ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ (ΣΕΡΡΕΣ)
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

Ανάπτυξη μοντέλου επανακαθοριζόμενου υλικού (Reconfigurable Hardware) σε FPGA που θα περιλαμβάνει soft processor και interface διασύνδεσης με Η/Υ για χρήση σε μεθόδους Εξελικτικού Υλικού (Evolvable Hardware)

Πτυχιακή Εργασία της

Καργιολάκη Βασιλικής (1432)

Επιβλέπων: Δρ. Σπυρίδων Α. Καζαρλής

ΣΕΡΡΕΣ, ΝΟΕΜΒΡΙΟΣ 2015

<u>Υπεύθυνη Δήλωση</u> *: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής Τ.Ε του Τ.Ε.Ι. Κεντρικής Μακεδονίας (Σερρών).*

ΠΕΡΙΛΗΨΗ

Σκοπός της πτυχιακής εργασίας είναι η μελέτη μοντέλου επανακαθοριζόμενου υλικού σε προγραμματιζόμενη ψηφίδα (διάταξη FPGA) που περιλαμβάνει μαλακό επεξεργαστή, όπως ο επεξεργαστής Nios II της εταιρείας Altera, επικοινωνία με Η/Υ αλλά και προγραμματισμό του μέσω της διασύνδεσης JTAG_UART. Γι' αυτό μελετήθηκε ο επεξεργαστής Nios II και αναπτύχθηκαν οι καταχωρητές, οι εντολές και άλλα χαρακτηριστικά του και στην συνέχεια δόθηκαν παραδείγματα προγραμματισμού του. Για την υλοποίηση του συστήματος Nios II σε FPGA πρέπει να χρησιμοποιηθούν τα εργαλεία σχεδιασμού υλικού Quartus II και το Qsys. Επίσης μελετήθηκαν τα Altera Monitor Program και Nios II SBT for Eclipse τα οποία βοηθούν, όπως και τα προηγούμενα, στην μετατροπή της διάταξης FPGA σε System on Chip. Επιπλέον, γίνεται αναφορά στην αναπτυξιακή πλακέτα DE2, η οποία χρησιμοποιείται στην εργασία, και αναπτύσσονται τα χαρακτηριστικά της. Τέλος, παρουσιάζονται τρεις εφαρμογές που βοηθούν στην κατανόηση της σειριακής επικοινωνίας του Η/Υ με την αναπτυξιακή πλακέτα και τον επεξεργαστή Nios II.

ABSTRACT

The purpose of this project is the study of reconfigurable hardware model on a programmable chip (FPGA device) that includes soft processor, such as the Nios II processor of Altera, communication with Host PC and programming via the serial port. For this reason the Nios II processor was studied and the registers, instructions and other features were referred and then programming examples were given. For the implementation of Nios II system to a FPGA must be used the hardware design tools Quartus II, SoPC Builder and Qsys. Also the Altera Monitor Program and Nios II SBT for Eclipse were studied, that help, as the previous, to the conversion of a FPGA device into a System on Chip. In addition, reference is made to the DE2 development board, which is used to this project, and its features are mentioned. Finally, three applications are presented that help in understanding the communication of Host PC with the development board and the Nios II processor.

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

Ευρετήριο Σχεδιαγραμμάτων

ΕΙΣΑΓΩΓΗ

Σκοπός της πτυχιακής εργασίας που ακολουθεί είναι η μελέτη μοντέλου επανακαθοριζόμενου υλικού σε προγραμματιζόμενη ψηφίδα (διάταξη FPGA) που περιλαμβάνει μαλακό επεξεργαστή (soft processor), όπως ο επεξεργαστής Nios II της εταιρείας Altera, και επικοινωνία με τον Η/Υ αλλά και προγραμματισμό του μέσω της διασύνδεσης JTAG UART (όπως αναφέρθηκε στην περίληψη). Αρχικά θα γίνει μια εισαγωγή για την κατανόηση των ενσωματωμένων συστημάτων. Θα γίνει αρχικά μια αναφορά στον ορισμό και την ιστορία των ενσωματωμένων συστημάτων, όπως επίσης και στα χαρακτηριστικά τους, και στην συνέχεια θα οριστούν τα ενσωματωμένα συστήματα σε προγραμματιζόμενες ψηφίδες (SoPCs). Στη συνέχεια θα περιγραφούν οι διατάξεις προγραμματιζόμενης λογικής (PLDs) και θα γίνει αναφορά στα παλαιότερα και στα πιο πρόσφατα PLDs, δίνοντας περισσότερο βάση στους Πίνακες Πυλών Προγραμματιζόμενους στο Πεδίο (FPGA) και αναλύοντας τα πλεονεκτήματά τους.

