LED της αναπτυξιακής πλακέτας, όπως φαίνεται στην Εικόνα 3.33.



Εικόνα 3.33 Ενδεικτική λειτουργία υλικού και λογισμικού για την εισαγωγική Άσκηση 0

4. Άσκηση 1 - Ανάπτυξη συστήματος με χρήση του Platform Designer και αποσφαλμάτωση λογισμικού

4.1 Εκφώνηση Άσκησης

Η διαδικασία εντοπισμού σφαλμάτων ορίζεται ως η διαδικασία εύρεσης και επίλυσης προβλημάτων σε ένα πρόγραμμα που εμποδίζει τη σωστή λειτουργία του λογισμικού ή του συστήματος. Η εκμάθηση του τρόπου με τον οποίο μπορεί να εντοπιστούν τα λάθη μίας εφαρμογής είναι ένα βασικό στοιχείο που πρέπει να κατέχει ένας μηχανικός ανάπτυξης υλικού και λογισμικού. Η διαδικασία αυτή αναγκάζει τον προγραμματιστή/σχεδιαστή να:

- Αναλύσει την κατάσταση
- Μάθει τα πάντα σχετικά με τη συμπεριφορά της εκάστοτε εφαρμογής και τα δεδομένα που αυτή διαχειρίζεται
- Εφαρμόσει διορθώσεις με βάση τις πληροφορίες που μαθαίνει

Οι τακτικές αποσφαλμάτωσης μπορούν επίσης να περιλαμβάνουν διαδραστικές μεθόδους εντοπισμού των σφαλμάτων (interactive debugging, trial and error), ανάλυση της ροής ελέγχου (control flow analysis), λειτουργίες δοκιμών και ρυθμίσεων των επιμέρους μονάδων (unit testing), δοκιμές ενσωμάτωσης νέων μεθόδων (integration testing), ανάλυση του αρχείου καταγραφής (logging), παρακολούθηση σε επίπεδο εφαρμογής ή συστήματος (monitoring at the application or in the system level), ελέγχους διαρροών μνήμης (memory leaks) και δημιουργία προφίλ συμπεριφοράς της εφαρμογής (profiling).

Ζητούμενο της Άσκησης 1 είναι να αποσφαλματώσετε τον ακόλουθο κώδικα, ο οποίος αποτελεί επέκταση της εισαγωγικής Άσκησης 0. Σκοπός της παρούσας άσκησης, όπως και στην Άσκηση 0, είναι να χρησιμοποιούνται όλα τα switches της πλακέτας, καθώς και τα LED που τους αντιστοιχούν. Επιπρόσθετα, θα πρέπει να προστεθούν επιπλέον έλεγχοι που να ορίζουν τον τρόπο ανάμματος των LED βάση τη θέση των διακοπτών. Με το κουμπί KEY3 θα πρέπει να επιτρέπεται το άναμμα των LED στα οποία έχει ενεργοποιηθεί ο διακόπτης που τους αντιστοιχεί, ενώ αντιθέτως, με το κουμπί KEY0 θα επιτρέπεται το άναμμα των LED στα οποία ο διακόπτης που τους αντιστοιχεί είναι απενεργοποιημένος. Και στις δύο περιπτώσεις τα LED θα πρέπει να είναι αναμμένα για όσο το κουμπί που αντιστοιχεί σε κάθε λειτουργία παραμένει πατημένο.

Στην Εικόνα 4.1 που ακολουθεί δίνεται ο κώδικας του λογισμικού, ο οποίος περιέχει λογικά λάθη:

```c
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"

// PKeys Status
#define RVselected 2
#define SWselected 1

void initial_message(){
    printf("****************************\n");
    printf("*   Hello from Nios II!    *\n");
    printf("****************************\n");
}// initial_message

int getCommand(alt_u32 pkeys_base){
    return IORD(pkeys_base,0) & 0x01;
}// getCommand

int getSwitches(alt_u32 sw_base){
    return IORD(sw_base, 0) & 0x01f;
}// getSwitches

void ledDisplay(alt_u32 led_base, int disp){
    IOWR(led_base, 0, disp);
}// ledDisplay

int main(){
    // check operation status - print welcome message
    initial_message();
    while(1){// running forever
            command = getCommand(PKEYS_BASE)

            // SWITCHES Display
            while(command == SWselected){
                ledDisplay(LED_BASE, getSwitches(SW_BASE));
            }// while

            // REVERT SWITCHES Display
            while(command == RVselected){
                ledDisplay(LED_BASE, ~getSwitches(SW_BASE));
            }// while

            // IDLE | ERROR
            ledDisplay(LED_BASE, 0xf1);
    }// while
    // unreachable statement
    return 0;
}// main
```

Εικόνα 4.1 Λανθασμένος κώδικας λειτουργίας της αναπτυξιακής πλακέτας με χρήση των φωτεινών LED

4.2 Εργαστηριακοί στόχοι

Οι στόχοι της πρώτης άσκησης συνοψίζονται ως εξής:

1. Επιπλέον εξοικείωση με τα περιβάλλοντα εργασίας που χρησιμοποιήθηκαν στη εισαγωγική Άσκηση 0
2. Δημιουργία νέου project στο περιβάλλον Quartus Prime της Intel
3. Δημιουργία σχεδιαστικών περιορισμών σε αρχείο .sdc
4. Δημιουργία αρχείου αντιστοίχισης εισόδων/εξόδων και ακροδεκτών .csv
5. Δημιουργία νέου αρχείου πηγαίου κώδικα στο περιβάλλον Nios II SBT
6. Εξοικείωση με τον έλεγχο των διακοπτών της αναπτυξιακής πλακέτας μέσω software
7. Σύνταξη, μεταγλώττιση, αποσφαλμάτωση και εκτέλεση του προγράμματος

4.3 Υποδείξεις

1. Το σύστημά σας στον Platform Designer θα πρέπει να περιλαμβάνει τις ακόλουθες μονάδες:

- Clock Source
- NIOS II processor (έκδοση e)
    - Στην καρτέλα “Vector” στα πεδία “Reset vector memory” και “Exception vector memory” επιλέξτε από την λίστα “<Όνομα μονάδας RAM>.s1”
- On-Chip Memory RAM μεγέθους 40 KByte
- System ID Peripheral
- Parallel I/O (PIO) εύρους 17 bit, ορισμένο σαν έξοδος για 17 LED
- Parallel I/O (PIO) εύρους 17 bit, ορισμένο σαν είσοδος για 17 SWITCHES
- Parallel I/O (PIO) εύρους 2 bit, ορισμένο σαν είσοδος για τους 2 διακόπτες πίεσης

2. Χρησιμοποιήστε τις οδηγίες της Άσκησης 0 για να δημιουργήσετε το project directory, να ορίσετε ως όνομα του project το “ledExtend”, και να δημιουργήσετε τα αρχεία “led_extend_pin_assignments.csv” και “ ledExtend.sdc”.

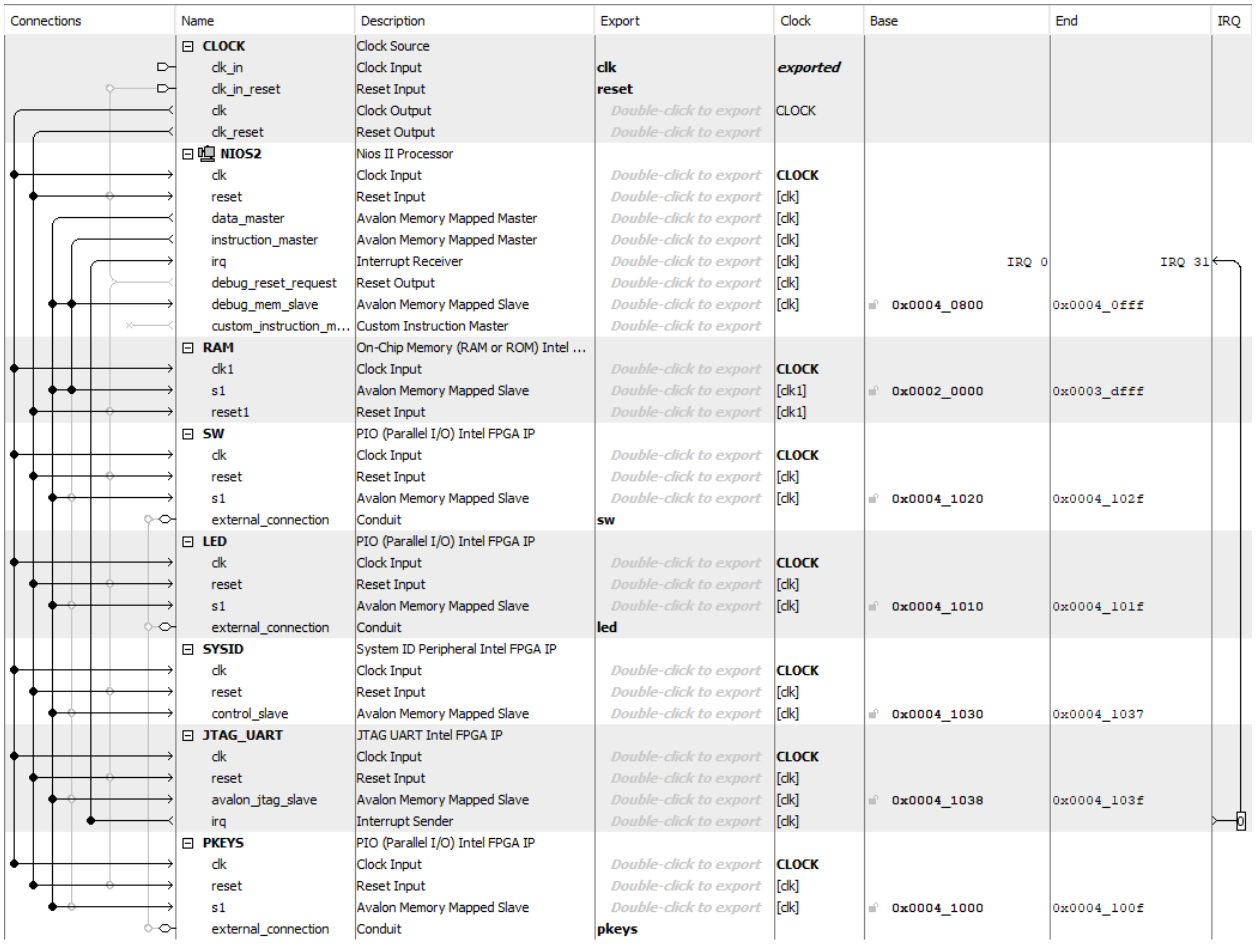
3. Τα κουμπιά πίεσης, όταν είναι πατημένα επιστρέφουν λογικό 0.

4. Για το reset του συστήματος χρησιμοποιήστε τον διακόπτη SW17 της πλακέτας. Σημειώστε ότι σε ένα σύστημα που υλοποιείται με τον Platform Designer, το reset είναι active low (ενεργοποιείται με την τιμή 0). Το τελευταίο σχετίζεται με τη θέση στην οποία πρέπει να βρίσκεται ο διακόπτης κατά τη λειτουργία της πλακέτας (θέση λογικού 1).

5. Φροντίστε να τυπώνετε ένα μήνυμα καλωσορίσματος κατά την εκτέλεση του προγράμματός σας, για επιβεβαίωση της έναρξης λειτουργίας του, μετά το “κατέβασμα” του λογισμικού σας στην πλακέτα.

4.4 Ενδεικτική λύση

Στην Εικόνα 4.2 δίνεται μία ενδεικτική διαμόρφωση του συστήματος που απαιτείται για την υλοποίηση της Άσκησης 1.



Εικόνα 4.2 Ενδεικτική διαμόρφωση συστήματος Άσκησης 1

Στην Εικόνα 4.3 παρατίθεται μία ενδεικτική λύση της Άσκησης 4.1. Ο προτεινόμενος τρόπος προσέγγισης για την επίλυση της συγκεκριμένης άσκησης είναι να χρησιμοποιηθούν οι τακτικές *control flow analysis* και *profiling*. Ακολουθώντας τις τακτικές αυτές μπορούμε να ανακαλύψουμε τα σημεία στα οποία υπάρχουν λάθη.

Μπορούμε να τα εντοπίσουμε τα λάθη αυτά στις γραμμές 17 και 21, και στα μπλοκ επανάληψης *while,* στις γραμμές 31 εώς 46 της Εικόνας 4.1:

- Γραμμή 17: Επιστρέφεται η κατάσταση μόνο του ενός από τα δύο κουμπιά πίεσης.
- Γραμμή 21: Δεν επιστρέφονται όλες οι καταστάσεις των διακοπτών, αλλά μέρος αυτών.
- Γραμμή 32: Θα πρέπει να διαγραφεί η συγκεκριμένη εντολή ανάθεσης, καθώς για να υλοποιηθεί σωστά η λειτουργία της άσκησης, θα πρέπει το διάβασμα της κατάστασης των διακοπτών να γίνεται εντός των δύο εμφωλευμένων δομών επανάληψης *while*, ώστε η κατάσταση των διακοπτών να ανανεώνεται και μέσα στις δομές αυτές.
- Γραμμή 45: Δεν ανατίθενται η αρχική/προκαθορισμένη κατάσταση λογικού 0 σε όλα τα LED, τα οποία προσδιορίζονται από την άσκηση.

```
1  #include "stdio.h"
2  #include "io.h"
3  #include "alt_types.h"
4  #include "system.h"
5
6  // PKeys Status
7  #define RVselected 2
8  #define SWselected 1
9
10 void initial_message(){
11     printf("****************************\n");
12     printf("*   Hello from Nios II!    *\n");
13     printf("****************************\n");
14 }// initial_message
15
16 int getCommand(alt_u32 pkeys_base){
17     return IORD(pkeys_base,0) & 0x03;
18 }// getCommand
19
20 int getSwitches(alt_u32 sw_base){
21     return IORD(sw_base, 0) & 0x01ffff;
22 }// getSwitches
23
24 void ledDisplay(alt_u32 led_base, int disp){
25     IOWR(led_base, 0, disp);
26 }// ledDisplay
27
28 int main(){
29     // check operation status - print welcome message
30     initial_message();
31     while(1){// running forever
32             // SWITCHES Display
33             while(getCommand(PKEYS_BASE) == SWselected){
34                 ledDisplay(LED_BASE, getSwitches(SW_BASE));
35             }// while
36
37             // REVERT SWITCHES Display
38             while(getCommand(PKEYS_BASE) == RVselected){
39                 ledDisplay(LED_BASE, ~getSwitches(SW_BASE));
40             }// while
41
42             // IDLE | ERROR
43             ledDisplay(LED_BASE, 0x00);
44     }// while
45     // unreachable statement
46     return 0;
47 }// main
```

Εικόνα 4.3 Ενδεικτική λύση της άσκησης (σωστή λειτουργία) της αναπτυξιακής πλακέτας με χρήση των φωτεινών LED

4.5 Σχόλια

Έχοντας ολοκληρώσει επιτυχώς τις δύο πρώτες ασκήσεις, Άσκηση 0 και Άσκηση 1, ο φοιτητής είναι έτοιμος, μέσω των υποδείξεων της κάθε άσκησης, της καθοδήγησης των διδασκόντων, αλλά και των πηγών του μαθήματος (ύλη και βιβλία μαθήματος, διαδίκτυο, οδηγοί χρήσης της αναπτυξιακής πλακέτας), να αναπτύξει και να ολοκληρώσει τις πιο προχωρημένες ασκήσεις που ακολουθούν.

5. Άσκηση 2 - Ανάπτυξη υλικού και λογισμικού ενός υπολογιστή ηλικίας

5.1 Εκφώνηση Άσκησης

Ζητούμενο της Άσκηση 2 είναι να φτιαχτεί σύστημα υλικού και λογισμικού για τη πλακέτα DE2-115, το οποίο θα υλοποιεί έναν υπολογιστή ηλικίας. Το σύστημα θα ελέγχεται μέσω των δύο διακοπτών που θα εναλλάσουν το τρόπο λειτουργίας του (δύο καταστάσεις λειτουργίας: λειτουργία εισόδου ημερομηνίας και λειτουργία ελέγχου ημερομηνίας). Επίσης το σύστημα θα έχει υποστήριξη από τα 4 πλήκτρα πίεσης τα οποία θα έχουν διαφορετική συμπεριφορά σε κάθε διαφορετική κατάσταση λειτουργίας. Μόλις δοθούν και οι δύο ημερομηνίες (γέννησης και τρέχουσα) θα μπορεί να εμφανιστεί στην LCD οθόνη η ηλικία που έχει υπολογιστεί.

Αρχικά, στη λειτουργία εισόδου ημερομηνίας θα πρέπει να δίνεται η δυνατότητα να εισάγεται η ημερομηνία γεννήσεως και η τρέχουσα ημερομηνία με χρήση και των 7-segment οθονών. Συγκεκριμένα θα μπορεί να γίνει η εισαγωγή των ημερομηνιών με χρήση των τεσσάρων κουμπιών πίεσης, και με χρήση των διακοπτών να ορίζεται ποια ημερομηνία θα δοθεί.

Για τη δεύτερη λειτουργία (λειτουργία αποθήκευσης ημερομηνίας), με το πρώτο πλήκτρο ο χρήστης θα μπορεί να αποθηκεύσει την ημερομηνία που θα φαίνεται στα ζεύγη 7-segment (το πρώτο ζεύγος θα απεικονίζει την ημέρα, το δεύτερο ζεύγος θα απεικονίζει το μήνα, και με τα τελευταία δύο ζευγάρια θα απεικονίζεται η χρονολογία), με το δεύτερο πλήκτρο θα επαναφέρεται στην οθόνη LCD το αρχικό μήνυμα καλωσορίσματος, με το τρίτο πλήκτρο θα επαναφέρονται τα ζεύγη 7-segment σε μία αρχική ημερομηνία, ενώ με το τέταρτο πλήκτρο θα γίνεται ο υπολογισμός της ημερομηνίας.

Κατά τη διαδικασία της εισαγωγής της ημερομηνίας, θα γίνεται επαναλαμβανόμενο άναμμα και σβήσιμο του ζευγαριού των οθονών 7-segment που είναι επιλεγμένα για ρύθμιση, καθώς και συνεχής αύξηση ή μείωση της τιμής του επιλεγμένου τμήματος, σε περίπτωση παρατεταμένου πατήματος του αντίστοιχου κουμπιού. Επιπλέον, σε κάθε εισαγωγή ημερομηνίας, θα εμφανίζεται στην LCD οθόνη μήνυμα με την ημερομηνία που εισήγαγε ο χρήστης.

5.2 Εργαστηριακοί στόχοι

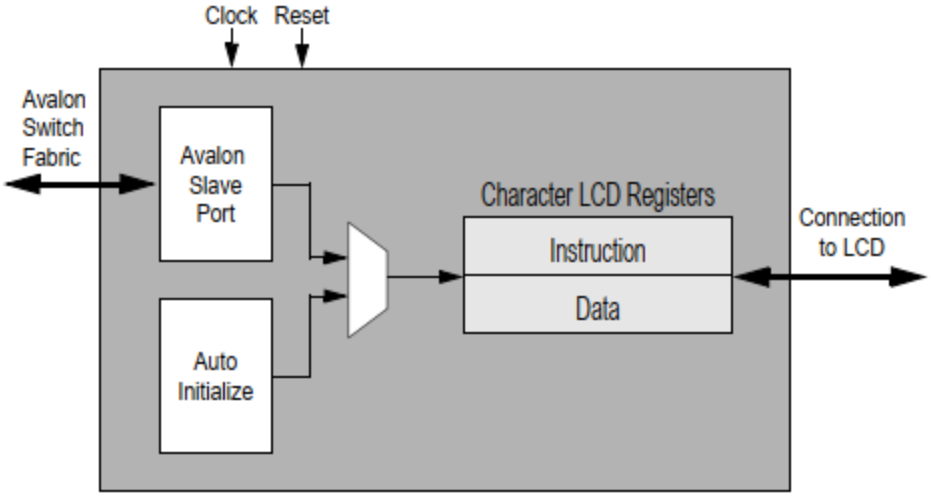
Οι στόχοι της δεύτερης άσκησης για ένα φοιτητή συνοψίζονται ως εξής:

- Εξοικείωση με τη χρήση της ρουτίνας εξυπηρέτησης διακοπής της μονάδας Interval Timer, και της αντίστοιχης βιβλιοθήκης λογισμικού
- Εξοικείωση με τη χρήση της LCD οθόνης, του API καθώς και της βιβλιοθήκης λογισμικού που τη συνοδεύει
- Εξοικείωση με τον έλεγχο οθονών των 7-segment
- Εξοικείωση με τον έλεγχο των κουμπιών πίεσης
- Ανάπτυξη, μεταγλώττιση, αποσφαλμάτωση και εκτέλεση του σχετικού προγράμματος λογισμικού

5.3 Πρόσθετη θεωρία

5.3.1 Οθόνη χαρακτήρων LCD

Η μονάδα απεικόνισης χαρακτήρων 16x2 διευκολύνει την επικοινωνία με την 16x2 οθόνη υγρών κρυστάλλων (Liquid Crystal Display - LCD) των αναπτυξιακών πλακετών της σειράς DE2 της Terasic. Ένα μπλοκ διάγραμμα της μονάδας αυτής φαίνεται στην Εικόνα 5.1.



Εικόνα 5.1 Μπλοκ διάγραμμα της μονάδας απεικόνισης χαρακτήρων 16x2 [23]

Η μονάδα επικοινωνεί με την οθόνη μέσω των καταχωρητών εντολών και δεδομένων που φαίνονται στην Εικόνα 5.1. Η μονάδα περιλαμβάνει κυκλώματα που αρχικοποιούν αυτόματα την οθόνη χαρακτήρων 16x2 όταν το σύστημα στο οποίο ενσωματώνεται βρίσκεται σε κατάσταση επαναφοράς (reset). Η μονάδα απεικόνισης χαρακτήρων 16x2 παρέχει μια διεπαφή για την αποστολή εντολών και δεδομένων στην οθόνη χαρακτήρων 16x2. Επίσης, υποστηρίζει συχνότητα ρολογιού 50 MHz, η οποία είναι άμεσα διαθέσιμη στη σειρά πλακετών DE2.

Η μονάδα απεικόνισης χαρακτήρων 16x2 είναι άμεσα διαθέσιμη στο εργαλείο Platform Designer, του λογισμικού Quartus Prime. Στη καρτέλα ρυθμίσεων ορίζεται μόνο ο τύπος του δρομέα, ενώ οι υπόλοιπες ρυθμίσεις αρχικοποιούνται αυτόματα. Η μονάδα υποστηρίζει τους παρακάτω τύπους λειτουργίας: κανονική (Normal), με αναβοσβήσιμο (Blinking), και τις δύο (κανονική και με αναβοσβήσιμο) και καμία (χωρίς δρομέα).

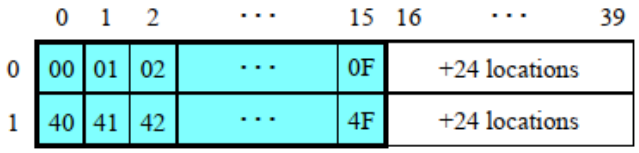
Η διεπαφή προγραμματισμού για τη μονάδα απεικόνισης χαρακτήρων 16x2 διαχειρίζεται τους δύο καταχωρητές που παρουσιάστηκαν στην Εικόνα 5.1. Ο καταχωρητής εντολών χρησιμοποιείται για τον έλεγχο της απεικόνισης χαρακτήρων 16x2 και ο καταχωρητής δεδομένων χρησιμοποιείται για την αποστολή δεδομένων χαρακτήρων στην οθόνη. Τα δεδομένα μπορούν να σταλούν ως κωδικοί ASCII, οι οποίοι μετατρέπονται αυτόματα σε μοτίβα δυαδικών ψηφίων που αναπαριστούν χαρακτήρες στην οθόνη. Η οθόνη υποστηρίζει επίσης και άλλους, μη-ASCII χαρακτήρες, όπως περιγράφεται στο φύλλο δεδομένων της μονάδας απεικόνισης χαρακτήρων 16x2 (LCD datasheet) [23].

Όπως δείχνει ο Πίνακας 5.1, κάθε ένας από τους δύο καταχωρητές της μονάδας είναι πλάτους ενός byte. Η διεύθυνση του καταχωρητή εντολών έχει μετατόπιση 0 από τη διεύθυνση βάσης της μονάδας, ενώ ο καταχωρητής δεδομένων έχει μετατόπιση 1.

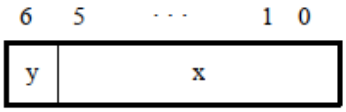
Πίνακας 5.1 Χάρτης καταχωρητών πυρήνα απεικόνισης χαρακτήρων 16x2

Μετατόπιση σε bytes	Όνομα καταχωρητή	Διάβασμα/Εγγραφή	7...0
0	Καταχωρητής εντολών	R/W	Τα bits εντολών που χρησιμοποιούνται για την ανάγνωση και την εγγραφή στην οθόνη.
1	Καταχωρητής δεδομένων	R/W	Διεύθυνση της ROM για την αντιστοίχιση χαρακτήρων. Χρησιμοποιείται για την ανάγνωση και την εγγραφή χαρακτήρων στην οθόνη

Οι καταχωρητές οδηγιών και δεδομένων μπορούν να χρησιμοποιηθούν από κοινού για την αποθήκευση δεδομένων χαρακτήρα σε κάθε θέση στην οθόνη. Η Εικόνα 5.2(α) δείχνει ότι η οθόνη χαρακτήρων 16x2 περιλαμβάνει θέσεις μνήμης για την αποθήκευση δύο σειρών των 40 χαρακτήρων. Οι πρώτες 16 θέσεις σε κάθε σειρά είναι ορατές στην οθόνη, ενώ οι υπόλοιπες όχι. Όπως φαίνεται στην Εικόνα 5.2(α), οι διευθύνσεις των ορατών θέσεων στην πρώτη σειρά είναι οι $(00)_{16}$ ... $(0F)_{16}$, ενώ στη δεύτερη σειρά είναι οι $(40)_{16}$ ... $(4F)_{16}$.



(a) 16 x 2 character display



(b) 16 x 2 character display addresses

Εικόνα 5.2 Μνήμη αποθήκευσης χαρακτήρων της LCD οθόνης και διευθυνσιοδότησή της [23]

Ο καταχωρητής εντολών χρησιμοποιείται για την αποστολή εντολών στην οθόνη χαρακτήρων 16x2, όπως ορίζεται και στο φύλλο δεδομένων της οθόνης. Ορισμένες από τις εντολές που υποστηρίζονται από την οθόνη παρατίθενται στον Πίνακα 5.2. Η πρώτη εντολή, η οποία προσδιορίζεται από το $b_7 = 1$, χρησιμοποιείται για να ορίσει τη θέση του δρομέα στην οθόνη σε μια συγκεκριμένη διεύθυνση. Η διεύθυνση προσδιορίζεται στα bits b_{6-0}, και ακολουθεί το σχήμα διευθυνσιοδότησης που απεικονίζεται στην Εικόνα 5.2(β). Η συγκεκριμένη εικόνα δείχνει πώς σχηματίζεται η διεύθυνση κάθε θέσης από τις συντεταγμένες x, y, όπου y = 0 για την πάνω σειρά, και y = 1 για την κάτω. Αφού οριστεί η θέση του δρομέα, ένας χαρακτήρας μπορεί να φορτωθεί στη θέση αυτή γράφοντας την τιμή του στον καταχωρητή δεδομένων.

Όταν εγγράφονται δεδομένα στη θέση του δρομέα, η οθόνη χαρακτήρων 16x2 προωθεί αυτόματα τον δρομέα μία θέση προς τα δεξιά. Επίσης, μπορούν να φορτωθούν πολλοί χαρακτήρες μαζί στην οθόνη, γράφοντας διαδοχικά κάθε χαρακτήρα στον καταχωρητή δεδομένων. Όπως παρουσιάστηκε στην Εικόνα 5.2, η μνήμη της οθόνης απεικόνισης χαρακτήρων 16x2 περιλαμβάνει 40 θέσεις σε κάθε σειρά. Όταν ο δρομέας είναι σε διεύθυνση μετά από τη $(0F)_{16}$ στην πρώτη γραμμή, οι χαρακτήρες αποθηκεύονται σε θέσεις που δεν είναι ορατές στην οθόνη. Μετά την εγγραφή 40 χαρακτήρων στην πρώτη σειρά, ο κέρσορας προχωρά στην κάτω σειρά, στη διεύθυνση $(40)_{16}$. Μετά το τέλος της κάτω σειράς, ο κέρσορας επιστρέφει στη διεύθυνση $(00)_{16}$.

Η οθόνη απεικόνισης χαρακτήρων 16x2 έχει τη δυνατότητα να μετατοπίσει ολόκληρο το περιεχόμενό της μία θέση προς τα αριστερά ή προς τα δεξιά. Όπως φαίνεται στον Πίνακα 2, η εντολή για μετατόπιση προς τα αριστερά είναι η $(18)_{16}$, και η εντολή για μετατόπιση προς τα δεξιά είναι η $(1C)_{16}$. Αυτές οι εντολές προκαλούν την παράλληλη μετατόπιση των δύο σειρών στην οθόνη. Όταν ένας χαρακτήρας μετατοπίζεται από το ένα άκρο μιας σειράς, επιστρέφει πίσω στο άλλο άκρο της ίδιας σειράς. Όπως αναφέρθηκε, είναι δυνατόν να απενεργοποιηθεί ο δρομέας που αναβοσβήνει στην οθόνη χρησιμοποιώντας την εντολή $(0C)_{16}$, και να ενεργοποιηθεί ξανά χρησιμοποιώντας την εντολή $(0F)_{16}$. Το περιεχόμενο της οθόνης μπορεί να διαγραφεί, και η θέση του δρομέα να ρυθμιστεί στο $(00)_{16}$, χρησιμοποιώντας την εντολή $(01)_{16}$.

Πίνακας 5.2 Εντολές για την οθόνη απεικόνισης χαρακτήρων 16x2 της πλακέτας DE2-115

Εντολή	b_7	b_{6-0}
Ορισμός θέσης του δρομέα	1	Address
Μετακίνηση οθόνης αριστερά	0	0011000
Μετακίνηση οθόνης δεξιά	0	0011100
Δρομέας εκτός λειτουργίας	0	0001100
Δρομέας να αναβοσβήνει	0	0001111
Καθαρισμός οθόνης	0	0000001

5.3.2 Λειτουργίες λογισμικού LCD οθόνης

Οι συναρτήσεις που ακολουθούν αποτελούν τμήμα της βιβλιοθήκης Hardware Abstraction Layer (HAL) της Intel. Για να χρησιμοποιήσετε τις παρακάτω λειτουργίες, ο κώδικας C πρέπει να περιλαμβάνει τη δήλωση:

#include "altera_up_avalon_character_lcd.h"

Οι λειτουργίες που παρέχονται αναφέρονται παρακάτω:

alt_up_character_lcd_init(alt_up_character_lcd_dev *lcd)
- Παράμετρος lcd: Η δομή που διαχειρίζεται τα στοιχεία της μονάδας ελέγχου της LCD (δομή συσκευής).
- Περιγραφή: Αρχικοποίηση της οθόνης LCD καθαρίζοντάς την.

alt_up_character_lcd_open_dev(char *name)
- Παράμετρος name: Το όνομα της οθόνης χαρακτήρων 16x2.
- Επιστρέφει: Την αντίστοιχη δομή συσκευής ή την τιμή NULL, εάν η συσκευή δεν βρεθεί.
- Περιγραφή: Ενεργοποιεί τη συσκευή χαρακτήρων LCD που καθορίζεται με το όνομα.

alt_up_character_lcd_write(alt_up_character_lcd_dev *lcd, char *ptr, int len)
- Παράμετρος lcd: Η δομή της συσκευής ελεγκτή της LCD.
- Παράμετρος ptr: Δείκτης στον buffer από τον οποίο γίνεται η αντιγραφή.
- Παράμετρος len: Το μήκος του buffer.
- Περιγραφή: Γράφει τους χαρακτήρες από το buffer με αρχική διεύθυνση ptr στην οθόνη LCD, ξεκινώντας από το σημείο όπου δείχνει ο τρέχων δρομέας.

alt_up_character_lcd_string(alt_up_character_lcd_dev *lcd, const char *ptr)
- Παράμετρος lcd: Η δομή συσκευής της LCD.
- Παράμετρος ptr: Δείκτης στον buffer από τον οποίο γίνεται η αντιγραφή.
- Περιγραφή: Γράφει τους χαρακτήρες της συμβολοσειράς, στην οποία δείχνει ο δείκτης ptr και έχει τερματιστεί με NULL στην οθόνη LCD.

alt_up_character_lcd_set_cursor_pos(alt_up_character_lcd_dev *lcd, int x_pos, int y_pos)
- Παράμετρος lcd: Η δομή συσκευής της LCD.
- Παράμετρος x_pos: x συντεταγμένη (0 έως 15, από αριστερά προς τα δεξιά).
- Παράμετρος y_pos: y συντεταγμένη (0 για την πάνω σειρά, 1 για την κάτω σειρά).
- Περιγραφή: Ρυθμίζει τη θέση του δρομέα.

alt_up_character_lcd_shift_cursor(alt_up_character_lcd_dev *lcd, int x_right_shift_offset)
- Παράμετρος lcd: Η δομή συσκευής της LCD.
- Παράμετρος x_right_shift_offset: Ο αριθμός των διαστημάτων για μετακίνηση του δρομέα προς τα δεξιά. Εάν η μετατόπιση είναι αρνητική, τότε ο δρομέας μετακινείται προς τα αριστερά.
- Περιγραφή: Μετακινεί τον δρομέα προς τα αριστερά ή προς τα δεξιά.

alt_up_character_lcd_erase_pos(alt_up_character_lcd_dev *lcd, int x_pos, int y_pos)
- Παράμετρος lcd: Η δομή της συσκευής ελεγκτή της LCD.
- Παράμετρος x_pos: x συντεταγμένες (0 έως 15, από αριστερά προς τα δεξιά).
- Παράμετρος y_pos: y συντεταγμένες (0 για την πάνω σειρά, 1 για την κάτω σειρά).
- Περιγραφή: Διαγράφει τον χαρακτήρα στη θέση που προσδιορίζεται από τις συντεταγμένες.

alt_up_character_lcd_cursor_off(alt_up_character_lcd_dev *lcd)
- Παράμετρος lcd: Η δομή συσκευής της LCD.
- Περιγραφή: Απενεργοποιεί τον δρομέα.

alt_up_character_lcd_cursor_blink_on(alt_up_character_lcd_dev *lcd)
- Παράμετρος lcd: Η δομή συσκευής της LCD.
- Περιγραφή: Ενεργοποιεί τον δρομέα.

5.4 Υποδείξεις

1. Το σύστημά σας στον Platform Designer θα πρέπει να περιλαμβάνει τις ακόλουθες μονάδες:

- Clock Source
- NIOS II processor (έκδοση e)
    - Στην καρτέλα "Vector" στα πεδία "Reset vector memory" και "Exception vector memory" επιλέγουμε από τη λίστα "<Όνομα μονάδας RAM>.s1"
- On-Chip Memory RAM μεγέθους 128K
- System ID Peripheral
- Interval Timer εύρους 32 bit
    - Συγγραφή της ρουτίνας εξυπηρέτησης της διακοπής. Στη ρουτίνα αυτή θα πρέπει οπωσδήποτε να αρχικοποιείται το bit "to" του καταχωρητή "status" του μετρητή στην τιμή 0, ώστε να ξεκινήσει εκ νέου η μέτρηση του "timer".
    - Δήλωση της ρουτίνας εξυπηρέτησης της διακοπής μέσω της συνάρτησης "alt_irq_register()".
    - Αρχικοποίηση του timer, η οποία θα πρέπει να περιλαμβάνει τον ορισμό της τιμής μέτρησης του timer (στους καταχωρητές periodl και periodh του timer), καθώς και την εκκίνηση της μέτρησης με επανάληψη και interrupt, θέτοντας τα bit start, cont και ito στον καταχωρητή control.
- 8x Parallel I/O (PIO) εύρους 7 bit, ορισμένα σαν έξοδοι για τα οκτώ 7-segment displays.
- Parallel I/O (PIO) εύρους 4 bit, ορισμένο σαν είσοδος για τους 4 διακόπτες πίεσης.
- Parallel I/O (PIO) εύρους 2 bit, ορισμένο σαν είσοδος για τους 2 διακόπτες.
- JTAG UART

2. Χρησιμοποιήστε τις οδηγίες της Άσκησης 0 για να δημιουργήσετε το project directory, να ορίσετε ως όνομα του project το "age_calc", και να δημιουργήσετε τα αρχεία "age_calc_pin_assignments.csv" και "age_calc.sdc".

3. Τα κουμπιά πίεσης, όταν είναι πατημένα επιστρέφουν λογικό 0.

4. Για το reset του συστήματος χρησιμοποιήστε τον διακόπτη SWITCH17 της πλακέτας.

5, Για να χρησιμοποιήσετε τις συναρτήσεις της LCD οθόνης κάντε "#include" τη βιβλιοθήκη "altera_up_avalon_character_lcd.h".

6. Για να χρησιμοποιήσετε τις συναρτήσεις που προσφέρει η Intel για τα interrupts, κάντε "#include" τη βιβλιοθήκη "priv/alt_legacy_irq.h".

7. Προσέξτε ότι για να χρησιμοποιήσετε τα interrupt του Timer, θα πρέπει να έχετε κάνει τις κατάλληλες συνδέσεις μεταξύ αυτού και του επεξεργαστή NIOS II, στη στήλη IRQ (Interrupt ReQuest) του Platform Designer, καθώς και να ορίσετε σωστά τις προτεραιότητές τους όταν θα χρησιμοποιείται για λόγους αποσφαλμάτωσης και το JTAG UART interrupt.

8. Φροντίστε να τυπώνετε ένα μήνυμα καλωσορίσματος κατά την εκτέλεση του προγράμματός σας για επιβεβαίωση της έναρξης λειτουργίας του μετά το "κατέβασμα" του λογισμικού σας στην πλακέτα.

5.5 Ενδεικτική λύση

Στην Εικόνα 5.3 δίνεται μία Ενδεικτική διαμόρφωση του ζητούμενου συστήματος:

Name	Description	Export	Clock	Base	End
⊞ **clk**	Clock Source		***exported***		
⊞ **nios2**	Nios II Processor		**clk**	0x0004_0800	0x0004_0fff
⊞ **ram**	On-Chip Memory (RAM or ROM) Intel ...		**clk**	0x0002_0000	0x0003_ffff
⊞ **lcd**	16x2 Character Display		**clk**	0x0004_10d0	0x0004_10d1
⊞ **sysid**	System ID Peripheral Intel FPGA IP		**clk**	0x0004_10c0	0x0004_10c7
⊞ **jtag_uart**	JTAG UART Intel FPGA IP		**clk**	0x0004_10c8	0x0004_10cf
⊞ **pkeys**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_10b0	0x0004_10bf
⊞ **switches**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_10a0	0x0004_10af
⊞ **timer**	Interval Timer Intel FPGA IP		**clk**	0x0004_1000	0x0004_101f
⊞ **sseg_0**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1090	0x0004_109f
⊞ **sseg_1**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1080	0x0004_108f
⊞ **sseg_2**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1070	0x0004_107f
⊞ **sseg_3**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1060	0x0004_106f
⊞ **sseg_4**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1050	0x0004_105f
⊞ **sseg_5**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1040	0x0004_104f
⊞ **sseg_6**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1030	0x0004_103f
⊞ **sseg_7**	PIO (Parallel I/O) Intel FPGA IP		**clk**	0x0004_1020	0x0004_102f

Εικόνα 5.3 Ενδεικτική διαμόρφωση συστήματος της Άσκησης 2

Στο Παράρτημα Α επισυνάπτεται η προτεινόμενη λύση (κώδικας και αντιστοίχιση ακροδεκτών) της Άσκησης 2. Παρακάτω περιγράφονται συνοπτικά η χρήση κάθε συνάρτησης που αναπτύχθηκε για την επίλυση της συγκεκριμένης άσκησης, καθώς καθώς και τη χρήση των βασικότερων global μεταβλητών:

numbers: (Μεταβλητή): Η μεταβλητή *numbers* χρησιμοποιήθηκε για την αντιστοίχιση των αριθμών 0 έως 9 με τις τιμές ενεργοποίησης ή μη των LED για την απεικόνιση των αριθμών αυτών στις οθόνες 7-segment.

temp_year / **temp_month** / **temp_date**: (Μεταβλητές) Προσωρινή αποθήκευση χρονολογίας, μήνα και ημέρας κατά την καταχώρηση τους από το χρήστη. Οι μεταβλητές είναι πίνακες αριθμών που αντιστοιχούν (ο καθένας) στα ζεύγη των οθονών απεικόνισης 7-segment.

curr_year / **curr_month** / **curr_date**: (Μεταβλητές) Αποθήκευση τρέχουσας χρονολογίας, μήνα και ημέρας.

byear / **bmonth** / **bdate**: (Μεταβλητές) Αποθήκευση χρονολογίας, μήνα και ημέρας γεννήσεως του χρήστη.

curr_age / **birth_d** / **curr_d** (Μεταβλητές) Έτη, μήνες και μέρες της ηλικίας του χρήστη.

initial_message(): (Βοηθητική) Συνάρτηση εκτύπωσης μηνύματος καλωσορίσματος κατά την εκτέλεση της εφαρμογής στο τερματικό του περιβάλλοντος ανάπτυξης λογισμικού Eclipse Nios II EDS.

getCommand(): (Βοηθητική) Η συνάρτηση διαβάζει και επιστρέφει το περιεχόμενο ενός καταχωρητή Nios PIO, δεδομένης της διεύθυνσης του καταχωρητή ενδιαφέροντος (χρήση για διάβασμα των καταστάσεων των διακοπτών πίεσης).

init_timer()/timer_isr(): (Βασικές) Συναρτήσεις αρχικοποίησης και εξυπηρέτησης της διακοπής του timer.

age_calculator(): (Βασική) Συνάρτηση υπολογισμού ηλικίας.

printToLCD(): (Βασική) Εκτύπωση μηνυμάτων στην οθόνη απεικόνισης χαρακτήρων LCD.

display_sseg(): (Βασική) Διαχείριση απεικόνισης των κατάλληλων αριθμών στα ζεύγη οθωνών 7-segment για την εισαγωγή της ημερομηνίας από το χρήστη.

reset_date(): (Βοηθητική) Επαναφορά προκαθορισμένης ημερομηνίας στις οθόνες απεικόνισης 7-segment.

setting_mode(): (Βασική) Λειτουργία εισαγωγής ημερομηνίας από το χρήστη.

save_mode(): (Βασική) Λειτουργία αποθήκευσης ημερομηνίας. Είναι η πιο βασική λειτουργία της εφαρμογής καθώς μετατρέπει το περιεχόμενο των πινάκων αριθμών temp_year, temp_month και temp_date στους αριθμούς που απεικονίζουν τα αντίστοιχα ζεύγη των 7-segment, καθώς επίσης αναλαμβάνει και να εκτελέσει τις εντολές τις οποίες δίνει ο χρήστης μέσω των διακοπτών πίεσης.

main(): Γενική συνάρτηση η οποία αποτελεί τη προκαθορισμένη αρχή του προγράμματος. Αναλαμβάνει να εναλλάξει τις δύο βασικές λειτουργίες, ανάλογα με τις τιμές των σχετικών διακοπτών, καθώς και να αναβοσβήσει τις οθόνες 7-segment κατα τη λειτουργία εισαγωγής των ημερομηνιών.

6. Άσκηση 3 - Ανάπτυξη υλικού και λογισμικού για την υλοποίηση του παιχνιδιού “Αγαλματάκια ακούνητα”

6.1 Εκφώνηση Άσκησης

Ζητούμενο της άσκησης 2 είναι να φτιαχτεί σύστημα υλικού και λογισμικού για την πλακέτα DE2-115, το οποίο θα υλοποιεί το παιχνίδι “Αγαλματάκια ακούνητα”. Το παιχνίδι θα χρησιμοποιεί τα 26 LED της πλακέτας που είναι τοποθετημένα στη σειρά (18 κόκκινα και 8 πράσινα). Στο ξεκίνημα του παιχνιδιού θα υπάρχουν δύο LED αναμμένα στο κέντρο (αρχική κατάσταση). Με το πάτημα του πρώτου κουμπιού (KEY3) θα ξεκινήσουν να “κινούνται” προς τις άκρες με τυχαία κατεύθυνση, ανάβοντας το ένα LED μετά το άλλο προσομοιώνοντας έτσι τη κίνηση. Όταν φτάσουν στην άκρη θα αλλάζουν κατεύθυνση γυρίζοντας προς τα πίσω, κάτι το οποίο θα συμβαίνει επίσης όταν συναντηθούν στη διαδρομή. Επιπλέον, με το πάτημα του πρώτου κουμπιού (KEY3), όλα τα αναμμένα LED θα αλλάζουν τυχαία την κατεύθυνση κίνησής τους.

Με το πάτημα του δεύτερου κουμπιού (KEY2) θα προστίθεται ένα επιπλέον LED στο κέντρο της τρέχουσας απόστασης που θα έχουν τα ήδη δύο αναμμένα LED από την αρχική κατάσταση. Αν είναι περισσότερα από δύο αναμμένα, τότε θα προστίθεται στο κέντρο της μεγαλύτερης απόστασης μεταξύ δύο LED. Ο μέγιστος αριθμός LED που θα μπορούν να είναι ταυτόχρονα αναμμένα είναι τέσσερα. Αντίστοιχα, με το τρίτο κουμπί (KEY1) θα επιλέγεται τυχαία ένα από τα LED και θα αφαιρείται από το παιχνίδι. Με το πάτημα του τέταρτου κουμπιού (KEY0) το παιχνίδι θα επαναφέρεται στην αρχική του κατάσταση (reset).

Επιπλέον, θα πρέπει να υλοποιηθεί έλεγχος της ταχύτητας με τη χρήση των τριών διακοπτών (SWITCHES 0 έως 2), ώστε να μπορεί να οριστεί η ταχύτητα του παιχνιδιού (ο πρώτος διακόπτης για το αργό, ο δεύτερος για το μέτριο και ο τρίτος για το γρήγορο αναβόσβημα των LED).