Στη συνέχεια θα περιγραφεί αναλυτικά ο επεξεργαστής Nios II της εταιρείας Altera, αναφέροντας την αρχιτεκτονική του, τα χαρακτηριστικά του, τους καταχωρητές και τις εντολές του και επιπλέον θα δοθούν παραδείγματα προγραμμάτων για την κατανόηση των εντολών του Nios II. Το επόμενο κεφάλαιο θα είναι αφιερωμένο στην αναπτυξιακή πλακέτα DE2, η οποία χρησιμοποιείται για την υλοποίηση της πτυχιακής εργασίας, περιγράφοντας τον σκοπό της αλλά και από τι αποτελείται η αναπτυξιακή πλακέτα. Ακολουθεί μια περιγραφή των εργαλείων λογισμικού που χρησιμοποιούμε για την δημιουργία ενός συστήματος σε προγραμματιζόμενο τσιπ (SoPC) και για τον προγραμματισμό του Nios II. Τα σχεδιαστικά αυτά εργαλεία είναι το Quartus II (για την σύνθεση), ο SoPC Builder ή το Qsys tool (για τη δημιουργία του συστήματος) και το Altera Monitor Program ή το Nios II Software Build Tools for Eclipse (για τον προγραμματισμό και την αποσφαλμάτωση).

Στο εργαστηριακό μέρος θα μελετηθούν τρεις εφαρμογές. Και οι τρεις εφαρμογές θα αναλύουν τον τρόπο αποστολής και λήψης χαρακτήρων μέσω της θύρας JTAG με διάφορες παραλλαγές.

Κεφάλαιο 1 : Ενσωματωμένα Συστήματα

1.1 Τι είναι ενσωματωμένα συστήματα

Ενσωματωμένο σύστημα (*embedded system*) είναι ένα συγκεκριμένου σκοπού (single purpose) υπολογιστικό υποσύστημα ενός συνολικού συστήματος, που έχει ως σκοπό την επίβλεψη και τον έλεγχο του συστήματος αυτού.

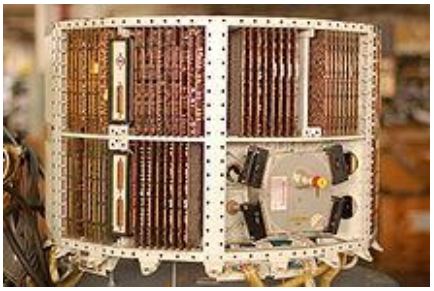
Μια σημαντική πλευρά του λογισμικού που δημιουργήθηκε για τα ενσωματωμένα συστήματα είναι ότι πρέπει να αλληλεπιδρά στενά με το υλικό. Είναι συνυφασμένα με φυσικές διαδικασίες. Άρα πρέπει να ανταποκριθούν σε προβλήματα πραγματικού κόσμου σε *πραγματικό χρόνο*. Ο όρος αντιδραστικό σύστημα χρησιμοποιείται συχνά για να περιγράψει το γεγονός ότι τα χρονικά σημεία στα οποία εκτελούνται οι διάφορες ρουτίνες καθορίζονται από τα γεγονότα εξωτερικά του επεξεργαστή, όπως το κλείσιμο ενός διακόπτη ή την άφιξη νέων δεδομένων σε μία θύρα εισόδου. Ο σχεδιαστής του λογισμικού πρέπει να αποφασίσει πώς θα επιτευχθεί αυτή η αλληλεπίδραση.

Ο έλεγχος με μικροεπεξεργαστές χρησιμοποιείται πλέον συχνά σε κάμερες, κινητά, smartphones, σε συσκευές κουζίνας, αυτοκίνητα και σε πολλά παιχνίδια. Χαμηλό κόστος, και υψηλή αξιοπιστία είναι οι βασικές απαιτήσεις αυτών των εφαρμογών. Όλα μπορεί να επιτευχθούν με την τοποθέτηση σε ένα μόνο chip, όχι μόνο το κύκλωμα του επεξεργαστή, αλλά και κάποιες διεπαφές μνήμης, εισόδου/εξόδου, κυκλώματα χρονισμού, και άλλα χαρακτηριστικά ώστε να είναι εύκολο να εφαρμοστεί ένα ολοκληρωμένο σύστημα ελέγχου με υπολογιστή, χρησιμοποιώντας πολύ λίγα τσιπ. Τα τσιπ μικροεπεξεργαστή αυτού του τύπου αναφέρονται γενικά ως μικροελεγκτές. Παρακάτω θα αναπτυχθούν οι απαιτήσεις για τον σχεδιασμό ενσωματωμένων συστημάτων, όπως επίσης και τα κύρια χαρακτηριστικά ενός μικροελεγκτή σε ενσωματωμένα συστήματα.