Όσο κρατιέται πατημένο ένα κουμπί, καθώς επίσης και όταν και οι τρεις διακόπτες είναι απενεργοποιημένοι, ή περισσότεροι από ένας είναι ενεργοποιημένοι, τότε τα LED θα παραμένουν στην τελευταία τους ενεργή θέση (“σε κάθε ενέργεια του «παίκτη» τα αγαλματάκια παραμένουν ακούνητα”).

6.2 Εργαστηριακοί στόχοι

Μετά την ολοκλήρωση της παρούσας άσκησης ο φοιτητής θα έχει αποκτήσει τις απαιτούμενες γνώσεις για να:

- Δημιουργεί ρουτίνες εξυπηρέτησης διακοπών που προκαλούνται από τα κουμπιά πίεσης.
- Αναπτύξει, μεταγλωττίσει, αποσφαλματώσει και να εκτελέσει ένα πρόγραμμα λογισμικού στο οποίο γίνεται χρήση ρουτινών εξυπηρέτησης διακοπών.

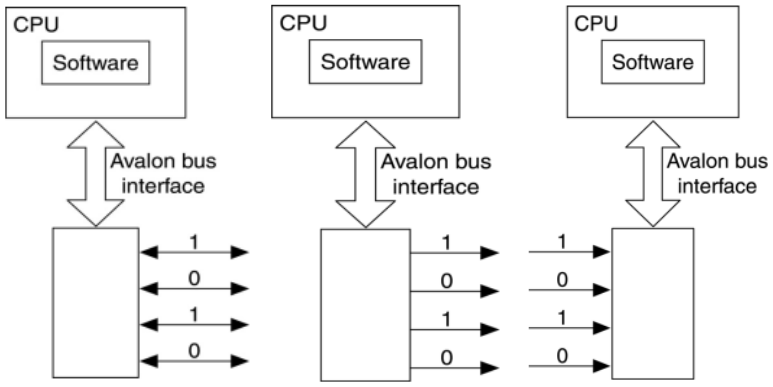
6.3 Πρόσθετη θεωρία

Η μονάδα παράλληλης εισόδου/εξόδου Nios (Nios PIO) είναι ένα στοιχείο βιβλιοθήκης του Platform Designer που περιλαμβάνεται στο κιτ ανάπτυξης του Nios. Πρόκειται για μία μονάδα παράλληλης εισόδου/εξόδου εύρους 1 έως 32 bit, τα οποία μπορούν να οριστούν σαν είσοδοι, έξοδοι ή και τα δύο. Ο πηγαίος κώδικας της PIO μονάδας είναι πλήρως διαθέσιμος σε Verilog και VHDL για περαιτέρω ανάπτυξη, ενώ επίσης διαθέσιμες είναι και οι απαραίτητες υπορουτίνες λογισμικού για εύκολη ανάπτυξη συστήματος.

6.3.1 Nios PIO

Μια παράλληλη μονάδα εισόδου/εξόδου (PIO) είναι μια διεπαφή μεταξύ του επεξεργαστή NIOS II και μιας περιφερειακής συσκευής που θέλουμε να συνδέσουμε σε αυτόν. Το PIO μπορεί να χρησιμοποιηθεί για διασύνδεση συσκευών που βρίσκονται είτε εντός είτε εκτός του FPGA, στο οποίο υλοποιείται ο NIOS II.

Τα σχήματα στην Εικόνα 6.1 απεικονίζουν τις δυνατές διαμορφώσεις PIO τεσσάρων θυρών. Αρχικά φαίνεται ένα PIO τριών καταστάσεων (tri-state) που περιλαμβάνει αμφίδρομες θύρες εισόδου/εξόδου, κατόπιν παρουσιάζεται ένα PIO μόνο για έξοδο που περιλαμβάνει θύρες μόνο για έξοδο, και τέλος ένα PIO μόνο για είσοδο, το οποίο περιλαμβάνει θύρες μόνο εισόδου.



Εικόνα 6.1 Δυνατές διαμορφώσεις PIO τεσσάρων θυρών [24]

6.3.2 Καταχωρητές PIO

Ακολουθεί ο χάρτης καταχωρητών του Nios PIO:

Πίνακας 6.1 Χάρτης καταχωρητών Nios PIO

Διεύθυνση μετατόπισης	**Όνομα καταχωρητή**		**R/W**	**Μέγεθος μεταβλητής - 1 έως 32 bit**
0	data	read	RO	Η τιμή δεδομένων στις εισόδους του PIO
		write	WO	Νέα τιμή για την οδήγηση των εξόδων του PIO
1	direction		RW	Κατεύθυνση δεδομένων (*προαιρετικό*): Ατομικός έλεγχος για κάθε bit του PIO
2	interruptmask		RW	Μάσκα διακοπής (*προαιρετικό*): Ενεργοποίηση/απενεργοποίηση IRQ ανά θύρα (όταν το PIO λειτουργεί σαν είσοδος)
3	edgecapture		RW	Καταγραφή ακμών (*προαιρετικό*): Ανίχνευση και συγκράτηση αλλαγής τιμής (ακμής) ανά θύρα (και πάλι για λειτουργία εισόδου)

Καταχωρητής **data**: Για ένα PIO μόνο εισόδου, η εγγραφή στον καταχωρητή δεδομένων δεν έχει καμία επίδραση. Για ένα PIO μόνο εξόδου, η ανάγνωση από τον καταχωρητή δεδομένων παράγει απροσδιόριστο αποτέλεσμα.

Καταχωρητής **direction**: Όταν μια συσκευή PIO έχει οριστεί σαν διπλής κατεύθυνσης (bidir), τόσο οι θύρες εισόδου, όσο και οι θύρες εξόδου, συνδέονται σε μια πύλη τριών καταστάσεων. Ο καταχωρητής κατεύθυνσης ελέγχει την κατεύθυνση δεδομένων για κάθε bit του PIO. Ένα bit κατεύθυνσης ίσο με 0(ή 1) θέτει την αντίστοιχη θύρα του PIO σαν είσοδο (ή έξοδο). Κατά την επαναφορά του συστήματος, όλα τα δυαδικά ψηφία κατεύθυνσης ρυθμίζονται στο 0.

Καταχωρητής **interruptmask**: Όταν ένα bit στον καταχωρητή αυτόν είναι ρυθμισμένο στο 1, οι διακοπές είναι ενεργοποιημένες για την αντίστοιχη θύρα PIO. Η ύπαρξη του καταχωρητή αυτού και το είδος της μεταβολής του αντίστοιχου σήματος εισόδου που προκαλεί τη διακοπή ρυθμίζονται στο πεδίο “Interrupt” της φόρμας ρυθμίσεων του PIO. Κατά την επαναφορά συστήματος, στο μητρώο διακοπής όλα τα bit γίνονται μηδενικά και οι διακοπές απενεργοποιούνται για όλες τις θύρες PIO.

Καταχωρητής **edgecapture**: Εάν η επιλογή edge_type στο πεδίο “Edge capture register” της φόρμας ρυθμίσεων του PIO έχει οριστεί σε RISING, FALLING ή ANY, στον καταχωρητή αυτόν τίθεται ένα bit στο 1 για να υποδεικνύει πότε ανιχνεύθηκε μια ακμή στην αντίστοιχη θύρα εισόδου PIO.

6.4 Υποδείξεις

1. Το σύστημά σας στον Platform Designer θα πρέπει να περιλαμβάνει τις ακόλουθες μονάδες:

- Clock Source
- NIOS II processor (έκδοση e)
    - Στην καρτέλα "Vector" στα πεδία "Reset vector memory" και "Exception vector memory" επιλέγουμε από τη λίστα "<Όνομα μονάδας RAM>.s1"
- On-Chip Memory RAM μεγέθους 80K
- System ID Peripheral
- Interval Timer εύρους 32 bit
    - Συγγραφή της ρουτίνας εξυπηρέτησης της διακοπής. Στη ρουτίνα αυτή θα πρέπει οπωσδήποτε να αρχικοποιείται το bit "to" του καταχωρητή "status" του μετρητή στην τιμή 0, ώστε να ξεκινήσει εκ νέου η μέτρηση του "timer".
    - Δήλωση της ρουτίνας εξυπηρέτησης της διακοπής μέσω της συνάρτησης "alt_irq_register()".
    - Αρχικοποίηση του timer η οποία θα πρέπει να περιλαμβάνει τον ορισμό της τιμής μέτρησης του timer (στους καταχωρητές periodl και periodh του timer), καθώς και την εκκίνηση της μέτρησης με επανάληψη και interrupt, θέτοντας τα bit start, cont και ito στον καταχωρητή control.
- Parallel I/O (PIO) εύρους 26 bit, ορισμένο σαν έξοδος για τα 26 LED.
- Parallel I/O (PIO) εύρους 4 bit, ορισμένο σαν είσοδος για τους 4 διακόπτες πίεσης.
    - Στις ρυθμίσεις του συγκεκριμένου PIO ενεργοποιούμε στο πεδίο "Edge capture register" την επιλογή "Synchronous capture", επιλέγουμε από τη λίστα "Edge type" την επιλογή "Falling", και τέλος ενεργοποιούμε το "Enable bit-clearing for edge capture register".
    - Στο επόμενο πεδίο θα πρέπει να έχει ενεργοποιηθεί το "Generate IRQ" και να έχει οριστεί από τη λίστα ο τύπος του "IRQ Type" ως "EDGE".
    - Θα πρέπει επίσης να γραφεί η αντίστοιχη ρουτίνα εξυπηρέτησης της διακοπής. Στη ρουτίνα αυτή θα πρέπει οπωσδήποτε να αρχικοποιείται ο καταχωρητής "interruptmask" στην τιμή 0, ώστε να ενεργοποιηθούν οι διακοπές ανά bit.
    - Τέλος θα πρέπει να γίνει δήλωση της ρουτίνας εξυπηρέτησης της διακοπής μέσω της συνάρτησης "alt_irq_register()".
- Parallel I/O (PIO) εύρους 3 bit, ορισμένο σαν είσοδος για τους 3 διακόπτες που ελέγχουν την ταχύτητα.
- JTAG UART

2. Χρησιμοποιήστε τις οδηγίες της Άσκησης 0 για να δημιουργήσετε το project directory, να ορίσετε ως όνομα του project το "statues", και να δημιουργήσετε τα αρχεία "statues_pin_assignments.csv" και "statues.sdc".

3. Τα κουμπιά πίεσης, όταν είναι πατημένα επιστρέφουν λογικό 0.

4. Για το reset του συστήματος χρησιμοποιήστε τον διακόπτη SWITCH17 της πλακέτας.

5. Προσέξτε ότι για να χρησιμοποιήσετε τα interrupt από το PIO των κουμπιών πίεσης και τον Timer, θα πρέπει να έχετε κάνει τις κατάλληλες συνδέσεις μεταξύ αυτών και του επεξεργαστή NIOS II, στη στήλη IRQ (Interrupt ReQuest) του Platform Designer, καθώς και να ορίσετε σωστά τις προτεραιότητές τους όταν θα χρησιμοποιείται για λόγους αποσφαλμάτωσης και το JTAG UART interrupt.

6. Για να χρησιμοποιήσετε τις συναρτήσεις που προσφέρει η Intel για τα interrupts, κάντε “#include” τη βιβλιοθήκη “priv/alt_legacy_irq.h”.

7. Φροντίστε να τυπώνετε ένα μήνυμα καλωσορίσματος κατά την εκτέλεση του προγράμματός σας για επιβεβαίωση της έναρξης λειτουργίας του μετά το “κατέβασμα” του λογισμικού σας στην πλακέτα.

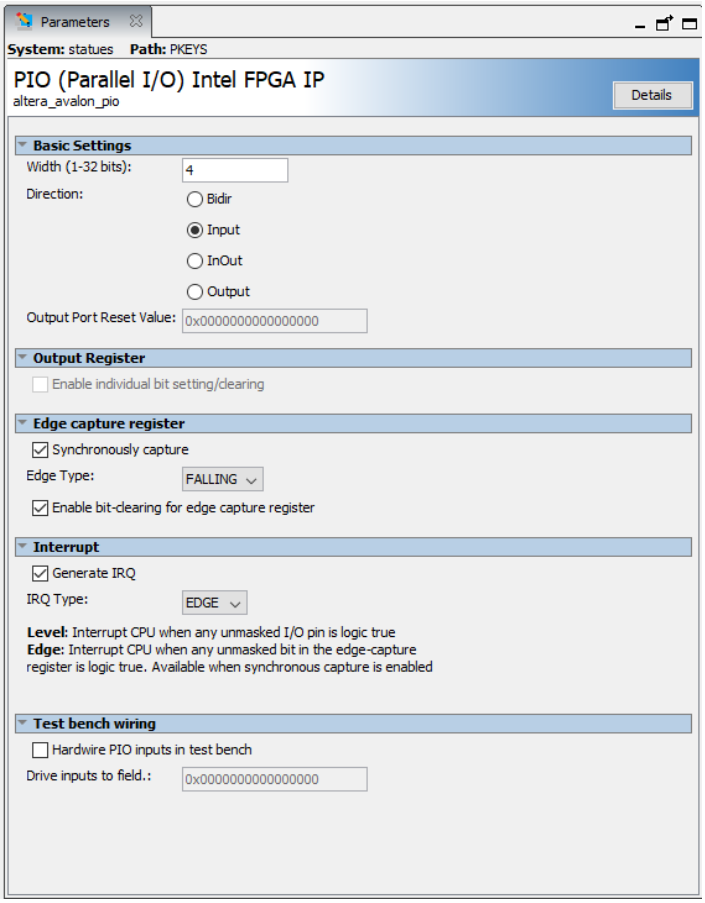
6.5 Ενδεικτική λύση

Στην Εικόνα 6.2 δίνεται μία Ενδεικτική διαμόρφωση του απαιτούμενου για αυτή την άσκηση συστήματος:

Name	Description	Export	Clock	Base	End	IRQ
CLOCK	Clock Source		*exported*			
NIOS2	Nios II Processor		CLOCK	0x0004_0800	0x0004_0fff	
RAM	On-Chip Memory (RAM or ROM) Intel ...		CLOCK	0x0002_0000	0x0003_17ff	
SYSID	System ID Peripheral Intel FPGA IP		CLOCK	0x0004_1050	0x0004_1057	
PKEYS	PIO (Parallel I/O) Intel FPGA IP		CLOCK	0x0004_1040	0x0004_104f	2
LEDS	PIO (Parallel I/O) Intel FPGA IP		CLOCK	0x0004_1020	0x0004_102f	
SWITCHES	PIO (Parallel I/O) Intel FPGA IP		CLOCK	0x0004_1030	0x0004_103f	
TIMER	Interval Timer Intel FPGA IP		CLOCK	0x0004_1000	0x0004_101f	1
JU	JTAG UART Intel FPGA IP		CLOCK	0x0004_1058	0x0004_105f	0

Εικόνα 6.2 Ενδεικτική διαμόρφωση συστήματος Άσκησης 3

Επισυνάπτονται οι ρυθμίσεις που πρέπει να πραγματοποιηθούν στην συσκευή εξόδου των κουμπιών πίεσης:



Εικόνα 6.3 Ρυθμίσεις παραμέτρων για το PIO των κουμπιών πίεσης της Άσκησης 3

Στο Παράρτημα Β επισυνάπτεται η προτεινόμενη λύση (κώδικας και αντιστοίχιση ακροδεκτών) της Άσκησης 3. Παρακάτω δίνεται μία συνοπτική περιγραφή της λειτουργίας κάθε συνάρτησης που αναπτύχθηκε για την επίλυση της συγκεκριμένης Άσκησης, καθώς και της χρησιμότητας των βασικότερων global μεταβλητών:

enable_led_left_direction: (Μεταβλητή) Πίνακας καταχώρησης κατεύθυνσης των LED. Η λογική τιμή 1 συμβολίζει την κατεύθυνση προς τα αριστερά, και αντίστοιχα, η λογική τιμή 0 συμβολίζει την κατεύθυνση προς τα δεξιά.

enabled_led: (Μεταβλητή) Πίνακας καταχώρησης ενεργοποιημένων LED. Η λογική τιμή 1 συμβολίζει ότι ένα LED είναι ενεργοποιημένο και, αντίστοιχα, η λογική τιμή 0 συμβολίζει ότι το LED είναι απενεργοποιημένο.

initial_message(): (Βοηθητική) Συνάρτηση εκτύπωσης μηνύματος καλωσορίσματος στο τερματικό του περιβάλλοντος ανάπτυξης λογισμικού Eclipse Nios II EDS κατά την έναρξη της εκτέλεσης της εφαρμογής.

getCommand(): (Βοηθητική) Η συνάρτηση αυτή διαβάζει και επιστρέφει το περιεχόμενο ενός καταχωρητή Nios PIO, δεδομένης της διεύθυνσης του καταχωρητή ενδιαφέροντος (χρήση για διάβασμα των καταστάσεων των διακοπτών πίεσης).

init_timer()/timer_isr(): (Βασικές) Συναρτήσεις αρχικοποίησης και εξυπηρέτησης της διακοπής του timer.

init_pkeys()/PKeys_isr(): (Βασικές): Συναρτήσεις αρχικοποίησης και εξυπηρέτησης της διακοπής του PIO στο οποίο συνδέονται οι διακόπτες πίεσης.

getSwitches(): (Βοηθητική) Η συνάρτηση διαβάζει και επιστρέφει το περιεχόμενο ενός καταχωρητή Nios PIO. Η χρήση της συγκεκριμένης συνάρτησης περιορίζεται στο να διαβάσει και να επιστρέψει τις καταστάσεις των τριών πρώτων διακοπτών της πλακέτας (αυτών που ορίζουν την ταχύτητα κίνησης των LED).

countEnabledLeds(): (Βοηθητική) Επιστρέφει το πλήθος των αναμένων LED.

findFirstDisabledLed(): (Βοηθητική) Επιστρέφει την πρώτη θέση απενεργοποιημένου LED από τον πίνακα *enabled_led*.

findEnabledLed(): (Βοηθητική) Επιστρέφει την πρώτη θέση ενεργοποιημένου LED από τον πίνακα *enabled_led*.

enableThirdLed()/enableFourthLed() (Βοηθητικές) Ενεργοποιούν το τρίτο και τέταρτο LED στο παιχνίδι.

bubbleSort(): (Βοηθητική) Ο απλούστερος αλγόριθμος ταξινόμησης που λειτουργεί με την κατ' επανάληψη εναλλαγή γειτονικών στοιχείων αν είναι σε λανθασμένη σειρά. Χρησιμοιείται από τις συναρτήσεις *enableThirdLed* και *enableFourthLed* για την εύρευση της σωστής θέσης ενεργοποίησης του τρίτου και του τέταρτου LED αντίστοιχα.

speedRegulator(): (Βασική) Καθορισμός ταχύτητας του παιχνιδιού λαμβάνοντας υπόψη τις καταστάσεις των τριών διακοπτών.

checkInBoard(): (Βοηθητική) Έλεγχος σωστής θέσης για κάθε αναμμένο LED.

left_led_flash(): (Βασική) “Μετατόπιση” της θέσης αναμένου LED προς τα αριστερά.

right_led_flash(): (Βασική) “Μετατόπιση” της θέσης αναμένου LED προς τα δεξιά.

ledDirection(): (Βασική) Συνάρτηση ορισμού κατεύθυνσης αναμένου LED.

rePositioning(): (Βασική) Συνάρτηση ελέγχου συγκρούσεων αναμένων LED.

statuesGame(): (Βασική) Κύρια συνάρτηση του λογισμικού του παιχνιδιού. Δεδομένης της ταχύτητας που καθορίζεται από τη συνάρτηση speedRegulator, για κάθε ενεργοποιημένο LED ορίζει τη θέση και την κατεύθυνση του.

main(): Γενική συνάρτηση η οποία είναι η προκαθορισμένη αρχή του προγράμματος.

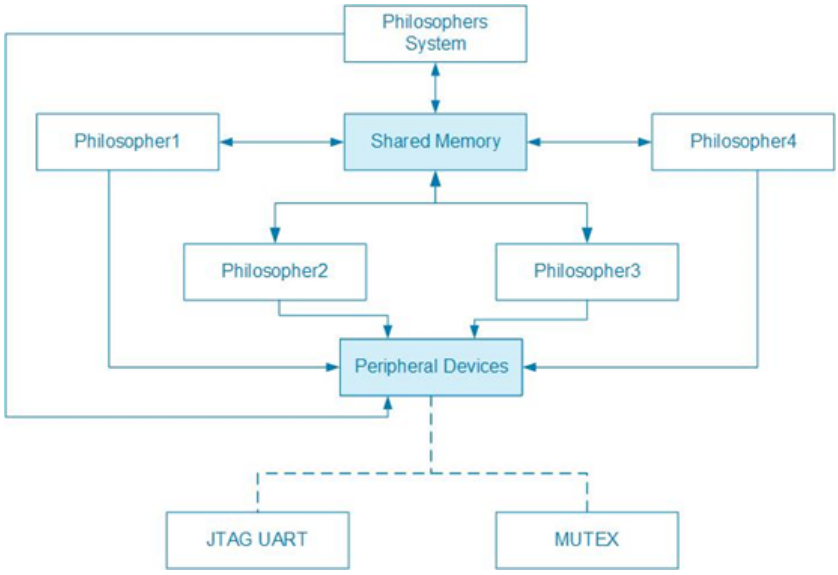
7 Άσκηση 4 - Ανάπτυξη υλικού και λογισμικού για την υλοποίηση ενός πολυεπεξεργαστικού συστήματος

7.1 Εκφώνηση Άσκησης

Η άσκηση αυτή εστιάζει στην υλοποίηση με πολλαπλούς επεξεργαστές Nios II σε ένα Intel FPGA. Το παράδειγμα των φιλοσόφων στοχεύει πρωτίστως στην εκμάθηση της σχεδίασης ενός πολυεπεξεργαστικού συστήματος, ενώ σε δεύτερη φάση εστιάζει στην χρήση κατάλληλου λογισμικού για τον συντονισμό μεταξύ των διεργασιών του συστήματος. Παρακάτω παρουσιάζεται το κλασικό πρόβλημα των συνδαιτυμόνων φιλοσόφων, το οποίο συμβολίζει την κατανόηση των ζητημάτων της ανταλλαγής πόρων και του συγχρονισμού.

Φανταστείτε πέντε φιλοσόφους καθισμένους σε ένα στρογγυλό τραπέζι. Ένα μοναδικό πιρούνι τοποθετείται ανάμεσα σε κάθε φιλόσοφο. Κάθε φιλόσοφος προσπαθεί πρώτα να αρπάξει το πιρούνι στα αριστερά του, και στη συνέχεια το πιρούνι στα δεξιά του. Εάν αποκτήσει και τα δύο πιρούνια, ο φιλόσοφος μπορεί να φάει. Μετά από μια μικρή καθυστέρηση, η οποία αντιπροσωπεύει το χρόνο που τρώει, ο φιλόσοφος αφήνει και τα δύο πιρούνια, τα οποία είναι πλέον διαθέσιμα στους υπόλοιπους φιλοσόφους. Για να αποφευχθεί το αδιέξοδο, εάν κάποιος φιλόσοφος δεν μπορεί να σηκώσει το δεξί πιρούνι αμέσως μετά το σήκωμα του αριστερού, πρέπει να αφήσει το αριστερό πιρούνι και να προσπαθήσει ξανά αργότερα.

Ζητούμενο της Άσκηση 4 είναι να φτιαχτεί σύστημα υλικού και λογισμικού για τη πλακέτα DE2-115, στο οποίο υλοποιείται μία παραλλαγή του παραδείγματος των φιλοσόφων, για την οποία ακολουθεί το σενάριο του συστήματος το οποίο παρουσιάζεται στο παρακάτω διάγραμμα:



Εικόνα 7.1 Περιγραφή συστήματος σεναρίου φιλοσόφων

Το σενάριο αφορά τέσσερις φιλοσόφους, συν έναν επιπλέον «συντονιστή», όπου όλοι μαζεύονται για να συζητήσουν. Ο συντονιστής δίνει το σύνθημα για να ξεκινήσει η συζήτηση μεταξύ των φιλοσόφων. Κάθε φορά που κάποιος φιλόσοφος λέει την άποψή του, οι υπόλοιποι θα πρέπει να ακούνε χωρίς να τον διακόπτουν. Κάθε κύκλος συζητήσεων τελειώνει αφού μιλήσουν όλοι οι φιλόσοφοι από μία φορά. Όλοι οι φιλόσοφοι διεκδικούν την ευκαιρία να μιλήσουν ισότιμα. Τους κανόνες της συζήτησης τους «επιβάλλει» ο συντονιστής της κουβέντας.

Σε κάθε φιλόσοφο θα πρέπει να ανατεθεί από ένα LED και ένα SWITCH. Κάθε φορά που κάποιος από τους φιλοσόφους θα λέει την άποψή του, το LED που του αντιστοιχεί θα αναβοσβήνει κάθε δευτερόλεπτο. Κάθε φιλόσοφος για να μιλήσει θα πρέπει να έχει ζητήσει τον λόγο, το οποίο υποδεικνύεται από την κατάσταση του αντίστοιχου διακόπτη (κατάσταση λογικού μηδέν - κατεβασμένος διακόπτης - δεν θέλει να μιλήσει, κατάσταση λογικού ένα - ανεβασμένος διακόπτης - θέλει να μιλήσει). Κάθε φιλόσοφος θα πρέπει να μπορεί να μιλήσει το πολύ 10 δευτερόλεπτα. Αν θελήσει να σταματήσει να μιλάει νωρίτερα μπορεί να αλλάξει την κατάσταση του διακόπτη του.

7.2 Εργαστηριακοί στόχοι

Μετά την ολοκλήρωση της παρούσας άσκησης, ο φοιτητής θα έχει αποκτήσει την απαιτούμενη γνώση για να:

- Δημιουργεί ιεραρχικά συστήματα στον Platform Designer, τα οποία θα περιέχουν πολλαπλούς Nios II επεξεργαστές, χρησιμοποιώντας διεπαφές, οι οποίες θα παρέχουν πρόσβαση στα περιφερειακά των γειτονικών συστημάτων.
- Διασφαλίζει την ακεραιότητα των shared δεδομένων, προλαμβάνοντας πιθανή αλλοίωσή τους.
- Αναπτύσει, μεταγλωττίζει, αποσφαλματώνει και εκτελεί κώδικα για ένα σύστημα πολυεπεξεργαστών χρησιμοποιώντας το περιβάλλον Nios II SBT για το Eclipse.

7.3 Πρόσθετη θεωρία

Κάθε σύστημα που ενσωματώνει πολλαπλούς μικροεπεξεργαστές οι οποίοι λειτουργούν παράλληλα για να υλοποιήσουν μια ή περισσότερες εργασίες ονομάζεται σύστημα πολυεπεξεργαστών. Χρησιμοποιώντας την πλακέτα της Terasic και τον Platform Designer μπορούμε εύκολα να σχεδιάσουμε ένα τέτοιο σύστημα.

Τα πολύ-επεξεργαστικά συστήματα έχουν το πλεονέκτημα της καλύτερης απόδοσης σε σχέση με τα μονοεπεξεργαστικά συστήματα, αλλά παράλληλα παρουσιάζουν αυξημένη πολυπλοκότητα, τόσο στο επίπεδο του υλικού όσο και στο επίπεδο του λογισμικού. Όσον αφορά τα ενσωματωμένα συστήματα, πολλαπλοί επεξεργαστές είναι πιθανό να χρησιμοποιηθούν όταν υπάρχει η ανάγκη διεκπεραίωσης πολύπλοκων διεργασιών σε πραγματικό χρόνο.

Ο Platform Designer επιτρέπει ιεραρχικό σχεδιασμό, μειώνοντας την πολυπλοκότητα ενός πολυεπεξεργαστικού συστήματος, διαχωρίζοντας ‘το σε διακριτά υποσυστήματα. Σε κάθε υποσύστημα δημιουργούνται, από το σχεδιαστή, διεπαφές, οι οποίες επιτρέπουν την ιεραρχική σύνδεση των υποσυστημάτων.

Το Nios II SBT για Eclipse περιλαμβάνει λειτουργίες που βοηθούν στην ανάπτυξη και αποσφαλμάτωση πολυ-επεξεργαστών. Οι πολλαπλοί επεξεργαστές Nios II είναι σε θέση να μοιράζονται αποτελεσματικά τα περιφερειακά, χάρη στις δυνατότητες που προσφέρει ο Platform Designer. Επειδή οι δυνατότητες του Platform Designer επιτρέπουν χωρίς ιδιαίτερο κόπο την προσθήκη πολυεπεξεργαστών σε ένα σύστημα, η πρόκληση αφορά κυρίως την ανάπτυξη του λογισμικού, έτσι ώστε να λειτουργεί αποτελεσματικά, χωρίς συγκρούσεις, στους επεξεργαστες για τους οποίους προορίζεται**.**

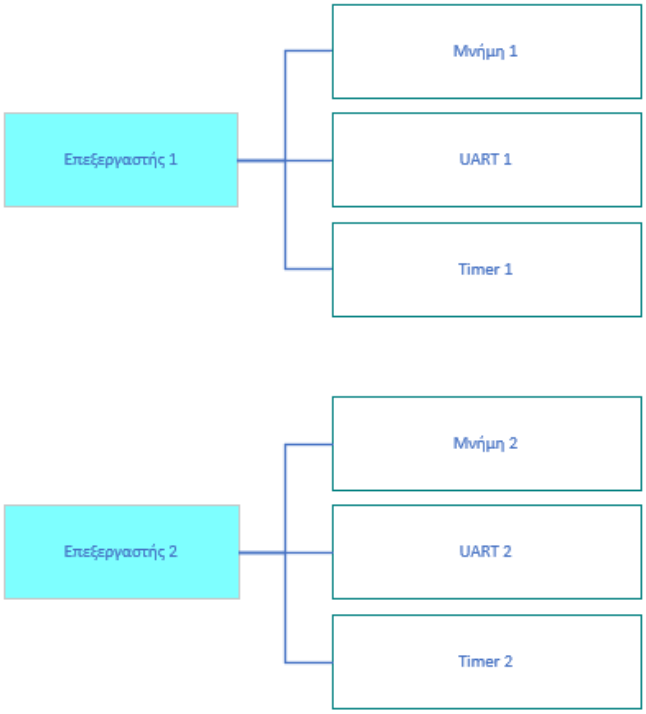
Για να αποφευχθεί η ταυτόχρονη προσπέλαση δύο ή περισσότερων επεξεργαστών σε κάποιο κρίσιμο τμήμα μνήμης, μία μονάδα υλικού αμοιβαίου αποκλεισμού (mutual exclusion - *mutex*) συμπεριλαμβάνεται στο Platform Designer. Η μονάδα mutex επιτρέπει σε διαφορετικούς επεξεργαστές να διεκδικήσουν ένα κοινόχρηστο περιφερειακό. Ο πρώτος εκ των επεξεργαστών που θα διεκδικήσει το περιφερειακό θα το χρησιμοποιήσει, ενώ οι υπόλοιποι θα περιμένουν μέχρι ο <<νικητής>> να ολοκληρώσει την εργασία του. Για την επίτευξη της λειτουργίας αυτής, θα πρέπει να αναπτυχθεί και το σχετικό λογισμικό.

7.3.1 Σχεδίαση περιφερειακού υλικού για κοινή χρήση

Τα συστήματα με πολλαπλούς επεξεργαστές Nios II χωρίζονται σε δύο κύριες κατηγορίες: στην πρώτη κάθε επεξεργαστής είναι αυτόνομος και δεν μοιράζεται τα περιφερειακά του με τους άλλους επεξεργαστές του συστήματος, ενώ στη δεύτερη υπάρχουν διαμοιραζόμενα περιφερειακά μεταξύ των επεξεργαστών.

7.3.1.1 Αυτόνομοι πολυεπεξεργαστές

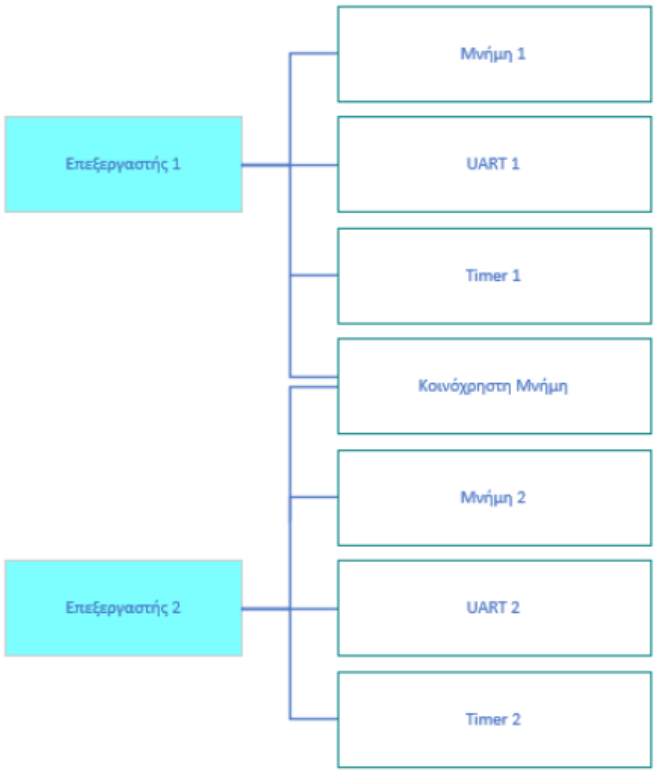
Ενώ τα συστήματα αυτά μπορούν να περιέχουν πολλαπλούς επεξεργαστές, αυτοί οι επεξεργαστές είναι εντελώς αυτόνομοι και δεν επικοινωνούν μεταξύ τους, σαν να είναι εντελώς ξεχωριστά συστήματα. Κατά τον σχεδιασμό, τα συστήματα αυτού του τύπου δεν μοιράζονται περιφερειακά, επομένως, τέτοια συστήματα είναι συνήθως λιγότερο περίπλοκα και θέτουν λιγότερες προκλήσεις κατά τη σχεδίαση τους. Η εικόνα 7.2 που ακολουθεί παρουσιάζει δύο αυτόνομους επεξεργαστές, οι οποίοι αποτελούν μέρη του ίδιου πολυεπεξεργαστικού συστήματος.



Εικόνα 7.2 Σύστημα πολλαπλών αυτόνομων επεξεργαστών [25]

7.3.1.2 Πολυεπεξεργαστές με διαμοιραζόμενα περιφερειακά

Τα συστήματα πολλαπλών επεξεργαστών που μοιράζονται περιφερειακά μπορούν να θέσουν πολλές προκλήσεις. Υπάρχουν λειτουργίες στο Platform Designer που καθιστούν δυνατή την ομαλή συνύπαρξη πολλαπλών επεξεργαστών που μοιράζονται περιφερειακά, ωστόσο όμως, η δημιουργία ενός τέτοιου συστήματος δεν είναι πάντα απλή. Η εικόνα 7.3 δείχνει ένα διάγραμμα ενός συστήματος πολλαπλών επεξεργαστών, στο οποίο οι δύο επεξεργαστές μοιράζονται μία μνήμη RAM.



Εικόνα 7.3 Σύστημα πολλαπλών επεξεργαστών με διαμοιραζόμενα περιφερειακά [25]

7.3.2 Κοινή χρήση μνήμης

Ο συνηθέστερος τύπος κοινόχρηστου περιφερειακού σε συστήματα πολλαπλών επεξεργαστών είναι η μνήμη. Η κοινή μνήμη μπορεί να χρησιμοποιηθεί για οποιαδήποτε χρήση, όπως για παράδειγμα την αποθήκευση μεταβλητών που δηλώνουν την κατάσταση του αντίστοιχου επεξεργαστή.

Εάν μία μονάδα μνήμης περιέχει τη μνήμη προγράμματος πολλών επεξεργαστών, κάθε επεξεργαστής που μοιράζεται τη μνήμη αυτή πρέπει να έχει την δική του ξεχωριστή περιοχή για εκτέλεση κώδικα. Οι επεξεργαστές δεν μπορούν να μοιράζονται την ίδια περιοχή μνήμης για αποθηκευτικό χώρο του προγράμματος. Κάθε πρόγραμμα πρέπει να έχει τα δικά του ξεχωριστά .text, .rodata, .rwdata, .heap, και .stack τμήματα.

Εάν μία μονάδα μνήμης πρέπει να μοιραστεί με σκοπό την αποθήκευση δεδομένων, πρέπει να συνδεθεί η θύρα slave της μονάδας αυτής με τις αντίστοιχες γραμμές δεδομένων των επεξεργαστών που μοιράζονται την κοινόχρηστη αυτή μνήμη. Σε ένα μη ιεραρχικό σύστημα, η σύνδεση γίνεται απευθείας, ενώ σε ένα ιεραρχικό σύστημα, η σύνδεση γίνεται μέσω της διεπαφής slave που έχει προσδιοριστεί για κάθε υποσύστημα.

Η κοινή χρήση μνήμης δεδομένων μεταξύ πολλών επεξεργαστών μπορεί να προκύψει δύσκολη υπόθεση, επειδή η μνήμη αυτή μπορεί να γραφτεί ενώ διαβάζεται. Εάν ένας επεξεργαστής γράφει σε μια συγκεκριμένη περιοχή κοινόχρηστης μνήμης και ένας άλλος επεξεργαστής διαβάζει ή γράφει σε αυτήν την περιοχή ταυτόχρονα, είναι πιθανό να προκύψει καταστροφή ή λάθος ανάγνωση δεδομένων, προκαλώντας σφάλματα στην εφαρμογή και ενδεχομένως κατάρρευση του συστήματος. Οι επεξεργαστές που μοιράζονται τη μνήμη χρειάζονται ένα μηχανισμό για να αλληλοενημερώνονται σχετικά με τη χρήση της μνήμης. Η ακόλουθη ενότητα περιγράφει έναν τέτοιο μηχανισμό: την μονάδα υλικού mutex της Intel.

7.3.3 Ο πυρήνας υλικού Mutex

Στα συστήματα που στηρίζονται στον επεξεργαστή Nios II παρέχεται προστασία των κοινόχρηστων περιφερειακών συσκευών με χρήση της μονάδας υλικού mutex. Ο όρος mutex σημαίνει αμοιβαίος αποκλεισμός, οπότε μία τέτοια μονάδα επιτρέπει στους συνεργαζόμενους επεξεργαστές να “συμφωνούν” ότι μόνο ένας επεξεργαστής κάθε φορά έχει πρόσβαση σε ένα διαμοιραζόμενο περιφερειακό.

Η μονάδα δίνει τη δυνατότητα λειτουργιών ατομικής δοκιμής και ρύθμισης, επιτρέποντας έτσι σε έναν επεξεργαστή να ελέγξει αν το διαμοιραζόμενο περιφερειακό είναι διαθέσιμο ή όχι. Όταν ο επεξεργαστής τελειώσει χρησιμοποιώντας το κοινόχρηστο περιφερειακό που σχετίζεται με το mutex, τότε ξεκλειδώνει το περιφερειακό αυτό. Στη συνέχεια, ένας άλλος επεξεργαστής μπορεί να αποκτήσει πρόσβαση στο διαμοιραζόμενο περιφερειακό αυτό.

Χωρίς το mutex, αυτή η λειτουργία θα απαιτούσε κανονικά από τον επεξεργαστή να εκτελέσει δύο ξεχωριστές λειτουργίες δοκιμής και ρύθμισης κατά την διάρκεια της οποίας ένας άλλος επεξεργαστής θα μπορούσε επίσης να ελέγξει για διαθεσιμότητα και να διεκδικήσει το περιφερειακό. Αυτή η κατάσταση θα άφηνε δύο επεξεργαστές να πιστεύουν ότι απέκτησαν αμοιβαία αποκλειστική πρόσβαση στο κοινόχρηστο περιφερειακό, ενώ στην πραγματικότητα δεν θα είχαν.

Η μονάδα υλικού mutex δεν έχει καμία σύνδεση με το διαμοιραζόμενο περιφερειακό. Αυτό που κάνει είναι να παρέχει απλώς τον μηχανισμό διαχείρισης μεταβλητών τύπου σημαιοφόρου (semaphore).

Πίνακας 7.1 Συναρτήσεις χειρισμού mutex

Συναρτήσεις	Περιγραφή
altera_avalon_mutex_open()	Δημιουργία επικοινωνίας με τη μονάδα mutex, επιτρέποντας σε όλες τις άλλες συναρτήσεις να τη χρησιμοποιήσουν.
altera_avalon_mutex_trylock()	Προσπάθεια κλειδώματος μίας μεταβλητής τύπου mutex. Επιστρέφει αμέσως εάν αποτύχει.
altera_avalon_mutex_lock()	Κλειδώνει μία μεταβλητή τύπου mutex. Δεν επιστρέφει μέχρι να το κλειδώσει επιτυχώς.
altera_avalon_mutex_unlock()	Ξεκλειδώνει μία μεταβλητή τύπου mutex.
altera_avalon_mutex_is_mine()	Διαπιστώνει εάν ο επεξεργαστής που την καλεί, είναι εκείνος που έχει κλειδώσει μία mutex μεταβλητή.
altera_avalon_mutex_first_lock()	Ελέγχει εάν μία mutex μεταβλητή έχει ελευθερωθεί μετά το πρώτο κλείδωμα.

7.3.4 Κοινή χρήση περιφερειακών

Η κοινή χρήση περιφερειακών σε συστήματα πολλαπλών επεξεργαστών παρουσιάζει μερικές δύσκολες προκλήσεις, όμως γενικά θεωρείται αποτελεσματική για τη σχεδίαση συστημάτων. Τα μεγαλύτερα προβλήματα αφορούν περιφερειακά που προκαλούν διακοπές (interrupts). Εάν επιτρέπεται σε μια περιφερειακή συσκευή να διακόψει όλους τους επεξεργαστές που τη μοιράζονται, δεν υπάρχει αξιόπιστος τρόπος για να εγγυηθούμε ποιος επεξεργαστής θα ανταποκριθεί και θα διακόψει την εκτέλεσή του.

Επιπλέον, εάν το περιφερειακό χρησιμοποιείται ως συσκευή εισόδου για πολλούς επεξεργαστές, καθίσταται δύσκολο να προσδιοριστεί ποιος επεξεργαστής συλλέγει τη δεδομένη είσοδο από τη συσκευή. Ενώ είναι λογικό ότι ένα πολύπλοκο σύστημα σημαφόρων θα μπορούσε να δημιουργηθεί για να χειριστεί αυτά τα σενάρια, ένα τέτοιο σύστημα είναι πέρα από τους στόχους της συγκεκριμένης διπλωματικής.

Η Intel συνιστά τα περιφερειακά μνήμης και τα περιφερειακά Mutex, αν και μπορούν να προσπελαστούν από πολλούς επεξεργαστές ενός συστήματος, να είναι προσβάσιμα μόνο από έναν επεξεργαστή του συστήματος αυτού. Εάν άλλοι επεξεργαστές απαιτούν τη χρήση ενός περιφερειακού, είναι καλύτερο να χρησιμοποιηθεί μία FIFO ή ένας buffer που προστατεύεται από mutex, για την επικοινωνία με έναν επεξεργαστή που είναι συνδεδεμένος με το περιφερειακό. Αυτός ο επεξεργαστής λειτουργεί ως εξυπηρετητής για τους άλλους επεξεργαστές που αιτούνται το περιφερειακό αυτό.

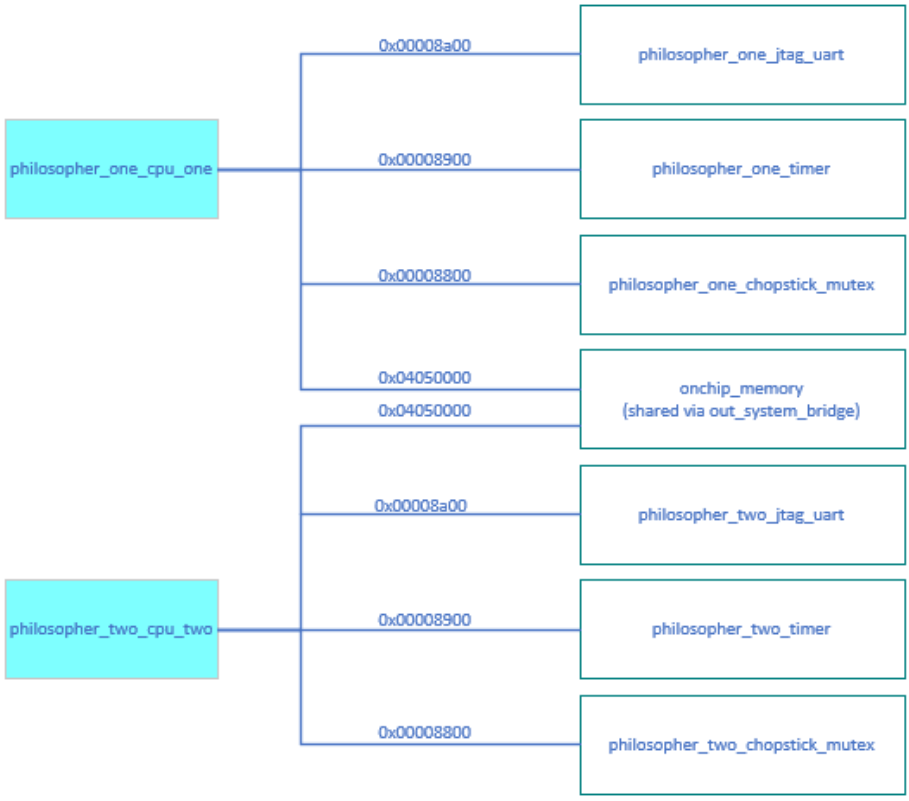
Σε συστήματα πολλαπλών επεξεργαστών, η ανάγκη σύνδεσης διαφόρων μονάδων εξαρτάται από τον σχεδιασμό. Επομένως, κατά τον σχεδιασμό συστημάτων πολλαπλών επεξεργαστών, σωστή πρακτική είναι να επαληθεύεται ρητά ότι κάθε μονάδα συνδέεται στον επιθυμητό επεξεργαστή.