1.2 Ιστορία ενσωματωμένων συστημάτων

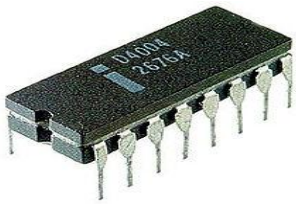
Εικόνα 1.1: Apollo Guidance Computer και DSKY (οθόνη και πληκτρολόγιο) [1]

Ένα από τα πρώτα αναγνωρίσιμα σύγχρονα ενσωματωμένα συστήματα ήταν το Apollo Guidance Computer [Εικόνα 1.2], που αναπτύχθηκε από τον Charles Stark Draper στο MIT Instrumentation Laboratory, το 1966. Κατά την έναρξη του έργου Apollo, ο υπολογιστής Apollo Guidance θεωρήθηκε το πιο ριψοκίνδυνο στοιχείο στο έργο, καθώς χρησιμοποιούσε τα τότε νεοαποκτηθέντα μονολιθικά ολοκληρωμένα κυκλώματα για την μείωση του μεγέθους και του βάρους. Ένα από τα πρώτα μαζικής παραγωγής ενσωματωμένο σύστημα ήταν το Autonetics D-17 [Εικόνα 1.1], υπολογιστής καθοδήγησης του πυραύλου Minuteman, που κυκλοφόρησε το 1961. Όταν δημιουργήθηκε ο πύραυλος Minuteman II το 1966, ο υπολογιστής καθοδήγησης D-17 αντικαταστάθηκε με ένα νέο σύστημα, και χρησιμοποιήθηκε για πρώτη φορά υψηλός όγκος ολοκληρωμένων κυκλωμάτων. Αυτό μόνο του μείωσε τις τιμές στις πύλες quad NAND των ICs (Integrated Circuits – Ολοκληρωμένα κυκλώματα) από 1000$ / τεμάχιο σε 3$ / τεμάχιο, επιτρέποντας πλέον τη χρήση τους ως εμπορικά προϊόντα.



Εικόνα 1.2: Autonetics D-17 [2]

Από αυτές τις πρώτες εφαρμογές στην δεκαετία του 1960, οι τιμές των ενσωματωμένων συστημάτων έπεσαν και υπήρξε δραματική αύξηση στην επεξεργαστική ισχύ και λειτουργικότητα. Ένας από τους πρώτους μικροεπεξεργαστές για παράδειγμα, ο Intel 4004 [Εικόνα 1.3], σχεδιάστηκε για υπολογιστή τσέπης, αριθμομηχανή, και άλλα μικρά συστήματα, αλλά ακόμη απαιτούνται εξωτερική μνήμη και υποστηρικτικά τσιπ.



Εικόνα 1.3: Intel 4004, ο 1ος εμπορικός μικροεπεξεργαστής [3]

Το 1978, η Εθνική Ένωση Μηχανικών Κατασκευαστών (National Engineering Manufacturers Association) κυκλοφόρησε ένα “πρότυπο” για προγραμματιζόμενους μικροελεγκτές, συμπεριλαμβανομένου σχεδόν οποιοδήποτε υπολογιστή – βασισμένο σε ελεγκτές, όπως αριθμητικούς και βάσει γεγονότων ελεγκτές.

Καθώς έπεσε το κόστος των μικροεπεξεργαστών και μικροελεγκτών, έγινε εφικτό να αντικατασταθεί το ακριβό χειριστήριο που βασίζεται σε αναλογικά εξαρτήματα, όπως ποτενσιόμετρα και μεταβλητούς πυκνωτές με κουμπιά πάνω/κάτω ή χειριστήρια που ελέγχονται από μικροεπεξεργαστή. Μέχρι τις αρχές της δεκαετίας του 1980, υποσυστήματα μνήμης, εισόδου και εξόδου του συστήματος είχαν ενσωματωθεί στο ίδιο τσιπ, όπως τον επεξεργαστή σχηματίζοντας ένα μικροελεγκτή. Μικροελεγκτές βρίσκουν εφαρμογές όπου ένας υπολογιστής γενικής σκοπού θα ήταν υπερβολικά δαπανηρό. Ένας χαμηλού κόστους μικροελεγκτής μπορεί να προγραμματιστεί ώστε να έχει τον ίδιο ρόλο όπως ένας μεγάλος αριθμός από ξεχωριστά εξαρτήματα.

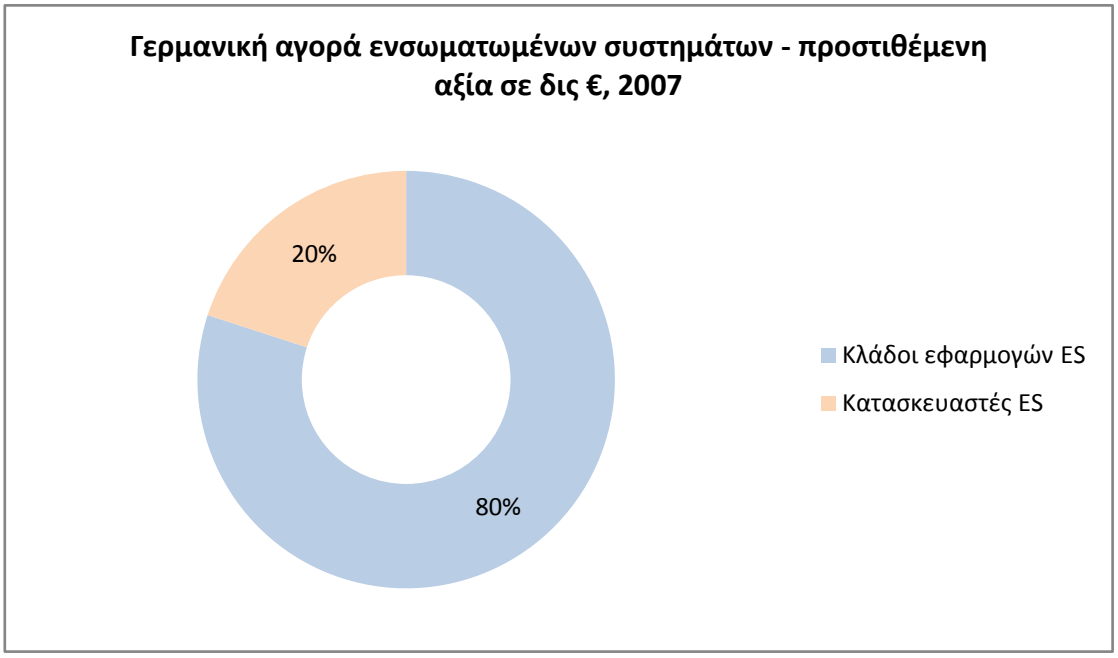
1.3 Αγορά και χρήση ενσωματωμένων συστημάτων [4]

Τα ενσωματωμένα συστήματα μπορούν να περιέχονται εκτός των υπολογιστών και σε κινητά τηλέφωνα, εκτυπωτές, αυτοκίνητα (μηχανή, φρένα), αεροπλάνα (μηχανή έλεγχοι πτήσης, πλοήγηση, επικοινωνία) ψηφιακή τηλεόραση, οικιακές συσκευές, σε συστήματα συλλογής δεδομένων, ρομποτικά συστήματα, βιομηχανικοί αυτοματισμοί, μηχανική όραση, έξυπνα κτίρια κ.α.

Το γεγονός ότι το φάσμα των εφαρμογών των ενσωματωμένων συστημάτων είναι τόσο ευρύ καθιστά δύσκολη την εκτίμηση του μεγέθους της αγοράς, με αποτέλεσμα πολύ μεγάλες αποκλίσεις: οι εκτιμήσεις για το μέγεθος της παγκόσμιας αγοράς ενσωματωμένων συστημάτων ξεκινούν από τα 60 δις € και φθάνουν έως τα 138 δις €.

Η Γερμανία θεωρείται η τρίτη μεγαλύτερη αγορά ενσωματωμένων συστημάτων παγκοσμίως, μετά τις ΗΠΑ και την Ιαπωνία. Το μέγεθος της γερμανικής αγοράς εκτιμάται στα 18,7 δις € για το 2007 (πηγή: BITKOM) και οι ρυθμοί ανάπτυξής της στο 9-10% ετησίως. Ανά περιοχή, η βασική αγορά των γερμανικών εταιρειών κατασκευής ενσωματωμένων συστημάτων είναι η εσωτερική με 69% και ακολουθούν οι αγορές των χωρών της ΕΕ με 17%, των ΗΠΑ με 7 %, της Ασίας με 4%, Αμερική (εκτός ΗΠΑ) 1% και λοιπές χώρες με 2%.