Οι περισσότερες μονάδες είναι καλύτερα διαχειρίσιμες από έναν επεξεργαστή. Εάν ο επεξεργαστής Α απαιτεί τις υπηρεσίες μιας περιφερειακής μονάδας που είναι συνδεδεμένη και τη διαχειρίζεται ένας επεξεργαστής Β, ο επεξεργαστής Α πρέπει να ζητήσει από τον επεξεργαστή Β να εκτελέσει τις απαιτούμενες λειτουργίες για λογαριασμό του επεξεργαστή Α. Είναι σωστή πρακτική για τον σκοπό αυτό να χρησιμοποιούμε κοινή μνήμη που προστατεύεται από mutex για την επικοινωνία μεταξύ των δύο επεξεργαστών.

7.3.5 Επικάλυψη χώρου διευθύνσεων

Τα συστήματα ενός επεξεργαστή τυπικά απαγορεύουν περισσότερα από ένα περιφερειακά να καταλαμβάνουν την ίδια περιοχή διευθύνσεων, επειδή κάτι τέτοιο προκαλεί συγκρούσεις. Ωστόσο, στα συστήματα πολλαπλών επεξεργαστών, διαφορετικά περιφερειακά μπορούν να καταλαμβάνουν την ίδια διεύθυνση μνήμης χωρίς συγκρούσεις, αρκεί κάθε περιφερικό να ελέγχεται αποκλειστικά από διαφορετικό επεξεργαστή.

Εάν ο επεξεργαστής A είναι συνδεδεμένος με ένα περιφερειακό που αντιστοιχεί στη διεύθυνση 0x8a00, ο επεξεργαστής B μπορεί να συνδεθεί με ένα διαφορετικό περιφερειακό στην ίδια διεύθυνση 0x8a00, αν ο επεξεργαστής A δεν είναι συνδεδεμένος με τον επεξεργαστή B, και ο επεξεργαστής B δεν είναι συνδεδεμένος με το περιφερειακό του επεξεργαστή A. Στην πραγματικότητα, η αποκλειστική χρήση κάθε περιφερειακού από διαφορετικό επεξεργαστή, επιτρέπει στους δύο επεξεργαστές να έχουν διακριτούς χώρους διεύθυνσης. Η Εικόνα 7.4 δείχνει το διάγραμμα ενός συστήματος πολλαπλών επεξεργαστών με διαφορετικές περιφερειακές συσκευές που έχουν αντιστοιχηθεί στην ίδια διεύθυνση βάσης.



Εικόνα 7.4 Περιφερειακά πολλαπλών επεξεργαστών χαρτογραφημένα στην ίδια διεύθυνση βάσης [25]

7.3.6 Μνήμη προγράμματος

Η ανάπτυξη και η λειτουργία λογισμικού σε συστήματα πολλαπλών επεξεργαστών είναι σχεδόν η ίδια με εκείνη των συστημάτων με έναν επεξεργαστή, αλλά απαιτεί την εξέταση μερικών επιπλέον περιπτώσεων. Όταν δημιουργείται ένα σύστημα πολλαπλών επεξεργαστών, ίσως χρειαστεί να εκτελεστεί το λογισμικό για όλους τους επεξεργαστές από την ίδια μονάδα φυσικής μνήμης. Όμως, το λογισμικό για κάθε επεξεργαστή πρέπει να βρίσκεται στη δική του μοναδική περιοχή μνήμης.

Έστω, για παράδειγμα, ότι το λογισμικό για κάθε επεξεργαστή απαιτεί μνήμη 8 kbytes (KB) για τον κώδικα του προγράμματος και τα δεδομένα του. Ο πρώτος επεξεργαστής θα μπορούσε να χρησιμοποιήσει την περιοχή μεταξύ 0x0 και 0x1FFF μίας μνήμης ως χώρο προγράμματος, ενώ ένας άλλος επεξεργαστής θα μπορούσε να χρησιμοποιεί την περιοχή μεταξύ 0x2000 και 0x3FFF της ίδιας μνήμης.

Το Nios II SBT παρέχει ένα απλό σχήμα διαμέρισης μνήμης που επιτρέπει σε πολλούς επεξεργαστές να εκτελούν το λογισμικό τους από διαφορετικές περιοχές της ίδιας φυσικής μνήμης. Το SBT χρησιμοποιεί την Exception Address για κάθε επεξεργαστή, για να προσδιορίσει την περιοχή μνήμης από την οποία κάθε επεξεργαστής μπορεί να εκτελέσει τον κώδικα του. Ο σχεδιαστής του συστήματος ορίζει στον Platform Designer τη διεύθυνση εξαίρεσης για κάθε επεξεργαστή ξεχωριστά.

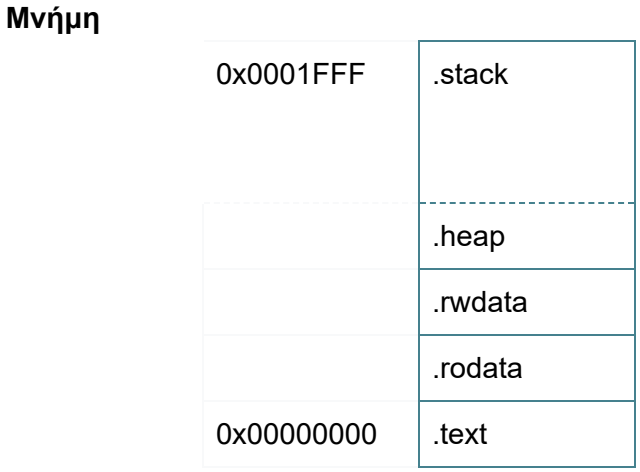
Παράλληλα, το Nios II SBT καθορίζει το πού βρίσκεται το λογισμικό κάθε επεξεργαστή στη μνήμη. Χρησιμοποιεί τις exception addresses και τοποθετεί την περιοχή κώδικα κάθε επεξεργαστή στη μονάδα μνήμης που περιέχει τη διεύθυνση εξαίρεσης.

Εάν το λογισμικό πολλαπλών επεξεργαστών πρέπει να τοποθετηθεί στην ίδια μονάδα φυσικής μνήμης, τότε το SBT χρησιμοποιεί κάθε επεξεργαστή για να καθορίσει τις διευθύνσεις της περιοχής προγράμματος του καθενός. Η περιοχή προγράμματος τελειώνει στη διεύθυνση της επόμενης exception address για κάθε διαφορετικό επεξεργαστή. Στον επεξεργαστή με τη μεγαλύτερη exception address αντιστοιχίζεται περιοχή κώδικα που εκτείνεται μέχρι το τέλος της περιοχής διευθύνσεων της φυσικής μνήμης.

Η μνήμη προγράμματος κάθε επεξεργαστή χωρίζεται σε πέντε τμήματα:

- **.text**: Ο εκτελέσιμος κώδικας
- **.rodata**: Όλα τα δεδομένα μόνο για ανάγνωση που χρησιμοποιούνται κατά την εκτέλεση του κώδικα
- **.rwdata**: Όπου αποθηκεύονται μεταβλητές ανάγνωσης-εγγραφής και δείκτες
- **.heap**: Περιοχή για δυναμική δέσμευση μνήμης
- **.stack**: Όπου αποθηκεύονται παράμετροι κλήσης συναρτήσεων και άλλα προσωρινά δεδομένα

Στην Εικόνα 7.5 φαίνεται πως τοποθετούνται τα τμήματα αυτά στον χώρο διευθύνσεων ενός επεξεργαστή Nios II:

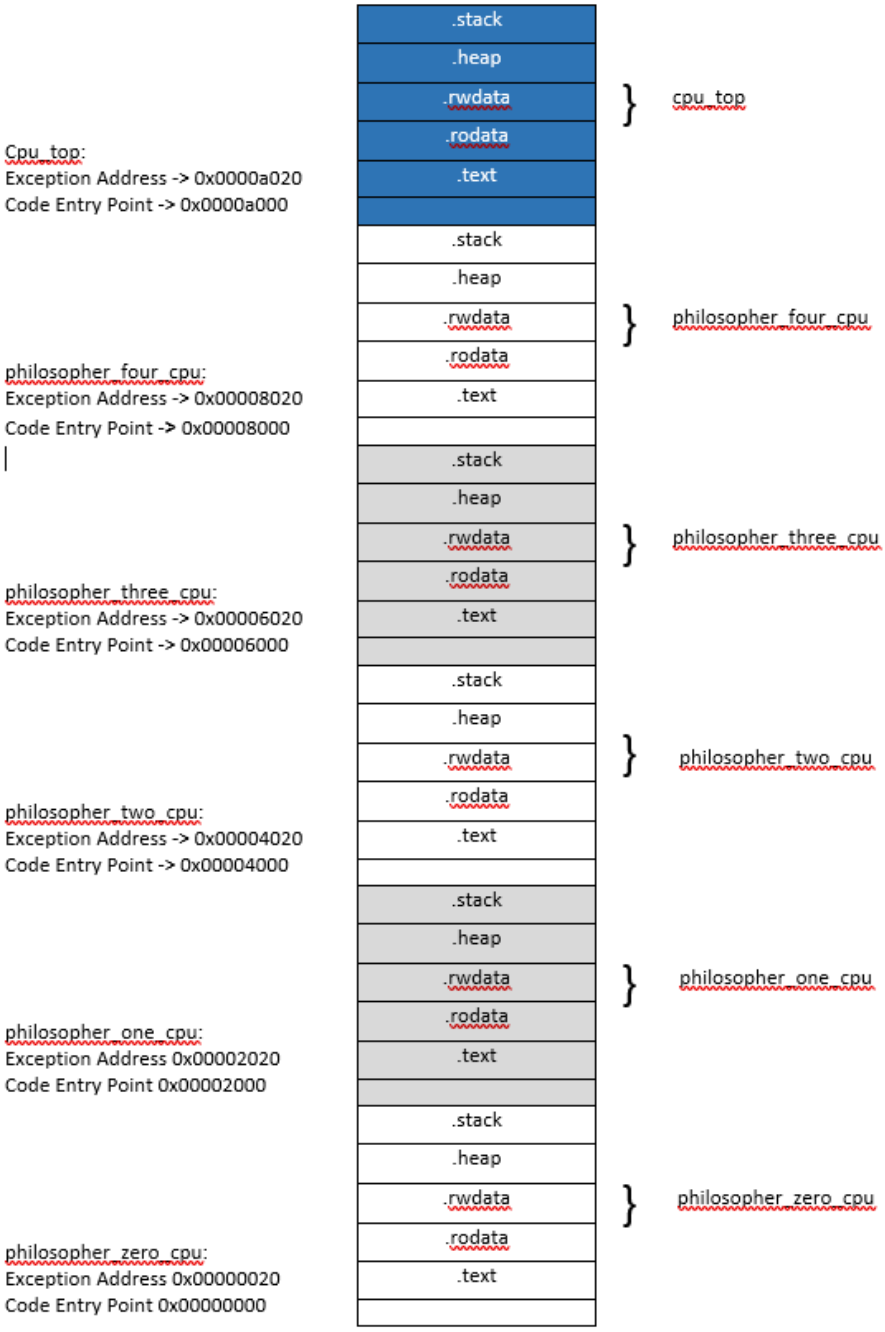


Εικόνα 7.5 Μνήμη προγράμματος ενός επεξεργαστή Nios II [25]

Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορεί να είναι πλεονεκτική η χρήση μίας μνήμης για την αποθήκευση όλων των τμημάτων κώδικα για κάθε επεξεργαστή. Σε αυτή την περίπτωση, η exception address που έχει οριστεί για κάθε επεξεργαστή στον Platform Designer χρησιμοποιείται για τον καθορισμό των ορίων μεταξύ της μνήμης προγράμματος ενός επεξεργαστή και της μνήμης προγράμματος του επόμενου επεξεργαστή. Για παράδειγμα, ακολουθεί ένα σύστημα, στο οποίο καταλαμβάνονται οι ακόλουθες περιοχές μνήμης:

- 0x00050000 to 0x0005FFFF - cpu_top επεξεργαστής
- 0x04000000 to 0x0405FFFF - επεξεργαστής σε οποιουδήποτε υποσυστήματος φιλοσόφου

Ο επεξεργαστής cpu_top και οι επεξεργαστές σε κάθε υποσύστημα φιλόσοφου έχουν ο καθένας οκτώ KB μνήμης για να εκτελέσουν το λογισμικό τους. Στην Εικόνα 7.6 βλέπουμε έναν χάρτη μνήμης που δείχνει πώς κατανέμεται η μνήμη RAM σε αυτό το παράδειγμα συστήματος.



Εικόνα 7.6 Κατανομή μνήμης OnChip για έξι επεξεργαστές [25]

7.4 Υποδείξεις

1. Δημιουργούμε πέντε διαφορετικά αρχεία συστήματος '.qsys' για τον Platform Designer, τα οποία αντιστοιχούν το ένα για το συντονιστή, και τα υπόλοιπα τέσσερα για τους φιλοσόφους, τα οποία θα πρέπει να περιλαμβάνουν τις ακόλουθες μονάδες:

- Clock Source
- NIOS II processor (έκδοση e)
    - Στην καρτέλα "Vector" στα πεδία "Reset vector memory" και "Exception vector memory" επιλέγουμε από την λίστα "<Όνομα μονάδας RAM>.s1"
- On-Chip Memory RAM μεγέθους 40K.

Τα τέσσερα συστήματα που αντιστοιχούν στους φιλοσόφους θα πρέπει να περιλαμβάνουν ξεχωριστά του συντονιστή:

- Γέφυρα Avalon-MM Pipeline

Το σύστημα που αντιστοιχεί στον συντονιστή θα πρέπει να περιλαμβάνει ξεχωριστά των φιλοσόφων:

- Τα τέσσερα συστήματα των φιλοσόφων, ως υποσυστήματα.
    - Για την αναζήτηση των υποσυστημάτων αυτών χρησιμοποιούμε το παράθυρο αναζήτησης "IP Catalog", και τα εισάγουμε παρομοίως με ένα οποιοδήποτε άλλο component, όπως αντίστοιχος είναι επίσης και ο τρόπος σύνδεσης των εισόδων και των εξόδων τους με το υπόλοιπο σύστημα.
- System ID Peripheral
- Interval Timer εύρους 32 bit
- Parallel I/O (PIO) εύρους 4 bit, ορισμένο σαν έξοδος για 4 LED
- Parallel I/O (PIO) εύρους 4 bit, ορισμένο σαν είσοδος 4 διακόπτες
- JTAG UART
- Mutex

2. Χρησιμοποιήστε τις οδηγίες της Άσκησης 0 για να δημιουργήσετε το "project directory", να ορίσετε ως όνομα του project το "philosophers", και να δημιουργήσετε τα αρχεία "philosophers_pin_assignments.csv" και "philosophers.sdc".

3. Για να χρησιμοποιήσετε το MUTEX που προσφέρει η Intel για συγχρονισμό των επεξεργαστών Nios II, κάντε "#include" τη βιβλιοθήκη "altera_avalon_mutex.h".

4. Φροντίστε να τυπώνετε ένα μήνυμα καλωσορίσματος κατά την εκτέλεση του προγράμματός σας για επιβεβαίωση της έναρξης λειτουργίας του μετά το "κατέβασμα" του λογισμικού σας στην πλακέτα, είτε κάποιο σχετικό μήνυμα με την εκάστοτε κατάσταση του κάθε φιλοσόφου.

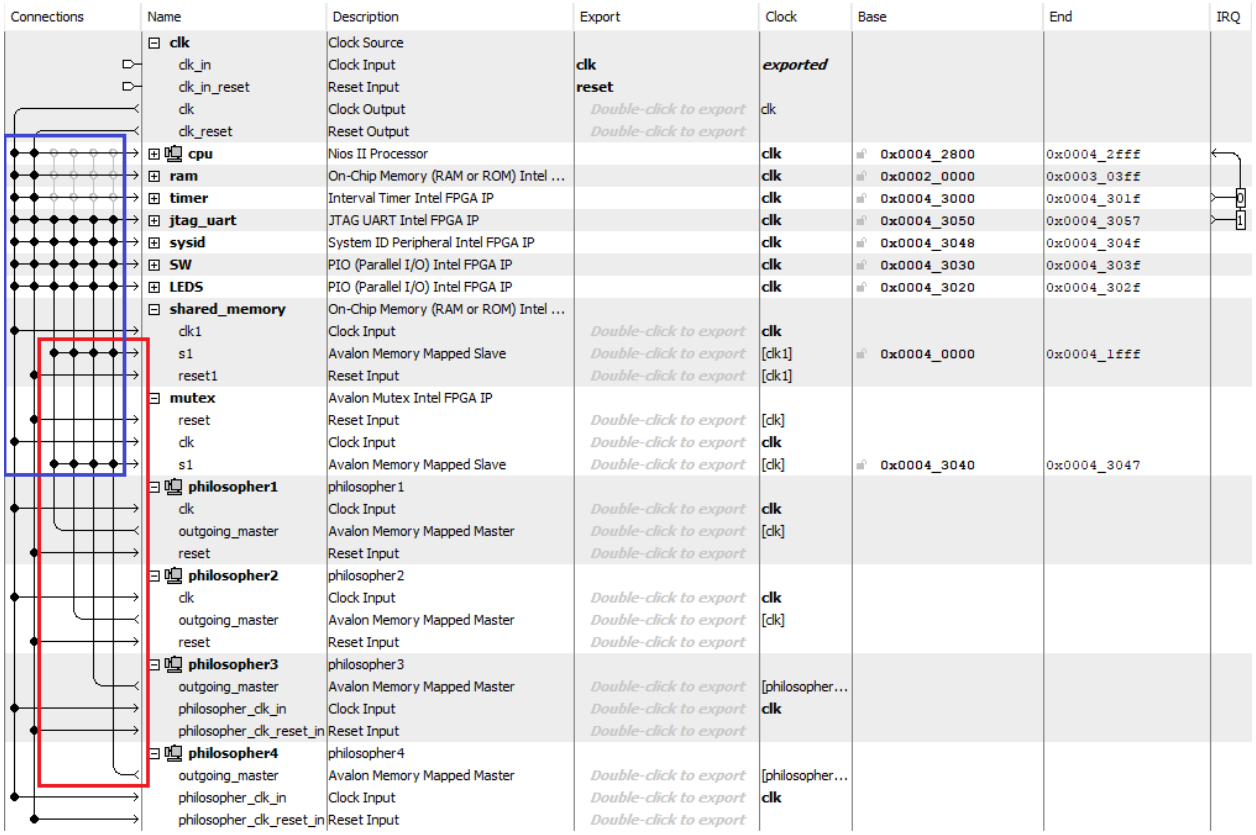
7.5 Ενδεικτική λύση

Ακολουθούν οι ενδεικτικές διαμορφώσεις των συστημάτων της Άσκησης 4. Κάθε ένα από τα υποσυστήματα των φιλοσόφων έχει την ακόλουθη διαμόρφωση της Εικόνας 7.7α:

Name	Description	Export	Clock	Base	End	IRQ
clk	Clock Source		exported			
cpu	Nios II Processor		clk	0x0802_0800	0x0802_0fff	
ram	On-Chip Memory (RAM or ROM) Intel ...		clk	0x0801_0000	0x0801_9fff	
out_system_bridge	Avalon-MM Pipeline Bridge		clk	0x0400_0000	0x07ff_ffff	

Εικόνα 7.7α Ενδεικτική διαμόρφωση υπο-συστήματος “φιλοσόφου” Άσκησης 4

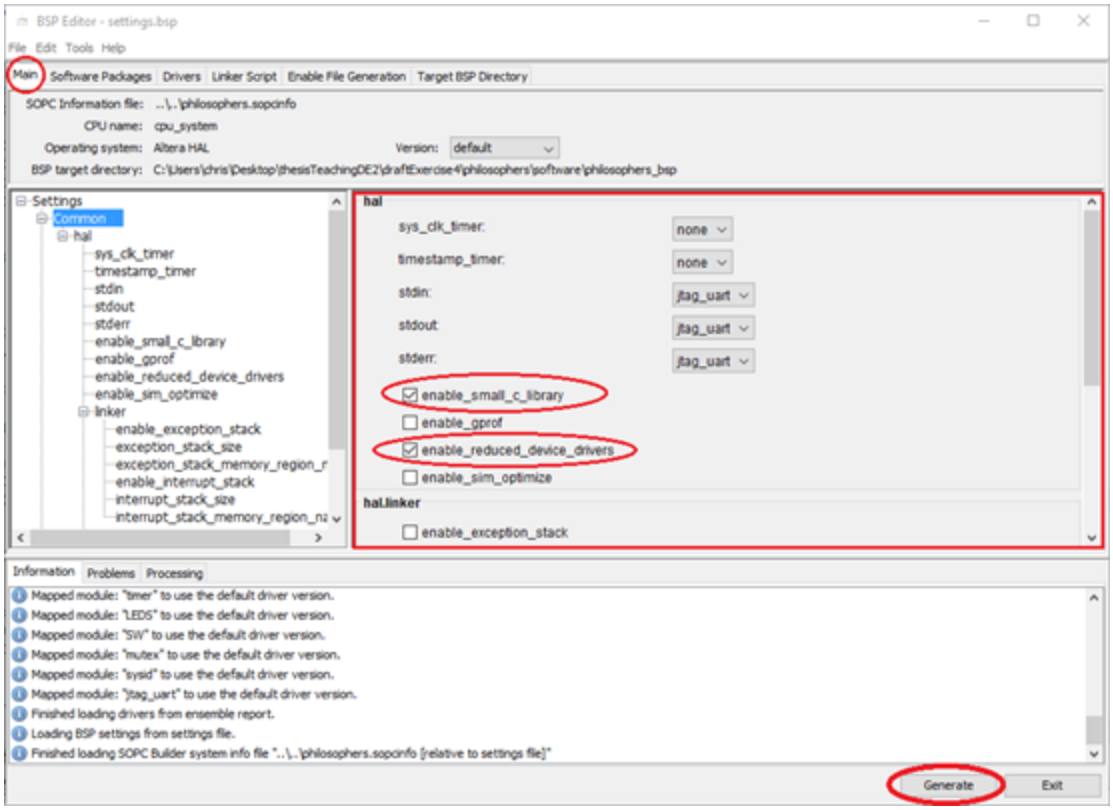
Στην ακόλουθη Εικόνα 7.7β τονίζεται με κόκκινο κουτί η συνδεσμολογία των υποσυστημάτων των φιλοσόφων με τη μονάδα Mutex, καθώς και με τη κοινόχρηστη μνήμη RAM του συστήματος του συντονιστή, στο οποίο επίσης εμπεριέχονται και όλα τα υπόλοιπα components του τελικού συστήματος, τα οποία έχουν τονιστεί με το μπλε κουτί:



Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double-click to export	clk			
	clk_reset	Reset Output	Double-click to export				
	cpu	Nios II Processor		clk	0x0004_2800	0x0004_2fff	
	ram	On-Chip Memory (RAM or ROM) Intel ...		clk	0x0002_0000	0x0003_03ff	
	timer	Interval Timer Intel FPGA IP		clk	0x0004_3000	0x0004_301f	0
	jtag_uart	JTAG UART Intel FPGA IP		clk	0x0004_3050	0x0004_3057	1
	sysid	System ID Peripheral Intel FPGA IP		clk	0x0004_3048	0x0004_304f	
	SW	PIO (Parallel I/O) Intel FPGA IP		clk	0x0004_3030	0x0004_303f	
	LEDS	PIO (Parallel I/O) Intel FPGA IP		clk	0x0004_3020	0x0004_302f	
	shared_memory	On-Chip Memory (RAM or ROM) Intel ...					
	clk1	Clock Input	Double-click to export	clk			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0004_0000	0x0004_1fff	
	reset1	Reset Input	Double-click to export	[clk1]			
	mutex	Avalon Mutex Intel FPGA IP					
	reset	Reset Input	Double-click to export	[clk]			
	clk	Clock Input	Double-click to export	clk			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_3040	0x0004_3047	
	philosopher1	philosopher1					
	clk	Clock Input	Double-click to export	clk			
	outgoing_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	reset	Reset Input	Double-click to export				
	philosopher2	philosopher2					
	clk	Clock Input	Double-click to export	clk			
	outgoing_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	reset	Reset Input	Double-click to export				
	philosopher3	philosopher3					
	outgoing_master	Avalon Memory Mapped Master	Double-click to export	[philosopher...			
	philosopher_clk_in	Clock Input	Double-click to export	clk			
	philosopher_clk_reset_in	Reset Input	Double-click to export				
	philosopher4	philosopher4					
	outgoing_master	Avalon Memory Mapped Master	Double-click to export	[philosopher...			
	philosopher_clk_in	Clock Input	Double-click to export	clk			
	philosopher_clk_reset_in	Reset Input	Double-click to export				

Εικόνα 7.7β Ενδεικτική διαμόρφωση συστήματος “συντονιστή” Άσκησης 4

Για το επιτυχές “building” του project θα πρέπει σε κάθε “<όνομα project>_bsp” που δημιουργήθηκε να γίνουν κάποιες ρυθμίσεις για περιορισμό της χρησιμοποιούμενης μνήμης από τον κάθε Nios. Κάνουμε δεξί κλικ πάνω σε κάθε “<όνομα project>_bsp” και επιλέγουμε “NIOS II” -> “BSP Editor...”. Στο παράθυρο που θα μας ανοίξει στη καρτέλα “Main” τσεκάρουμε τις επιλογές “enable_small_c_library” και “enable_reduced_device_drivers” από το πεδίο “hal” και πατάμε “Generate” (Εικόνα 7.8).