Η προστιθέμενη αξία στους κλάδους όπου βρίσκουν εφαρμογές τα ενσωματωμένα συστήματα υπολογίζεται στα 15 δις €, ενώ 3,7 δις € είναι ο κύκλος εργασιών των εταιρειών που κατασκευάζουν/αναπτύσσουν ενσωματωμένα συστήματα [Σχεδιάγραμμα 1.1].



Σχεδιάγραμμα 1.1: Γερμανική αγορά ενσωματωμένων συστημάτων 2007 (Πηγή: BITKOM) [4]

Ανά κλάδο εφαρμογών, το μεγαλύτερο τμήμα του κύκλου εργασιών των εταιρειών κατασκευής ενσωματωμένων συστημάτων πραγματοποιείται στις τηλεπικοινωνίες/κατασκευή ηλεκτρολογικού εξοπλισμού (25%) και στην αυτοκινητοβιομηχανία (25%) - Τα ηλεκτρικά και ηλεκτρονικά συστήματα αποτελούν ολοένα και μεγαλύτερο ποσοστό της αξίας ενός αυτοκινήτου (25% το 2007, πρόβλεψη για 35% το 2015)- , ενώ ακολουθούν ο κλάδος κατασκευής

μηχανολογικού εξοπλισμού (19%), της αεροδιαστημικής (8%) (υπολογίζεται ότι 12% της αξίας ενός αεροπλάνου αφορά τα ενσωματωμένα συστήματα) και άλλοι κλάδοι (23%), όπως περιβαλλοντική και ιατρική τεχνολογία.

Ας δούμε μερικά παραδείγματα ενσωματωμένων εφαρμογών, κατηγοριοποιημένα. Ο κατάλογος είναι μόνο ενδεικτικός, για να εκτιμηθεί το εύρος της αγοράς αλλά και οι δυνατότητες για περαιτέρω ανάπτυξη τέτοιων συστημάτων. Σε όλες τις παρακάτω κατηγορίες συστημάτων υπάρχει επεξεργαστής, στον οποίο ο χρήστης δεν έχει άμεση πρόσβαση. Για παράδειγμα, μπορεί σε ψηφιακά παιχνίδια τύπου Gameboy να μπορεί ο χρήστης να επιλέξει παιχνίδι με εισαγωγή κατάλληλης κασέτας, αλλά δεν έχει την δυνατότητα να προγραμματίσει την κονσόλα. Επίσης, σε διάφορα συστήματα όπως δρομολογητές δικτύων η κατασκευάστρια εταιρία μπορεί να αλλάζει το πρόγραμμά τους, αλλά και πάλι ο χρήστης δεν έχει άμεση πρόσβαση στο λογισμικό. Ενδεικτικές κατηγορίες είναι:

- **Υπολογιστές και Περιφερειακά**
  - Ασύρματα περιφερειακά (π.χ. bluetooth ακουστικά/μικρόφωνα, IR ποντίκια και πληκτρολόγια, κλπ.)
  - Ασύρματα δίκτυα (routers και κάρτες για WiFi, 802.11, bluetooth, κλπ.)
  - Κονσόλες παιχνιδιών (π.χ. Sony Playstation, Microsoft Xbox, κλπ.)
- **Είδη Προσωπικής Ευκολίας**
  - Φορητά Παιχνίδια (π.χ. Gameboy, Nintendo, κλπ.)
  - Κινητά τηλέφωνα
  - Προσωπικοί Ψηφιακοί Βοηθοί (PDA)
  - Φορητά συστήματα παγκοσμίου εντοπισμού (GPS – Global Positioning Systems)
- **Αυτοκίνητα**
  - Συστήματα ελέγχου απόδοσης μηχανής (καθορισμό μίγματος, χρονισμό, κλπ.)
  - Συστήματα ελέγχου ρύπων
  - Συστήματα ελέγχου άνεσης καμπίνας επιβατών (π.χ. κλιματισμός, ρυθμίσεις καθισμάτων, ρυθμίσεις καθρεφτών, κλπ.)

- **Οικιακές Συσκευές**
  - Ψυγεία, κουζίνες, φούρνοι μικροκυμάτων
  - Τηλεοράσεις, συσκευές εικόνας (Video Cassette recorder – VCR, Digital Video Disc - DVD)
  - Στερεοφωνικά νέας γενιάς και συστήματα αιθουσών προβολής σπιτιού (Home Theater)
- **Βιομηχανικά Συστήματα**
  - Βιομηχανικά Ρομπότ
  - Αριθμητικά ελεγχόμενα μηχανουργικά μηχανήματα (π.χ. φρέζες με αριθμητικό έλεγχο – Computer Numerical Control – CNC)
- **Υγεία, και Υποστηρικτική Τεχνολογία για Άτομα με Ειδικές Ανάγκες (ΑΜΕΑ)**
  - Φορητοί καρδιογράφοι
  - Συστήματα καθαρισμού αίματος
  - Συστήματα παρακολούθησης ζωτικών λειτουργιών ασθενών
  - Απηνιδοτές
  - Αναπηρικά αμαξίδια
  - Συσκευές εισόδου ελεγχόμενες από επιστόμιο
  - Συστήματα δημιουργίας ήχου (σύνθεση φωνής)
- **Τηλεπικοινωνίες**
  - Ψηφιακά τηλεφωνικά κέντρα
  - Δικτυακός εξοπλισμός (δρομολογητές - routers, μεταγωγείς - switches, κλπ.)
  - Δορυφορικά συστήματα
- **Αγροτική Παραγωγή και Περιβάλλον**
  - Συστήματα παρακολούθησης και ελέγχου συνθηκών εδάφους
  - Συστήματα ελέγχου περιβάλλοντος και αυτόματου ταΐσματος σε κτηνοτροφικές μονάδες
  - Συστήματα παρακολούθησης ρύπων
  - Συστήματα έγκαιρης προειδοποίησης φυσικών καταστροφών
- **Ασφάλεια**
  - Συστήματα παρακολούθησης σπιτιών και συναγερμοί
  - Συστήματα πυρόσβεσης

- **Μεταφορές**
  - Ηλεκτρονικά αεροσκαφών (avionics)
  - Συστήματα εντοπισμού θέσης λεωφορείων, τραίνων, κλπ. και ενημέρωσης επιβατών για επικείμενες αφίξεις
  - Φωτεινές ενδείξεις χρόνου αναμονής σε σηματοδότες οδικής κυκλοφορίας και κυκλοφορίας πεζών
  - Συστήματα για αυτόματη παρακολούθηση θέσης γραμμάτων/δεμάτων σε ταχυμεταφορικές εταιρίες
- **Υπηρεσίες**
  - Συστήματα αυτόματων τραπεζικών συναλλαγών (ATM)
  - Συστήματα διατήρησης προτεραιότητας ουρών (π.χ. σε τράπεζες)
  - Φωτεινές ενδείξεις (π.χ. κυλιόμενες οθόνες)
  - Φορητά συστήματα για παραγγελίες σε εστιατόρια, κλπ.
- **Δημόσια Διοίκηση**
  - Αυτόματα συστήματα στάθμευσης
  - Συστήματα για κλήσεις σε παραβάτες κυκλοφορίας, στάθμευσης, κλπ.

1.4 Χαρακτηριστικά Ενσωματωμένων Συστημάτων

- Ειδική λειτουργία (όχι γενικού σκοπού).
- Αλληλεπίδραση με το περιβάλλον (πραγματικού χρόνου).
- Περιορισμός πόρων (ισχύς, χώρος, κόστος).
- Κρισιμότητα ασφάλειας (απώλεια ζωής, ιδιοκτησίας, κ.λπ.).
- Αύξηση της πίεσης on time-to-market.
- Εξελιγμένη λειτουργικότητα.
- Λειτουργία πραγματικού χρόνου.
- Χαμηλό κατασκευαστικό κόστος.
- Χαμηλή ισχύς.
- Σχεδίαση σε αυστηρές προθεσμίες από μικρές ομάδες.