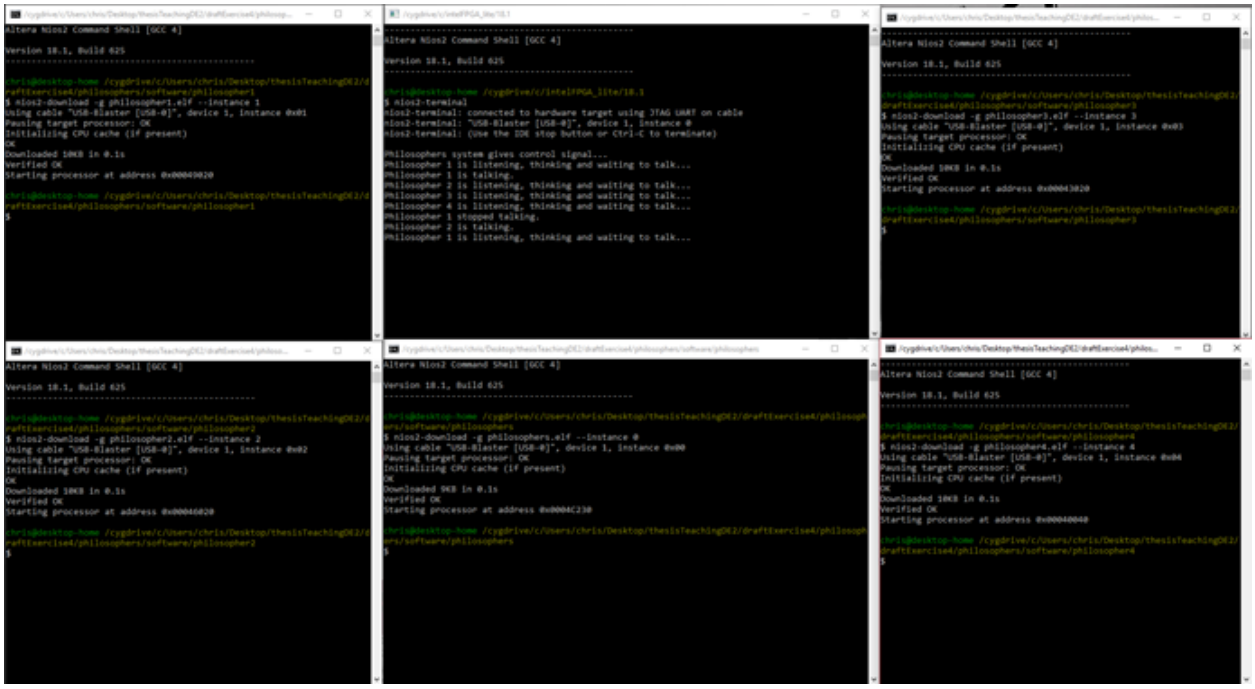
Εικόνα 7.8 Περιορισμός μεγέθους βιβλιοθηκών της Embedded C

Για την εκτέλεση του σεναρίου υπάρχουν δύο τρόποι. Ο πρώτος (προτεινόμενος για λόγους κατανόησης της άσκησης) είναι να χρησιμοποιηθεί το “NIOS II Command Shell”. Οι εντολές που θα πρέπει να εκτελεστούν σε κάθε τερματικό, αφού ανοιχθούν συνολικά έξι τερματικά είναι οι εξής:

1. nios2-terminal
2. nios2-download -g philosophers.elf --instance 0
3. nios2-download -g philosopher1.elf --instance 1
4. nios2-download -g philosopher2.elf --instance 2
5. nios2-download -g philosopher3.elf --instance 3
6. nios2-download -g philosopher4.elf --instance 4

Το πρώτο τερματικό θα πρέπει να ανοιχθεί πατώντας το κουμπί “Start” των Windows -> “Programs->Intel FPGA 18.1.XYZ Lite Edition->NIOS II EDS 18.1.XYZ->NIOS II Command Shell (Quartus Prime 18.1)” (Εικόνα 7.9), στο οποίο και θα εκτελεστεί πρώτα το πρόγραμμα του συντονιστή. Θα πρέπει για την επιτυχή εκτέλεση του προγράμματος, το παραγόμενο philosophers.sof αρχείο να έχει ήδη ανέβει μέσω του “Quartus Prime Programmer” και η πλακέτα να είναι σε κατάσταση αναμονής.

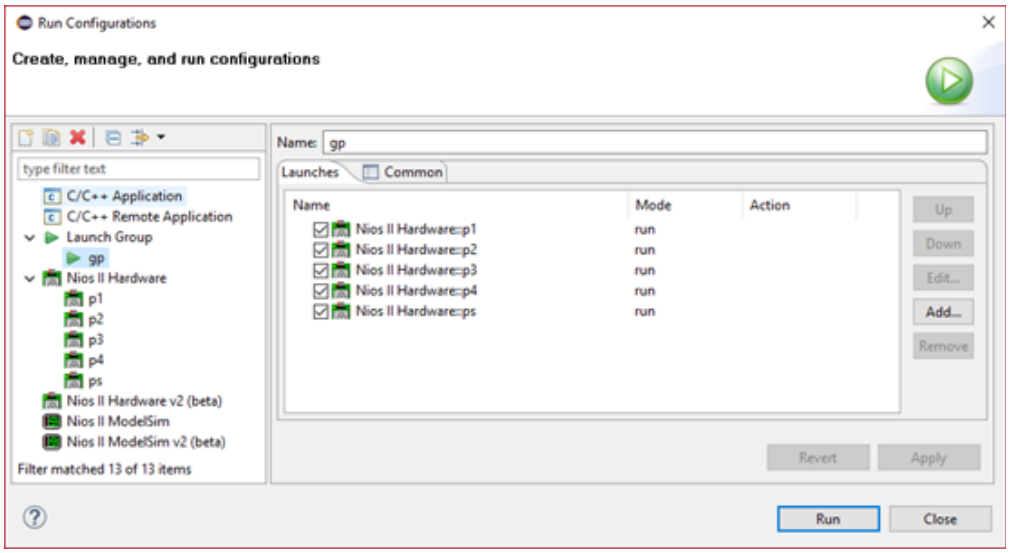
Τα υπόλοιπα πέντε τερματικά θα πρέπει να ανοιχθούν μέσα από το Eclipse Nios II EDS πατώντας δεξί κλικ πάνω στο εκάστοτε project -> “NIOS II->NIOS II Command Shell…”, σε κάθε ένα από τα οποία θα τρέξουμε το πρόγραμμα του αντίστοιχου φιλοσόφου (Εικόνα 7.9).



Εικόνα 7.9 Παράδειγμα εκτέλεσης της άσκησης χρησιμοποιώντας τον πρώτο τρόπο

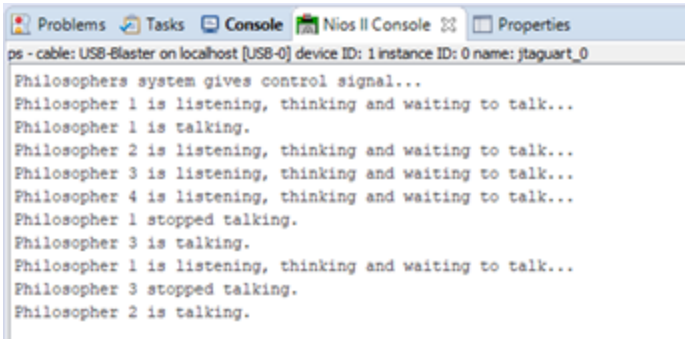
Ο δεύτερος τρόπος είναι δημιουργώντας ένα “Launch Configuration Group” (Εικόνα 7.11). Πατώντας το κουμπί “Run” από τη γραμμή μενού και έπειτα επιλέγοντας “Run Configurations…” στο Eclipse, θα ανοίξει ένα παράθυρο στο οποίο μπορούμε να δημιουργήσουμε μία “ομάδα διαμόρφωσης” της πλακέτας στο Eclipse, η οποία μας επιτρέπει να «τρέξουμε» όλα τα project μαζί. Κάνοντας πέντε φορές διπλό κλικ πάνω στο πεδίο “Nios II Hardware” της αριστερής στήλης δημιουργούμε πέντε διαφορετικές ομάδες από “Configuration Settings”.

Θα πρέπει να επιλέξουμε για το καθένα, ένα όνομα διαφορετικό του project, και προσέχουμε να επιλέξουμε σωστά το εκάστοτε project στο οποίο αντιστοιχούν οι ρυθμίσεις στην καρτέλα “Target Connection”. Έπειτα, κάνοντας διπλό κλικ στο “Launch Group” δημιουργούμε την «ομάδα διαμόρφωσης» επιλέγοντας όλα τα “Configuration Settings” που δημιουργήσαμε.



Εικόνα 7.10 Δημιουργία ομάδας διαμόρφωσης

Ακολουθεί ένα παράδειγμα εκτέλεσης του σεναρίου (Εικόνα 7.12) με χρήση του δεύτερου τρόπου, και εκτύπωση μηνυμάτων στην κονσόλα του Eclipse.



Εικόνα 7.11 Ενδεικτικό αποτέλεσμα εκτέλεσης του προγράμματος των φιλοσόφων

Στο Παράρτημα Γ επισυνάπτονται για το κάθε υποσύστημα φιλοσόφου, και του συστήματος του συντονιστή, η προτεινόμενη υλοποίηση κώδικα, καθώς και η αντιστοίχιση των ακροδεκτών, της Άσκησης 4. Στην ακόλουθη Εικόνα 7.12 απεικονίζεται η δομή που προτείνεται για κάθε υποσύστημα φιλοσόφου, όσον αφορά τον έλεγχο της συσκευής του Mutex. Στην αρχή, η ροή του κώδικα αναμένει την εντολή έναρξης από το συντονιστή, και έπειτα προσπαθεί να κλειδώσει τη συσκευή Mutex, δηλαδή να “πάρει” τον έλεγχο των κοινόχρηστων περιοχών. Οι κοινόχρηστες περιοχές για τις οποίες πρέπει να χρησιμοποιήσουμε το Mutex είναι τα LED, καθώς και η κοινόχρηστη μνήμη RAM.

Σύμφωνα με ορισμό της άσκησης, κατά τη διάρκεια που ένας φιλόσοφος μιλάει αναβοσβήνει μόνο το led που του έχει αντιστοιχηθεί, οπότε κανείς από τους υπόλοιπους φιλοσόφους δεν μπορεί να έχει πρόσβαση σε αυτή τη συσκευή PIO όσο διαρκεί αυτή η διαδικασία. Επιπλέον, μόλις κάποιο από τα συστήματα των φιλοσόφων κλειδώσει το Mutex, αποκτά και τον έλεγχο της κοινόχρηστης μνήμης RAM, την οποία κάθε σύστημα ενημερώνει όταν ολοκληρώσει την διαδικασία που εκτελούσε, και μέσω της οποίας ο συντονιστής ελέγχει ποιο από τα υποσυστήματα των φιλοσόφων έχει δικαίωμα να διεκδικήσει τον έλεγχο των κοινόχρηστων περιοχών σε κάθε κύκλο.

```
int main(){

    // Get the mutex device handle
    alt_mutex_dev *mutex = altera_avalon_mutex_open("/dev/mutex");

    while(1){

        ...

        // Waiting for control signal
        while(IORD(OnChip,0x01) != ALT_CPU_CPU_ID_VALUE);

        // Printing Data
        printf("Philosopher %d is listening, thinking and waiting to talk... \n", ALT_CPU_CPU_ID_VALUE);

        // Locking MUTEX
        while(altera_avalon_mutex_trylock(mutex, ALT_CPU_CPU_ID_VALUE));

        printf("Philosopher %d is talking. \n", ALT_CPU_CPU_ID_VALUE );

        while(...){
            ...
        }// for

        // Reset Philosopher's status
        printf("Philosopher %d stopped talking. \n", ALT_CPU_CPU_ID_VALUE );

        ...

        // Unlock MUTEX
        altera_avalon_mutex_unlock(mutex);

    }// while

    return 0;
}// main
```

Εικόνα 7.12 Δομή κώδικα για τον έλεγχο της συσκευής Mutex

Συμπεράσματα

Δεδομένης της πανταχού παρουσίας ενσωματωμένων συστημάτων είναι σαφές ότι υπάρχει μεγάλη ανάγκη για μαθήματα ενσωματωμένων συστημάτων, ειδικά για τους φοιτητές και φοιτήτριες της επιστήμης και της μηχανικής των υπολογιστών. Σε αυτή τη διπλωματική εργασία περιγράψαμε μία πρακτική προσέγγιση, η οποία στηρίζεται στους ακόλουθους κανόνες:

- Ένας σχεδιαστής συστημάτων θα πρέπει να κατανοεί τις βασικές έννοιες και να έχει αποκτήσει τις δεξιότητες σχετικά με τον σχεδιασμό συστημάτων.
- Τα θεωρητικά μαθήματα πρέπει να ενισχύονται με σχετικά πρακτικά μέρη.
- Είναι σημαντικό αυτά τα πρακτικά μέρη να παρέχουν μία βήμα προς βήμα μαθησιακή προσέγγιση στους φοιτητές, αντί για μία, μεγαλύτερη ανάθεση εργασίας.
- Το υλικό του μαθήματος πρέπει να ενημερώνεται σε τακτά χρονικά διαστήματα, ώστε να περιλαμβάνει νέες τεχνολογικές εξελίξεις, να εφαρμόζει νέες διδακτικές πρακτικές, και να επιδιώκεται η ανατροφοδότηση σχολίων από τους φοιτητές και φοιτήτριες ως αξιολόγηση, για τη βελτιστοποίηση της εκπαιδευτικής εμπειρίας.

Παράρτημα Α

Κώδικας C Άσκησης 2

```
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "string.h"
#include "math.h"

#include "priv/alt_legacy_irq.h"
#include "altera_up_avalon_character_lcd.h"

#define TIMER_STAT_REG 0 // status register address offset
#define TIMER_CTRL_REG 1 // control register address offset
#define TIMER_PRDL_REG 2 // period reg (lower 16 bits) addr offset
#define TIMER_PRDH_REG 3 // period reg (upper 16 bits) addr offset
#define ENABLE_TIMER 7 // configure timer to start, continuous mode, enable interrupt (0111 = 7)
#define NO_DATA        0 // No initial drive values

#define UP    7
#define DOWN  11
#define LEFT  13
#define RIGHT 14

#define SAVE     7
#define RST_LCD  11
#define RST_SSEG 13
#define CALC     14

#define LCD_WR_COMMAND_REG 0

int clock_ticks = 0, change = 1, sseg_flash = 1, selected_piece=1;

//convert binary display of numbers to hex numbers for 7-segment display
alt_u8 numbers[11] = {
            0x040, // 7'b1000000 number 0
            0x079, // 7'b1111001 number 1
            0x024, // 7'b0100100 number 2
            0x030, // 7'b0110000 number 3
            0x019, // 7'b0011001 number 4
            0x012, // 7'b0010010 number 5
            0x002, // 7'b0000010 number 6
            0x078, // 7'b1111000 number 7
            0x000, // 7'b0000000 number 8
            0x018, // 7'b0011000 number 9
            0x07f, // Off
};

void initial_message(){
   printf("***************************\n");
   printf("*   Hello from Nios II!  *\n");
   printf("***************************\n\n");
}// initial_message

int getCommand(alt_u32 base, int addr){
            return IORD(base, addr) & 0x0f;
}// getCommand

void init_timer(alt_u32 timer_base, alt_u32 period){
            alt_u16 high, low;
```

```
	//unpack 32-bit timeout period into two 16-bit half words
	high = (alt_u16) (period >> 16);
	low = (alt_u16) (period & 0x0000ffff);

	//write timeout period
	IOWR(timer_base, TIMER_PRDH_REG, high);
	IOWR(timer_base, TIMER_PRDL_REG, low);

	//configure timer to start, continuous mode, enable interrupt (0111 = 7)
	IOWR(timer_base, TIMER_CTRL_REG, ENABLE_TIMER);
}

void timer_isr(void* context, alt_u32 id){
	//clear timer's status register to (timeout) bit (bit 0) to start the timer over again
	IOWR(TIMER_BASE, TIMER_STAT_REG, NO_DATA);

	if(clock_ticks == 999) clock_ticks = 0;
	else clock_ticks++;
}

int temp_year[] = {2,0,1,9}, temp_month[] = {0,1}, temp_date[] = {0,1};

int curr_year = 2019, curr_month = 1, curr_date = 1;
int byear     = 1991, bmonth     = 5, bdate     = 23;

char curr_age[16], birth_d[16], curr_d[16];

// function to calculate current age
void age_calculator(
	int current_date, int current_month, int current_year,
	int birth_date,   int birth_month,   int birth_year
){
	// days of every month
	int month[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

	// if birth date is greater then current birth
	// month then do not count this month and add 30
	// to the date, so to subtract the date and
	// get the remaining days
	if (birth_date > current_date) {
		current_date = current_date + month[birth_month - 1];
		current_month = current_month - 1;
	}

	// if birth month exceeds current month, then do
	// not count this year and add 12 to the month so
	// that we can subtract and find out the difference
	if (birth_month > current_month) {
		current_year = current_year - 1;
		current_month = current_month + 12;
	}

	// calculate date, month, year
	int calculated_date = current_date - birth_date;
	int calculated_month = current_month - birth_month;
	int calculated_year = current_year - birth_year;

	if( calculated_year < 0 || calculated_month < 0 || calculated_date < 0 ) {

		printf ("ERROR : Invalid dates inserted. \n\n");
		sprintf(curr_age, "%dY / %dM / %dD", 0, 0, 0);
	} else {
		sprintf(curr_age, "%dY / %dM / %dD", calculated_year, calculated_month, calculated_date);
	}
```

```
		//printf("curr_age : %s\n", curr_age);
}// age_calculator

//declare LCD device
alt_up_character_lcd_dev * char_lcd_dev;

void printToLCD(char line1[], char line2[]){

		// Display clear
		IOWR(LCD_BASE, LCD_WR_COMMAND_REG, 0x01);

		alt_up_character_lcd_set_cursor_pos(char_lcd_dev, 0, 0);
		alt_up_character_lcd_string(char_lcd_dev, line1);
		alt_up_character_lcd_set_cursor_pos(char_lcd_dev, 0, 1);
		alt_up_character_lcd_string(char_lcd_dev, line2);

}// printToLCD

void display_sseg(int enable, int selected_piece){
		if(enable==1){
			IOWR(SSEG_0_BASE, 0, numbers[temp_year[3]]);
			IOWR(SSEG_1_BASE, 0, numbers[temp_year[2]]);
			IOWR(SSEG_2_BASE, 0, numbers[temp_year[1]]);
			IOWR(SSEG_3_BASE, 0, numbers[temp_year[0]]);
			IOWR(SSEG_4_BASE, 0, numbers[temp_month[1]]);
			IOWR(SSEG_5_BASE, 0, numbers[temp_month[0]]);
			IOWR(SSEG_6_BASE, 0, numbers[temp_date[1]]);
			IOWR(SSEG_7_BASE, 0, numbers[temp_date[0]]);
		} else {
			switch(selected_piece){
			case 1:
				IOWR(SSEG_6_BASE, 0, numbers[10]); // Off
				IOWR(SSEG_7_BASE, 0, numbers[10]); // Off
				break;
			case 2:
				IOWR(SSEG_4_BASE, 0, numbers[10]); // Off
				IOWR(SSEG_5_BASE, 0, numbers[10]); // Off
				break;
			case 3:
				IOWR(SSEG_0_BASE, 0, numbers[10]); // Off
				IOWR(SSEG_1_BASE, 0, numbers[10]); // Off
				IOWR(SSEG_2_BASE, 0, numbers[10]); // Off
				IOWR(SSEG_3_BASE, 0, numbers[10]); // Off
				break;
			}// switch
		}// if
}// display_sseg

void reset_date(){

		int reset_year[]  = {2,0,1,9};
		int reset_month[] = {0,1};
		int reset_date[]  = {0,1};

		memcpy(temp_year, reset_year, sizeof temp_year);
		memcpy(temp_month, reset_month, sizeof temp_month);
		memcpy(temp_date, reset_date, sizeof temp_date);

}// reset_date

void save_mode(int mode, int command){

		switch(command){
		case SAVE :

			if (mode == 0){
```

```
                                curr_year = 0, curr_month = 0, curr_date = 0;
                                int i;

                                for (i = 0; i < 4; i++)
                                        curr_year = 10 * curr_year + temp_year[i];

                                for (i = 0; i < 2; i++)
                                        curr_month = 10 * curr_month + temp_month[i];

                                for (i = 0; i < 2; i++)
                                        curr_date = 10 * curr_date + temp_date[i];

                                //printf("INFO  : Current Day : %d / %d / %d\n", curr_date, curr_month, curr_year);
                                sprintf(curr_d, "%d / %d / %d", curr_date, curr_month, curr_year);

                                printToLCD("Current Day :", curr_d);
                        } else if(mode == 2){

                                byear = 0, bmonth = 0, bdate = 0;
                                int i;

                                for (i = 0; i < 4; i++)
                                        byear = 10 * byear + temp_year[i];

                                for (i = 0; i < 2; i++)
                                        bmonth = 10 * bmonth + temp_month[i];

                                for (i = 0; i < 2; i++)
                                        bdate = 10 * bdate + temp_date[i];

                                //printf("INFO  : Birth   Day : %d - %d - %d\n", bdate, bmonth, byear);
                                sprintf(birth_d, "%d / %d / %d", bdate, bmonth, byear);

                                printToLCD("Birth Day :", birth_d);
                        }// if

                        break;
                case RST_SSEG :
                        reset_date();
                        break;
                case RST_LCD :
                        printToLCD("Welcome to", "Age Calculator..");
                        break;

                case CALC :
                        age_calculator( curr_date, curr_month, curr_year, bdate, bmonth, byear);

                        //printf("INFO  : Age     : %s\n\n", curr_age );
                        printToLCD("Age :", curr_age);
                        break;
                default : // IDLE OR ERROR
                        break;
                }// switch
}

void setting_mode(int command){

                // Push buttons when in idle scenario return ..
                // .. logic 1 - (1111 in binary)/(15 in decimal).
                switch(command){
                case LEFT :
                        selected_piece--;
                        break;
                case RIGHT :
                        selected_piece++;
                        break;
```

```
case UP : // increasing the date
        if(clock_ticks==0 && change==0){
                change=1;
        }// if
        if(clock_ticks==999 && change==1){
                switch(selected_piece){
                case 1:
                        temp_date[1]++;
                        if(temp_date[1]>=10 ){
                                temp_date[1]=0;
                                temp_date[0]++;
                        }
                        if(temp_date[0]>=3 && temp_date[1]>1){
                                temp_date[1]=1;
                                temp_date[0]=0;
                        }
                        break;
                case 2:
                        temp_month[1]++;
                        if(temp_month[1]>=10){
                                temp_month[1]=0;
                                temp_month[0]++;
                        }
                        if(temp_month[0]>=1 && temp_month[1]>2){
                                temp_month[1]=1;
                                temp_month[0]=0;
                        }
                        break;
                case 3:
                        temp_year[3]++;
                        if(temp_year[3]>=10){
                                temp_year[3]=0;
                                temp_year[2]++;
                        }
                        if(temp_year[2]>=10){
                                temp_year[2]=0;
                                temp_year[1]++;
                        }
                        if(temp_year[1]>=10){
                                temp_year[1]=0;
                                temp_year[0]++;
                        }
                        break;
                }// switch
                change=0;
        }// if
        display_sseg(1, selected_piece);
        break;
case DOWN : // decreasing the date
        if(clock_ticks==0 && change==0){
                change=1;
        }// if
        if(clock_ticks==999 && change==1){
                switch(selected_piece){
                case 1:
                        temp_date[1]--;
                        if(temp_date[1]<0){
                                temp_date[1]=9;
                                temp_date[0]--;
                        }
                        if(temp_date[0]<=0 && temp_date[1]==0){
                                temp_date[0]=3;
                                temp_date[1]=1;
                        }
                        break;
                case 2:
```

```
                                        temp_month[1]--;
                                        if(temp_month[1]<0){
                                                temp_month[1]=9;
                                                temp_month[0]--;
                                        }
                                        if(temp_month[0]<=0 && temp_month[1]==0){
                                                temp_month[0]=1;
                                                temp_month[1]=2;
                                        }
                                        break;
                                case 3:
                                        temp_year[3]--;
                                        if(temp_year[3]<0){
                                                temp_year[3]=9;
                                                temp_year[2]--;
                                        }
                                        if(temp_year[2]<0){
                                                temp_year[2]=9;
                                                temp_year[1]--;
                                        }
                                        if(temp_year[1]<0){
                                                temp_year[1]=9;
                                                temp_year[0]--;
                                        }
                                        if(temp_year[0] == 1 && temp_year[1]<9){
                                                reset_date();
                                        }
                                        break;
                                }// switch
                                change=0;
                        }// if
                        display_sseg(1, selected_piece);
                        break;
                default : // IDLE OR ERROR
                        break;
                }// switch

                if(selected_piece<1){
                        selected_piece = 1;
                }else if(selected_piece>3){
                        selected_piece = 3;
                }// if
}// getPKeys_isr

int main(void){
        initial_message();

        // initialize time interrupt
        init_timer(TIMER_BASE, 10000); //init timer with period = 1 msec

        //declare timer isr
        alt_irq_register(TIMER_IRQ, NULL, (alt_isr_func)timer_isr);

        // open the Character LCD port
        char_lcd_dev = alt_up_character_lcd_open_dev ("/dev/lcd");

        if ( char_lcd_dev == NULL){
                printf ("ERROR : could not open character LCD device\n\n");
        } else {
                printf ("INFO  : Opened character LCD device\n\n");

                printToLCD("Welcome to", "Age Calculator..");

        } //if

        if ( char_lcd_dev != NULL) {
```

```
		int command, mode;

		while(1){ // infinite loop

			mode = getCommand(SWITCHES_BASE,0);
			command = getCommand(PKEYS_BASE,   0);

			if ( mode == 1 || mode == 3) {
				// flashing the date
				if(clock_ticks==0 && change==0){
					sseg_flash = 1-sseg_flash;
					change=1;
				} else if(clock_ticks==999 && change==1){
					setting_mode(command);
					display_sseg(sseg_flash, selected_piece);
					change=0;
				}// if
			} else {
				display_sseg(1, selected_piece);
				save_mode(mode, command);
			}// if
		}// while
	}// if

	// Unreachable statement
	return 0;
}// main
```

Αντιστοίχιση ακροδεκτών Άσκησης 2

```
# Clock
clk_clk,Input,PIN_Y2,2,B2_N0,3.3-V LVTTL,

# Seven Segment Displays
sseg_0_export[6],Output,PIN_H22,6,B6_N0,2.5 V,
sseg_0_export[5],Output,PIN_J22,6,B6_N0,2.5 V,
sseg_0_export[4],Output,PIN_L25,6,B6_N1,2.5 V,
sseg_0_export[3],Output,PIN_L26,6,B6_N1,2.5 V,
sseg_0_export[2],Output,PIN_E17,7,B7_N2,2.5 V,
sseg_0_export[1],Output,PIN_F22,7,B7_N0,2.5 V,
sseg_0_export[0],Output,PIN_G18,7,B7_N2,2.5 V,
sseg_1_export[6],Output,PIN_U24,5,B5_N0,2.5 V,
sseg_1_export[5],Output,PIN_U23,5,B5_N1,2.5 V,
sseg_1_export[4],Output,PIN_W25,5,B5_N1,2.5 V,
sseg_1_export[3],Output,PIN_W22,5,B5_N0,2.5 V,
sseg_1_export[2],Output,PIN_W21,5,B5_N1,2.5 V,
sseg_1_export[1],Output,PIN_Y22,5,B5_N0,2.5 V,
sseg_1_export[0],Output,PIN_M24,6,B6_N2,2.5 V,
sseg_2_export[6],Output,PIN_W28,5,B5_N1,2.5 V,
sseg_2_export[5],Output,PIN_W27,5,B5_N1,2.5 V,
sseg_2_export[4],Output,PIN_Y26,5,B5_N1,2.5 V,
sseg_2_export[3],Output,PIN_W26,5,B5_N1,2.5 V,
sseg_2_export[2],Output,PIN_Y25,5,B5_N1,2.5 V,
sseg_2_export[1],Output,PIN_AA26,5,B5_N1,2.5 V,
sseg_2_export[0],Output,PIN_AA25,5,B5_N1,2.5 V,
sseg_3_export[6],Output,PIN_Y19,4,B4_N0,3.3-V LVTTL,
sseg_3_export[5],Output,PIN_AF23,4,B4_N0,3.3-V LVTTL,
sseg_3_export[4],Output,PIN_AD24,4,B4_N0,3.3-V LVTTL,
sseg_3_export[3],Output,PIN_AA21,4,B4_N0,3.3-V LVTTL,
sseg_3_export[2],Output,PIN_AB20,4,B4_N0,3.3-V LVTTL,
sseg_3_export[1],Output,PIN_U21,5,B5_N0,2.5 V,
sseg_3_export[0],Output,PIN_V21,5,B5_N1,2.5 V,
sseg_4_export[6],Output,PIN_AE18,4,B4_N2,3.3-V LVTTL,
sseg_4_export[5],Output,PIN_AF19,4,B4_N1,3.3-V LVTTL,
sseg_4_export[4],Output,PIN_AE19,4,B4_N1,3.3-V LVTTL,
sseg_4_export[3],Output,PIN_AH21,4,B4_N2,3.3-V LVTTL,
sseg_4_export[2],Output,PIN_AG21,4,B4_N2,3.3-V LVTTL,
sseg_4_export[1],Output,PIN_AA19,4,B4_N0,3.3-V LVTTL,
sseg_4_export[0],Output,PIN_AB19,4,B4_N0,3.3-V LVTTL,
sseg_5_export[6],Output,PIN_AH18,4,B4_N2,3.3-V LVTTL,
sseg_5_export[5],Output,PIN_AF18,4,B4_N1,3.3-V LVTTL,
sseg_5_export[4],Output,PIN_AG19,4,B4_N2,3.3-V LVTTL,
sseg_5_export[3],Output,PIN_AH19,4,B4_N2,3.3-V LVTTL,
sseg_5_export[2],Output,PIN_AB18,4,B4_N0,3.3-V LVTTL,
sseg_5_export[1],Output,PIN_AC18,4,B4_N1,3.3-V LVTTL,
sseg_5_export[0],Output,PIN_AD18,4,B4_N1,3.3-V LVTTL,
sseg_6_export[6],Output,PIN_AC17,4,B4_N2,3.3-V LVTTL,
sseg_6_export[5],Output,PIN_AA15,4,B4_N2,3.3-V LVTTL,
sseg_6_export[4],Output,PIN_AB15,4,B4_N2,3.3-V LVTTL,
sseg_6_export[3],Output,PIN_AB17,4,B4_N1,3.3-V LVTTL,
sseg_6_export[2],Output,PIN_AA16,4,B4_N2,3.3-V LVTTL,
sseg_6_export[1],Output,PIN_AB16,4,B4_N2,3.3-V LVTTL,
sseg_6_export[0],Output,PIN_AA17,4,B4_N1,3.3-V LVTTL,
sseg_7_export[6],Output,PIN_AA14,3,B3_N0,3.3-V LVTTL,
```

```
sseg_7_export[5],Output,PIN_AG18,4,B4_N2,3.3-V LVTTL,
sseg_7_export[4],Output,PIN_AF17,4,B4_N2,3.3-V LVTTL,
sseg_7_export[3],Output,PIN_AH17,4,B4_N2,3.3-V LVTTL,
sseg_7_export[2],Output,PIN_AG17,4,B4_N2,3.3-V LVTTL,
sseg_7_export[1],Output,PIN_AE17,4,B4_N2,3.3-V LVTTL,
sseg_7_export[0],Output,PIN_AD17,4,B4_N2,3.3-V LVTTL,

# Push Buttons
pkeys_export[3],Input,PIN_R24,5,B5_N0,2.5 V,
pkeys_export[2],Input,PIN_N21,6,B6_N2,2.5 V,
pkeys_export[1],Input,PIN_M21,6,B6_N1,2.5 V,
pkeys_export[0],Input,PIN_M23,6,B6_N2,2.5 V,

# LCD
lcd_BLON,Output,PIN_L6,1,B1_N2,3.3-V LVTTL,
lcd_DATA[7],Bidir,PIN_M5,1,B1_N2,3.3-V LVTTL,
lcd_DATA[6],Bidir,PIN_M3,1,B1_N1,3.3-V LVTTL,
lcd_DATA[5],Bidir,PIN_K2,1,B1_N1,3.3-V LVTTL,
lcd_DATA[4],Bidir,PIN_K1,1,B1_N1,3.3-V LVTTL,
lcd_DATA[3],Bidir,PIN_K7,1,B1_N1,3.3-V LVTTL,
lcd_DATA[2],Bidir,PIN_L2,1,B1_N2,3.3-V LVTTL,
lcd_DATA[1],Bidir,PIN_L1,1,B1_N2,3.3-V LVTTL,
lcd_DATA[0],Bidir,PIN_L3,1,B1_N1,3.3-V LVTTL,
lcd_EN,Output,PIN_L4,1,B1_N1,3.3-V LVTTL,
lcd_ON,Output,PIN_L5,1,B1_N1,3.3-V LVTTL,
lcd_RS,Output,PIN_M2,1,B1_N2,3.3-V LVTTL,
lcd_RW,Output,PIN_M1,1,B1_N2,3.3-V LVTTL,


# Reset switch
reset_reset_n,Input,PIN_Y23,5,B5_N2,2.5 V,

# Switches
switches_export[1],Input,PIN_AC28,5,B5_N2,2.5 V,
switches_export[0],Input,PIN_AB28,5,B5_N1,2.5 V,
```

Παράρτημα Β

Κώδικας C Άσκησης 3

```
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "math.h"

#include "priv/alt_legacy_irq.h"

// timer definitions
#define TIMER_STAT_REG 0 // status register address offset
#define TIMER_CTRL_REG 1 // control register address offset
#define TIMER_PRDL_REG 2 // period reg (lower 16 bits) addr offset
#define TIMER_PRDH_REG 3 // period reg (upper 16 bits) addr offset
#define ENABLE_TIMER 7 // configure timer to start, continuous mode, enable interrupt (0111 = 7)
#define NO_DATA        0 // No initial drive values

alt_u16 clock_ticks = 0, iteration = 999;

// push buttons definitions
#define PIO_DATA          0  // Data value currently on PIO inputs
#define PIO_INTERRUPT_MASK 2  // Interrupt register addr offset
#define PIO_EDGE_CAPTURE   3  // Edge capture detecte and hold addr offset
#define ENABLE_PIO         15 // Per bit IRQ Enable (1111=15)

// led definitions
#define MOST_LEFT_LED 0x02000000
#define MOST_RIGHT_LED 0x01

//system pause (ERROR / PUSH BUTTON IRQ)
#define PAUSE 1
#define PLAY  0

int system_status = PAUSE;

//initialize pattern for  first - second - third - fourth led
alt_u32 led_pattern[] = {0x02000, 0x01000, 0x0000, 0x0000};
alt_u32 final_pattern;

int running = 1;
//initialize directions
int enable_led_left_direction[4] = {1, 0, 0, 0};
//initialize enabled leds
int enabled_led[4] = {1, 1, 0, 0};

void initial_message(){
   printf("*********************************\n");
   printf("* Hello from Still Statues Game *\n");
   printf("*********************************\n");
```

```
}// initial_message

int getCommand(alt_u32 base, int addr){
        return IORD(base, addr) & 0x0f;
}// getCommand

int countEnabledLeds(){
   int i, count = 0;

   for(i=0; i<4; i++){
      if(enabled_led[i] & 1){
         count++;
      }// if
   }// for

   return count;
}// countEnabledLeds

int findFirstDisabledLed(){
   int i;

   for(i=0; i<4; i++){
      if((enabled_led[i] & 1) == 0){
         return i;
      }// if
   }// for

   return -1;
}// findFirstDisabledLed
int findEnabledLed(int cnt){
   int i, count = 0;

   for(i=0; i<4; i++){
      if(enabled_led[i] & 1){
         count++;
         if(cnt==count){
            //printf("%d is %d enabled.\n",i, cnt);
            return i;
         }
      }// if
   }// for

   return -1;
}// findEnabledLed

void bubbleSort(int array[], int n){
        int i, j, temp;

        for (i = 0 ; i < ( n - 1 ); i++){
            for (j= 0 ; j < n - i - 1; j++){
                if(array[j] < array[j+1]){
                    temp=array[j];
            array[j]   = array[j+1];
            array[j+1] = temp;
            }// if
          }// for
        }// for
```

```
}// bubbleSort

void enableThirdLed(){

	int leds_position[2], led3_position;
	int firstDisabled = findFirstDisabledLed();
	//printf("First Disabled Led : %d \n", firstDisabled);

	// First enabled led
	leds_position[0] = log2(led_pattern[findEnabledLed(1)]);

	// Second enabled led
	leds_position[1] = log2(led_pattern[findEnabledLed(2)]);


	// Descending sort of the led positions
	bubbleSort(leds_position, 2);

	//If the first two leds are in trail re-calculate
	if(leds_position[0] - leds_position[1] < 5){
		//MOST_LEFT_LED distance to first led
		int dist1 = 26 - leds_position[0];
		//MOST_RIGHT_LED distance to second led
		int dist2 = leds_position[1] - 1;

		if(dist1 > dist2){
			led3_position = ceil((26 + leds_position[0])/2);
		}else{
			led3_position = ceil((leds_position[1] + 1)/2);
		}// if
	}else{
		// Calculate third Led Position
		led3_position = ceil((leds_position[0] + leds_position[1])/2);
	}// if

	enabled_led[firstDisabled] = 1;
	enable_led_left_direction[firstDisabled] = rand() % 2;
	led_pattern[firstDisabled] = pow(2, led3_position);

	//printf("3rd Statue added.\n");
}// enableThirdLed

void enableFourthLed(){

	int leds_position[3], led4_position;

	int firstDisabled = findFirstDisabledLed();
	//printf("First Disabled Led : %d \n", firstDisabled);

	// First enabled led
	leds_position[0] = log2(led_pattern[findEnabledLed(1)]);

	// Second enabled led
	leds_position[1] = log2(led_pattern[findEnabledLed(2)]);

	// Third enabled led
	leds_position[2] = log2(led_pattern[findEnabledLed(3)]);
```

```
		// Descending sort of the led positions
		bubbleSort(leds_position, 3);

		// Calculate fourth Led Position
		int dist1_2 = leds_position[0] - leds_position[1];
		int dist2_3 = leds_position[1] - leds_position[2];

		if(dist1_2 > dist2_3){
			led4_position = ceil((leds_position[0] + leds_position[1])/2);
		}else{
			led4_position = ceil((leds_position[1] + leds_position[2])/2);
		}

		enabled_led[firstDisabled] = 1;
		enable_led_left_direction[firstDisabled] = rand() % 2;
		led_pattern[firstDisabled] = pow(2, led4_position);

		//printf("4th Statue added.\n");
}// enableFourthLed

int oneTimePass = 0;

void PKeys_isr(void* context, alt_u32 id){

		int command = getCommand(PKEYS_BASE, PIO_DATA);
		//printf("Given command : %d \n", command );

		//int edge = getCommand(PKEYS_BASE, PIO_EDGE_CAPTURE);
		//printf("EDGE CAPTURE : %d \n", edge );

		int enabledLeds = countEnabledLeds();

		// Push buttons when in idle scenario return ..
		// .. logic 1 - (1111 in binary)/(15 in decimal).
		switch(command){
		case 7 : // Start game | Change directions - (0111 in binary)/(7 in decimal)
			if(system_status == PAUSE || oneTimePass){
				//printf("Game started.\n");
				enable_led_left_direction[0] = 0;
				enable_led_left_direction[1] = 1;
				system_status = PLAY;
			}else{
				if(oneTimePass){
					int i;
					for(i=0; i<4; i++){
						//Randomize direction
						if(enabled_led[i]==1){
							enable_led_left_direction[i] = rand() % 2;
						}// if
					}//for
				}// if
			}// if
			break;

		case 11 : // Adds statue - (1011 in binary)/(11 in decimal)
			//printf("Enabled Leds : %d \n", enabledLeds);
```

```
                if(system_status == PLAY){
                        if(enabledLeds == 2 && oneTimePass){
                                enableThirdLed();
                                oneTimePass = 0;
                        }// if

                        if(enabledLeds == 3 && oneTimePass){
                                enableFourthLed();
                                oneTimePass = 0;
                        }// if
                }// if

                //Reset interrupt when debugging with screen output (printf)
                //IOWR(PKEYS_BASE, PIO_EDGE_CAPTURE, 0x0f);
                break;

        case 13 : // Removes statue - (1101 in binary)/(13 in decimal)
                //printf("Enabled Leds : %d\n", enabledLeds);
                if(enabledLeds>2 && oneTimePass){
                        int disableRandomLed = rand()%enabledLeds;
                        enabled_led[disableRandomLed] = 0;
                        led_pattern[disableRandomLed] = 0x00;
                        oneTimePass = 0;
                        //printf("disableRandomLed : %d \n", disableRandomLed);
                }// if

                //Reset interrupt when debugging with screen output (printf)
                //IOWR(PKEYS_BASE, PIO_EDGE_CAPTURE, 0x0f);
                break;

        case 14 : // Resets the game - (1110 in binary)/(14 in decimal)
                //printf("Game Reset.\n");
                system_status = PAUSE;

                led_pattern[0] = 0x02000;
                enabled_led[0] = 1;
                enable_led_left_direction[0] = 0;

                led_pattern[1] = 0x01000;
                enabled_led[1] = 1;
                enable_led_left_direction[1] = 1;

                led_pattern[2] = 0x00;
                enabled_led[2] = 0;

                led_pattern[3] = 0x00;
                enabled_led[3] = 0;

                final_pattern = led_pattern[0] | led_pattern[1] | led_pattern[2] | led_pattern[3];
                IOWR(LEDS_BASE, 0, final_pattern);

                break;

        //default : // IDLE OR ERROR
        //        Reset interrupt in case of an error or when idle command.
        //          IOWR(PKEYS_BASE, PIO_EDGE_CAPTURE, 0x0f);
        //          break;
```

```
		}// switch
		//Reset interrupt when debugging with screen output (printf)
		IOWR(PKEYS_BASE, PIO_EDGE_CAPTURE, 0x0f);
}// getPKeys_isr

void init_timer(alt_u32 timer_base, alt_u32 period){
		alt_u16 high, low;

		//unpack 32-bit timeout period into two 16-bit half words
		high = (alt_u16) (period >> 16);
		low = (alt_u16) (period & 0x0000ffff);

		//write timeout period
		IOWR(timer_base, TIMER_PRDH_REG, high);
		IOWR(timer_base, TIMER_PRDL_REG, low);

		IOWR(timer_base, TIMER_CTRL_REG, ENABLE_TIMER);
}// init_timer

void init_pkeys(alt_u32 base){
		IOWR(base, PIO_INTERRUPT_MASK, ENABLE_PIO);
}// init_pkeys

void timer_isr(void* context, alt_u32 id){
		//clear timer's status register to (timeout) bit (bit 0) to start the timer over again
		IOWR(TIMER_BASE, TIMER_STAT_REG, NO_DATA);

		if(clock_ticks == iteration) clock_ticks = 0;
		else clock_ticks++;
}// timer_isr

int getSwitches(){
		return (int)(IORD(SWITCHES_BASE, 0) & 0x07);
}// getSwitches

void speedRegulator(){
		int slow_itr = 999, medium_itr = 99, fast_itr = 9;

		switch(getSwitches()){
		case 1 :
				iteration = slow_itr;
				break;
		case 2:
				iteration = medium_itr;
				break;
		case 4:
				iteration = fast_itr;
				break;
		default:
				clock_ticks = 0;
				break;
		}// switch
}// readSpeed

void checkInBoard(alt_u32 *led_pattern, int *led_direction){
		if( *led_pattern >= MOST_LEFT_LED){
				*led_pattern = MOST_LEFT_LED;
```

```
                        *led_direction = 0; // Left
                }// if
                if( *led_pattern <= MOST_RIGHT_LED){
                        *led_pattern = MOST_RIGHT_LED;
                        *led_direction = 1; // Right
                }// if
}// checkInBoard

alt_u32 left_led_flash(alt_u32 led_pattern){
                led_pattern <<= 1; //left shifting
                return led_pattern;
}// left_led_flash

alt_u32 right_led_flash(alt_u32 led_pattern){
                led_pattern >>= 1; //right shifting
                return led_pattern;
}// right_led_flash

void ledDirection(alt_u32 *led_pattern, int *led_direction){
                if(*led_direction == 1){
                        *led_pattern = left_led_flash(*led_pattern);
                }else{
                        *led_pattern = right_led_flash(*led_pattern);
                }// if
}// ledDirection

void rePositioning(int leftLed, int rightLed){
                enable_led_left_direction[leftLed]  = 1 - enable_led_left_direction[leftLed];
                enable_led_left_direction[rightLed] = 1 - enable_led_left_direction[rightLed];

                led_pattern[leftLed]  <<= 1;
                led_pattern[rightLed] >>= 1;

                ledDirection(&led_pattern[leftLed], &enable_led_left_direction[leftLed]);
                ledDirection(&led_pattern[rightLed], &enable_led_left_direction[rightLed]);
}// rePositioning

void statuesGame(){

                if(clock_ticks == iteration && running == 1 && system_status == PLAY){
                        IOWR(LEDS_BASE, 0, final_pattern); // write LEDs
                        running = 0;
                }// if


                if(clock_ticks == 0 && running == 0 && system_status == PLAY){
                        int i, j;

                        // normal movements
                        for(i=0; i<4; i++){
                                if(enabled_led[i] == 1){
                                        ledDirection(&led_pattern[i], &enable_led_left_direction[i]);
                                }// if
                        }// for

                        // Redirection of colliding leds
                        for(i=0; i<4 ;i++){
```

```
                        for(j=i+1; j<4; j++){
                                if(enabled_led[i] && enabled_led[j]){
                                        int i_position = log2(led_pattern[i]);
                                        int j_position = log2(led_pattern[j]);
                                        if(i_position == j_position ){
                                                //printf("Led %d : Position %d - Led %d : Position %d \n", i, i_position, j,
j_position);
                                                if(enable_led_left_direction[i] < enable_led_left_direction[j]){
                                                        rePositioning(i, j);
                                                }else{
                                                        rePositioning(j, i);
                                                }
                                        }// if
                                        // colliding from right to left
                                        if((1-enable_led_left_direction[i]) && (i_position+1)== j_position){
                                                rePositioning(i, j);
                                        }// if
                                        // colliding from left to right
                                        if((1-enable_led_left_direction[j]) && (j_position+1)== i_position){
                                                rePositioning(j, i);
                                        }// if
                                }// if
                        }// for
                }// for

                //led safety of misplassing after redirections
                for(i=0; i<4; i++){
                        if(enabled_led[i] == 1){
                                checkInBoard(&led_pattern[i], &enable_led_left_direction[i]);
                        }// if
                }// for

                final_pattern = led_pattern[0] | led_pattern[1] | led_pattern[2] | led_pattern[3];
                running = 1;
        }// if

}// statuesGame

int main(){

        // check operation status - print welcome message
        initial_message();

        // initialize time interrupt
        init_timer(TIMER_BASE, 50000); //init timer with period = 1 msec

        //initialize push buttons interrupts
        init_pkeys(PKEYS_BASE);

        // register time interrupt
        alt_irq_register(TIMER_IRQ, NULL, (alt_isr_func)timer_isr);

        // register push buttons interrupts
        alt_irq_register(PKEYS_IRQ, NULL, (alt_isr_func)PKeys_isr);

        // Initializes led display
        final_pattern = led_pattern[0] | led_pattern[1] | led_pattern[2] | led_pattern[3];
```

```
	IOWR(LEDS_BASE, 0, final_pattern);

	int command;

	while(1){// running forever

		command = getCommand(PKEYS_BASE, PIO_DATA);

		if(command != 11 && command != 13){
			speedRegulator();
			statuesGame();
		}


		// Clearance flag
		oneTimePass = 1;
	}// while

	// unreachable statement
	return 0;
}// main
```