1.5 Απαιτήσεις σχεδιασμού

Ο σχεδιαστής ενός ενσωματωμένου συστήματος έχει να πάρει πολλές σημαντικές αποφάσεις. Η φύση της εφαρμογής ή του προϊόντος που πρέπει να σχεδιαστεί παρουσιάζει ορισμένες προϋποθέσεις και περιορισμούς. Μερικά από τα πιο σημαντικά ζητήματα που αντιμετωπίζει ο σχεδιαστής είναι τα εξής:

- **Κόστος:** Το κόστος των ηλεκτρονικών σε πολλές ενσωματωμένες εφαρμογές πρέπει να είναι χαμηλό. Η λιγότερο δαπανηρή λύση υλοποιείται αν ένα μόνο τσιπ αρκεί για την εκτέλεση όλων των λειτουργιών που πρέπει να παρέχονται. Αυτό είναι δυνατό μόνο αν αυτό το τσιπ παρέχει ικανότητα εισόδου/εξόδου (Input/Output – I/O), ώστε να ανταποκριθεί στις απαιτήσεις της εφαρμογής και αρκετή μνήμη στο τσιπ (on-chip memory) για την αποθήκευση των προγραμμάτων και των δεδομένων.
- **Δυνατότητα Εισόδου/Εξόδου (I/O):** Τα τσιπ μικροελεγκτών παρέχουν μία ποικιλία από I/O, που κυμαίνονται από απλές παράλληλες και σειριακές θύρες για χρονιστές και κυκλώματα μετατροπής A/D και D/A. Ο αριθμός των διαθέσιμων γραμμών I/O είναι σημαντικός. Χωρίς επαρκείς γραμμές I/O είναι απαραίτητο να χρησιμοποιηθούν εξωτερικά κυκλώματα για να καλύψουν το έλλειμμα.
- **Μέγεθος:** Τα τσιπ μικροελεγκτών διατίθενται σε διάφορα μεγέθη. Εάν η εφαρμογή μπορεί να υλοποιηθεί με ένα μικροελεγκτή 8-bit, τότε δεν έχει νόημα να χρησιμοποιηθεί τσιπ μεγαλύτερου μεγέθους, 16 ή 32-bit, που είναι πιθανό πιο ακριβό, φυσικά μεγαλύτερο, και καταναλώνει περισσότερη ενέργεια. Η πλειονότητα των πρακτικών εφαρμογών μπορούν να υλοποιηθούν με σχετικά μικρά τσιπ.
- **Κατανάλωση ενέργειας:** Η κατανάλωση ισχύος είναι ένας σημαντικός παράγοντας σε όλες τις εφαρμογές του υπολογιστή. Σε συστήματα υψηλής απόδοσης, η κατανάλωση ισχύος τείνει να είναι υψηλή, γι’ αυτό απαιτείται κάποιος μηχανισμός για να μειώσει την θερμότητα που παράγεται. Σε πολλές ενσωματωμένες εφαρμογές η κατανάλωση ενέργειας είναι αρκετά χαμηλή και έτσι η διάχυση της θερμότητας δεν απασχολεί. Ωστόσο, αυτές οι εφαρμογές

περιλαμβάνουν συχνά μπαταρία, έτσι η διάρκεια ζωής της μπαταρίας, η οποία εξαρτάται από την κατανάλωση ενέργειας, είναι ένας βασικός παράγοντας.

- **On-chip memory:** Η ενσωμάτωση της μνήμης σε ένα τσιπ μικροελεγκτή επιτρέπει την εφαρμογή ενιαίου τσιπ σε απλές ενσωματωμένες εφαρμογές. Το μέγεθος και το είδος της μνήμης έχουν σημαντικές προεκτάσεις. Μια σχετικά μικρή ποσότητα της μνήμης RAM μπορεί να είναι επαρκής για την αποθήκευση δεδομένων κατά τη διάρκεια των υπολογισμών. Μια μεγαλύτερη μνήμη μόνο για ανάγνωση είναι απαραίτητη για την αποθήκευση προγραμμάτων. Η μνήμη αυτή μπορεί να είναι ROM, PROM, EPROM, EEPROM, ή τύπου Flash. Για προϊόντα μεγάλου όγκου, οι πιο οικονομικές επιλογές είναι μικροελεγκτές με ROM. Ωστόσο, αυτή είναι επίσης η λιγότερο ευέλικτη επιλογή επειδή τα περιεχόμενα της ROM είναι μόνιμα καθορισμένα την στιγμή που κατασκευάζεται το τσιπ. Η μεγαλύτερη ευελιξία παρέχεται από τις EEPROM και τις Flash μνήμες, που μπορούν να προγραμματιστούν πολλαπλές φορές.

  Για εφαρμογές που έχουν μεγαλύτερες απαιτήσεις αποθήκευσης, είναι απαραίτητο να χρησιμοποιηθεί μια εξωτερική μνήμη. Οι μικροελεγκτές που προορίζονται για πιο εξελιγμένες εφαρμογές όπου απαιτείται ένα σημαντικό ποσό της μνήμης, το οποίο δεν μπορεί να υλοποιηθεί εντός του chip μικροελεγκτή, δεν έχουν καμία μνήμη on-chip..