Αντιστοίχιση ακροδεκτών Άσκησης 3

```
# Clock
clk_clk,Input,PIN_Y2,2,B2_N0,3.3-V LVTTL,

# Reset
reset_reset_n,Input,PIN_Y23,5,B5_N2,2.5 V,

# Push Buttons
pkeys_export[3],Input,PIN_R24,5,B5_N0,2.5 V,
pkeys_export[2],Input,PIN_N21,6,B6_N2,2.5 V,
pkeys_export[1],Input,PIN_M21,6,B6_N1,2.5 V,
pkeys_export[0],Input,PIN_M23,6,B6_N2,2.5 V,

# Display leds
leds_export[25],Output,PIN_H15,7,B7_N2,2.5 V,
leds_export[24],Output,PIN_G16,7,B7_N2,2.5 V,
leds_export[23],Output,PIN_G15,7,B7_N2,2.5 V,
leds_export[22],Output,PIN_F15,7,B7_N2,2.5 V,
leds_export[21],Output,PIN_H17,7,B7_N2,2.5 V,
leds_export[20],Output,PIN_J16,7,B7_N2,2.5 V,
leds_export[19],Output,PIN_H16,7,B7_N2,2.5 V,
leds_export[18],Output,PIN_J15,7,B7_N2,2.5 V,
leds_export[17],Output,PIN_G17,7,B7_N1,2.5 V,
leds_export[16],Output,PIN_J17,7,B7_N2,2.5 V,
leds_export[15],Output,PIN_H19,7,B7_N2,2.5 V,
leds_export[14],Output,PIN_J19,7,B7_N2,2.5 V,
leds_export[13],Output,PIN_E18,7,B7_N1,2.5 V,
leds_export[12],Output,PIN_F18,7,B7_N1,2.5 V,
leds_export[11],Output,PIN_F21,7,B7_N0,2.5 V,
leds_export[10],Output,PIN_E19,7,B7_N0,2.5 V,
leds_export[9],Output,PIN_F19,7,B7_N0,2.5 V,
leds_export[8],Output,PIN_G19,7,B7_N2,2.5 V,
leds_export[7],Output,PIN_G21,7,B7_N1,2.5 V,
leds_export[6],Output,PIN_G22,7,B7_N2,2.5 V,
leds_export[5],Output,PIN_G20,7,B7_N1,2.5 V,
leds_export[4],Output,PIN_H21,7,B7_N2,2.5 V,
leds_export[3],Output,PIN_E24,7,B7_N1,2.5 V,
leds_export[2],Output,PIN_E25,7,B7_N1,2.5 V,
leds_export[1],Output,PIN_E22,7,B7_N0,2.5 V,
leds_export[0],Output,PIN_E21,7,B7_N0,2.5 V,

# Switches
switches_export[2],Input,PIN_AC27,5,B5_N2,2.5 V,
switches_export[1],Input,PIN_AC28,5,B5_N2,2.5 V,
switches_export[0],Input,PIN_AB28,5,B5_N1,2.5 V,
```

Παράρτημα Γ

Κώδικας C Άσκησης 4

philosophers.c

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "priv/alt_legacy_irq.h"

#define TIMER_STAT_REG 0 //status register address offset
#define TIMER_CTRL_REG 1 //control register address offset
#define TIMER_PRDL_REG 2 //period reg (lower 16 bits) addr offset
#define TIMER_PRDH_REG 3 //period reg (upper 16 bits) addr offset

//Predefined place as shared memory to read/write data
#define OnChip SHARED_MEMORY_BASE

alt_u16 clock_ticks = 0;

void init_timer(alt_u32 timer_base, alt_u32 period){
	alt_u16 high, low;

	//unpack 32-bit timeout period into two 16-bit half words
	high = (alt_u16) (period >> 16);
	low = (alt_u16) (period & 0x0000ffff);

	//write timeout period
	IOWR(timer_base, TIMER_PRDH_REG, high);
	IOWR(timer_base, TIMER_PRDL_REG, low);

	//configure timer to start, continuous mode, enable interrupt (0111 = 7)
	IOWR(timer_base, TIMER_CTRL_REG, 0x0007);
}// init_timer

void timer_isr(void* context, alt_u32 id){
	//clear timer's status register to (timeout) bit (bit 0) to start the timer over again
	IOWR(TIMER_BASE, TIMER_STAT_REG, 0);

	if(clock_ticks == 999) clock_ticks = 0;
	else clock_ticks++;

	IOWR(OnChip,0x02,clock_ticks); // Giving timing
}// timer_isr

int main(){
	printf("Philosophers system gives control signals...\n");

	IOWR(LEDS_BASE, 0x00, 0x00); // Initialize LEDs
	IOWR(OnChip, 0x00, 1); // System is running
```

```
IOWR(OnChip, 0x03, 0); // Controlling Philosopher1
IOWR(OnChip, 0x04, 0); // Controlling Philosopher2
IOWR(OnChip, 0x05, 0); // Controlling Philosopher3
IOWR(OnChip, 0x06, 0); // Controlling Philosopher4

int philosopherOneHasTalked;
int philosopherTwoHasTalked;
int philosopherThreeHasTalked;
int philosopherFourHasTalked;

init_timer(TIMER_BASE, 50000); //init timer with period = 1 msec
alt_irq_register(TIMER_IRQ, NULL, (alt_isr_func)timer_isr);

int philosopherOneIsAwake;
int philosopherTwoIsAwake;
int philosopherThreeIsAwake;
int philosopherFourIsAwake;

int giveChance = 1; // Timing combination of checking the next philosopher

while(1){

	if(clock_ticks==0 && giveChance==0){
		giveChance=1;
	}// if

	philosopherOneIsAwake = IORD(SW_BASE, 0) & 0x01;
	philosopherOneHasTalked = IORD(OnChip,0x03);
	if(clock_ticks==999 && giveChance==1 && philosopherOneHasTalked == 0 && philosopherOneIsAwake){
		IOWR(OnChip, 0x01, 1); // Giving for control signal
		giveChance=0;
	} else if(philosopherOneIsAwake == 0){
		philosopherOneHasTalked = 1;
	}

	philosopherTwoIsAwake = IORD(SW_BASE, 0) & 0x02;
	philosopherTwoHasTalked = IORD(OnChip,0x04);
	if(clock_ticks==999 && giveChance==1 && philosopherTwoHasTalked == 0 && philosopherTwoIsAwake){
		IOWR(OnChip, 0x01, 2); // Giving for control signal
		giveChance=0;
	} else if(philosopherTwoIsAwake == 0){
		philosopherTwoHasTalked = 1;
	}

	philosopherThreeIsAwake = IORD(SW_BASE, 0) & 0x04;
	philosopherThreeHasTalked = IORD(OnChip,0x05);
	if(clock_ticks==999 && giveChance==1 && philosopherThreeHasTalked == 0 && philosopherThreeIsAwake){
		IOWR(OnChip, 0x01, 3); // Giving for control signal
		giveChance=0;
	} else if(philosopherThreeIsAwake == 0){
		philosopherThreeHasTalked = 1;
	}

	philosopherFourIsAwake = IORD(SW_BASE, 0) & 0x08;
	philosopherFourHasTalked = IORD(OnChip,0x06);
	if(clock_ticks==999 && giveChance==1 && philosopherFourHasTalked == 0 && philosopherFourIsAwake){
```

```
                        IOWR(OnChip, 0x01, 4); // Giving for control signal
                        giveChance=0;
                } else if(philosopherFourIsAwake == 0){
                        philosopherFourHasTalked = 1;
                }// if

                if((philosopherOneHasTalked & philosopherTwoHasTalked & philosopherThreeHasTalked &
philosopherFourHasTalked) == 1){
                        IOWR(OnChip, 0x03, 0); // Controlling Philosopher1
                        IOWR(OnChip, 0x04, 0); // Controlling Philosopher2
                        IOWR(OnChip, 0x05, 0); // Controlling Philosopher3
                        IOWR(OnChip, 0x06, 0); // Controlling Philosopher4
                }
        }// while

        return 0;
}
```

philosopher1.c

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "math.h"
#include "altera_avalon_mutex.h"
#include "priv/alt_legacy_irq.h"

//Predefined place as shared memory to read/write data
#define OnChip SHARED_MEMORY_BASE

int main(){
	int led_pattern=1, change=1, clock_ticks, countedWords=0;

	// Get the mutex device handle
	alt_mutex_dev *mutex = altera_avalon_mutex_open("/dev/mutex");

	while(1){
		// if the Philosopher wants to talk and the system Philosophers system started
		if((IORD(SW_BASE, 0) & 0x01) != 0 && IORD(OnChip,0x00) == 1){

			// Waiting for control signal
			while(IORD(OnChip,0x01) != ALT_CPU_CPU_ID_VALUE);

			// Printing Data
			printf("Philosopher %d is listening, thinking and waiting to talk... \n", ALT_CPU_CPU_ID_VALUE);

			// Locking MUTEX
			while(altera_avalon_mutex_trylock(mutex, ALT_CPU_CPU_ID_VALUE));

			printf("Philosopher %d is talking. \n", ALT_CPU_CPU_ID_VALUE );

			// Blink the led while the switch is ON - Philosopher still wants to talk after the taking the lead
			while(countedWords<10 && (IORD(SW_BASE, 0) & 0x01) != 0){
				clock_ticks = IORD(OnChip,0x02);
				if(clock_ticks==0 && change==0){
					led_pattern = ~led_pattern;
					change=1;
				}// if
				if(clock_ticks==999 && change==1){
					IOWR(LEDS_BASE, 0, led_pattern&0x01);
					change=0;
					countedWords++;
				}// if
			}// for

			// Reset Philosopher's status
			printf("Philosopher %d stopped talking. \n", ALT_CPU_CPU_ID_VALUE );
			IOWR(OnChip, 0x03, 1); // Philosopher1 finished

			IOWR(LEDS_BASE, 0x00, 0);
			IOWR(OnChip, 0x01, 0);
			countedWords = 0;
```

```
                        // Unlock MUTEX
                        altera_avalon_mutex_unlock(mutex);

                }// if
        }// while

        return 0;
}// main
```

philosopher2.c

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "math.h"
#include "altera_avalon_mutex.h"
#include "priv/alt_legacy_irq.h"

//Predefined place as shared memory to read/write data
#define OnChip SHARED_MEMORY_BASE

int main(){
	int led_pattern=1, change=1, clock_ticks, countedWords=0;

	// Get the mutex device handle
	alt_mutex_dev *mutex = altera_avalon_mutex_open("/dev/mutex");

	while(1){
		// if the Philosopher wants to talk and the system Philosophers system started
		if((IORD(SW_BASE, 0) & 0x02) != 0 && IORD(OnChip,0x00) == 1){
			// Waiting for control signal
			while(IORD(OnChip,0x01) != ALT_CPU_CPU_ID_VALUE);

			// Printing Data
			printf("Philosopher %d is listening, thinking and waiting to talk... \n", ALT_CPU_CPU_ID_VALUE);

			// Locking MUTEX
			while(altera_avalon_mutex_trylock(mutex, ALT_CPU_CPU_ID_VALUE));

			printf("Philosopher %d is talking. \n", ALT_CPU_CPU_ID_VALUE );

			// Blink the led while the switch is ON - Philosopher still wants to talk after the taking the lead
			while(countedWords<10 && (IORD(SW_BASE, 0) & 0x02) != 0){
				clock_ticks = IORD(OnChip,0x02);
				if(clock_ticks==0 && change==0){
					led_pattern = ~led_pattern;
					change=1;
				}// if
				if(clock_ticks==999 && change==1){
					IOWR(LEDS_BASE, 0, led_pattern&0x02);
					change=0;
					countedWords++;
				}// if
			}// for

			// Reset Philosopher's status
			printf("Philosopher %d stopped talking. \n", ALT_CPU_CPU_ID_VALUE );
			IOWR(OnChip, 0x04, 1); // Philosopher2 finished

			IOWR(LEDS_BASE, 0x00, 0);
			IOWR(OnChip, 0x01, 0);
			countedWords = 0;
```

```
                        // Unlock MUTEX
                        altera_avalon_mutex_unlock(mutex);

                }// if
        }// while

        return 0;
}// main
```

philosopher3.c

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "math.h"
#include "altera_avalon_mutex.h"
#include "priv/alt_legacy_irq.h"

//Predefined place as shared memory to read/write data
#define OnChip SHARED_MEMORY_BASE

int main(){
	int led_pattern=1, change=1, clock_ticks, countedWords=0;

	// Get the mutex device handle
	alt_mutex_dev *mutex = altera_avalon_mutex_open("/dev/mutex");

	while(1){
		// if the Philosopher wants to talk and the system Philosophers system started
		if((IORD(SW_BASE, 0) & 0x04) != 0 && IORD(OnChip,0x00) == 1){
			// Waiting for control signal
			while(IORD(OnChip,0x01) != ALT_CPU_CPU_ID_VALUE);

			// Printing Data
			printf("Philosopher %d is listening, thinking and waiting to talk... \n", ALT_CPU_CPU_ID_VALUE);

			// Locking MUTEX
			while(altera_avalon_mutex_trylock(mutex, ALT_CPU_CPU_ID_VALUE));

			printf("Philosopher %d is talking. \n", ALT_CPU_CPU_ID_VALUE );

			// Blink the led while the switch is ON - Philosopher still wants to talk after the taking the lead
			while(countedWords<10 && (IORD(SW_BASE, 0) & 0x04) != 0){
				clock_ticks = IORD(OnChip,0x02);
				if(clock_ticks==0 && change==0){
					led_pattern = ~led_pattern;
					change=1;
				}// if
				if(clock_ticks==999 && change==1){
					IOWR(LEDS_BASE, 0, led_pattern&0x04);
					change=0;
					countedWords++;
				}// if
			}// for

			// Reset Philosopher's status
			printf("Philosopher %d stopped talking. \n", ALT_CPU_CPU_ID_VALUE );
			IOWR(OnChip, 0x05, 1); // Philosopher3 finished

			IOWR(LEDS_BASE, 0x00, 0);
			IOWR(OnChip, 0x01, 0);
			countedWords = 0;
```

```
                        // Unlock MUTEX
                        altera_avalon_mutex_unlock(mutex);

                }// if
        }// while

        return 0;
}// main
```

philosopher4.c

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "io.h"
#include "alt_types.h"
#include "system.h"
#include "stdlib.h"
#include "math.h"
#include "altera_avalon_mutex.h"
#include "priv/alt_legacy_irq.h"

//Predefined place as shared memory to read/write data
#define OnChip SHARED_MEMORY_BASE

int main(){
	int led_pattern=1, change=1, clock_ticks, countedWords=0;

	// Get the mutex device handle
	alt_mutex_dev *mutex = altera_avalon_mutex_open("/dev/mutex");

	while(1){
		// if the Philosopher wants to talk and the system Philosophers system started
		if((IORD(SW_BASE, 0) & 0x08) != 0 && IORD(OnChip,0x00) == 1){
			// Waiting for control signal
			while(IORD(OnChip,0x01) != ALT_CPU_CPU_ID_VALUE);

			// Printing Data
			printf("Philosopher %d is listening, thinking and waiting to talk... \n", ALT_CPU_CPU_ID_VALUE);

			// Locking MUTEX
			while(altera_avalon_mutex_trylock(mutex, ALT_CPU_CPU_ID_VALUE));

			printf("Philosopher %d is talking. \n", ALT_CPU_CPU_ID_VALUE );

			// Blink the led while the switch is ON - Philosopher still wants to talk after the taking the lead
			while(countedWords<10 && (IORD(SW_BASE, 0) & 0x08) != 0){
				clock_ticks = IORD(OnChip,0x02);
				if(clock_ticks==0 && change==0){
					led_pattern = ~led_pattern;
					change=1;
				}// if
				if(clock_ticks==999 && change==1){
					IOWR(LEDS_BASE, 0, led_pattern&0x08);
					change=0;
					countedWords++;
				}// if
			}// for

			// Reset Philosopher's status
			printf("Philosopher %d stopped talking. \n", ALT_CPU_CPU_ID_VALUE );
			IOWR(OnChip, 0x06, 1); // Philosopher4 finished

			IOWR(LEDS_BASE, 0x00, 0);
			IOWR(OnChip, 0x01, 0);
			countedWords = 0;
```

```
                        // Unlock MUTEX
                        altera_avalon_mutex_unlock(mutex);

                }// if
        }// while

        return 0;
}// main
```

Αντιστοίχιση ακροδεκτών Άσκησης 4

```
# Clock
clk_clk,Input,PIN_Y2,2,B2_N0,3.3-V LVTTL,

# Display leds
leds_export[3],Output,PIN_F21,7,B7_N0,2.5 V,
leds_export[2],Output,PIN_E19,7,B7_N0,2.5 V,
leds_export[1],Output,PIN_F19,7,B7_N0,2.5 V,
leds_export[0],Output,PIN_G19,7,B7_N2,2.5 V,

# Switches
sw_export[3],Input,PIN_AD27,5,B5_N2,2.5 V,
sw_export[2],Input,PIN_AC27,5,B5_N2,2.5 V,
sw_export[1],Input,PIN_AC28,5,B5_N2,2.5 V,
sw_export[0],Input,PIN_AB28,5,B5_N1,2.5 V,

# Reset switch
reset_reset_n,Input,PIN_Y23,5,B5_N2,2.5 V,
```

Αναφορές - Βιβλιογραφία

[1] “What is Pedagogy?”, Child Australia. 2017
[2] "Embedded Systems Design, Second Edition, S. Heath"
[3] “Teaching Skills and Concepts for Embedded Systems Design”, P. Bertels, M. D’ Haene, T. Degryse, D. Stroobandt
[4] Getting Hands on Altera Quartus Prime Software
[5] https://en.wikipedia.org/wiki/Embedded_system
[6] https://en.wikipedia.org/wiki/Programmable_logic_device
[7] https://www.amiq.com/consulting/misc/free_pdf_books/fpgas_for_dummies_ebook.pdf
[8] Ψηφιακή Σχεδίαση, 4η έκδοση, M. Mano, M. Ciletti
[9] Σχεδιασμός Ψηφιακών Συστημάτων σε FPGAs, W. Wolf
[10] Intel, DE2-115 User manual
[11] Intel, My First FPGA for Altera DE2-115 Board
[12] Intel, My First Nios II for Altera DE2-115 Board
[13] Intel, Introduction to the Quartus Prime Software
[14] Intel, Quartus Prime Introduction Using Verilog Designs
[15] Intel, Quartus Prime Handbook
[16] Intel, Nios II Software Developer Handbook
[17] Intel, Nios II Processor Reference Handbook
[18] Intel, Quartus Prime Introduction Using Schematic Design
[19] Intel, Introduction to the Altera Nios II Soft Processor
[20] Intel, Introduction to the Altera SOPC Builder Using Verilog Design
[21] Intel, Nios II Embedded Design Suite
[22] https://software.intel.com/en-us/
[23] Intel, 16x2 Character Display for Altera DE2-Series Boards
[24] Intel, Nios PIO Data Sheet
[25] Intel, Creating Multiprocessor Nios II Systems Tutorial