- **Επίδοση:** Η επίδοση δεν αποτελεί σημαντικό παράγοντα, όταν οι μικροελεγκτές χρησιμοποιούνται σε εφαρμογές όπως οι οικιακές συσκευές και τα παιχνίδια. Σε αυτές τις περιπτώσεις μπορεί να επιλεχθούν μικρά και φθηνά τσιπ. Όμως, σε εφαρμογές όπως οι ψηφιακές φωτογραφικές μηχανές, τα κινητά τηλέφωνα και κάποια βιντεοπαιχνίδια με τηλεχειριστήρια είναι απαραίτητο να έχουν πολύ υψηλότερη επίδοση. Η υψηλή επίδοση απαιτεί πιο ισχυρά τσιπ, με αποτέλεσμα να κοστίζουν ακριβότερα και να καταναλώνουν περισσότερο ρεύμα. Δεδομένου ότι η εφαρμογή συχνά λειτουργεί με μπαταρία, είναι σημαντικό να ελαχιστοποιηθεί η κατανάλωση ενέργειας.

- **Λογισμικό:** Υπάρχουν πολλά πλεονεκτήματα στη χρήση υψηλού επιπέδου γλωσσών προγραμματισμού. Διευκολύνουν την διαδικασία της ανάπτυξης του προγράμματος και διευκολύνουν την συντήρηση και τροποποίηση του λογισμικού στο μέλλον. Ωστόσο, υπάρχουν περιπτώσεις, όταν είναι αναγκαίο, που χρησιμοποιείται η γλώσσα assembly. Ένα καλά διατυπωμένο πρόγραμμα σε γλώσσα assembly είναι πιθανό δημιουργήσει κώδικα που είναι 10% έως 20% πιο συμπαγής (σε σχέση με το ποσό της αποθήκευσης που χρειάζεται) από τον κώδικα που παράγεται μετά την μετάφραση (compiler). Εάν μία ενσωματωμένη εφαρμογή βασίζεται σε ένα μικροελεγκτή που έχει περιορισμένη μνήμη on-chip, είναι σημαντικό πλεονέκτημα εάν ο απαραίτητος κώδικας μπορεί να χωρέσει στην μνήμη που παρέχεται στο τσιπ, αποφεύγοντας την ανάγκη για εξωτερική μνήμη.

  Η περιορισμένη χωρητικότητα των διαθέσιμων μνημών on-chip RAM είναι ένας σημαντικός παράγοντας για έναν σχεδιαστή συστήματος. Αυτή η μνήμη χρησιμοποιείται για την αποθήκευση δυναμικών δεδομένων, ως προσωρινό buffer, και για την υλοποίηση μιας στοίβας. Όταν ο χρήστης γράφει ένα πρόγραμμα σε γλώσσα υψηλού επιπέδου όπως η C, θα πρέπει να ληφθεί μέριμνα ώστε να διασφαλιστεί ότι το συνολικό ποσό του κώδικα και των δεδομένων δεν υπερβαίνει τη διαθέσιμη μνήμη.

- **Σετ εντολών:** Ένα άλλο σημαντικό ζήτημα είναι η φύση του σετ εντολών που χρησιμοποιεί ο επεξεργαστής. Οι εντολές τύπου CISC (Complex instruction set computer) οδηγούν σε κώδικα πιο συμπαγή από τις εντολές τύπου RISC (Reduced instruction set computer). Έτσι, η επιλογή του επεξεργαστή έχει αντίκτυπο στο μέγεθος του κώδικα. Ένα ενδιαφέρον παράδειγμα μιας προσέγγισης που αντιμετωπίζει αυτό το πρόβλημα παρέχεται από την έκδοση Thumb της αρχιτεκτονικής ARM, όπου ένα σύνολο εντολών τύπου RISC σχεδιάστηκε για επεξεργαστές 32-bit έχει τροποποιηθεί σε μια πιο υψηλά κωδικοποιημένη μορφή με τη χρήση 16-bit εντολών. Τα προγράμματα που έχουν γραφτεί για τις εκδόσεις Thumb είναι μέχρι 30 τοις εκατό πιο συμπαγές από αυτά που γράφτηκαν για την πλήρη αρχιτεκτονική ARM.

- **Αναπτυξιακά εργαλεία:** Οι σχεδιαστές των ψηφιακών συστημάτων βασίζονται σε μεγάλο βαθμό σε εργαλεία ανάπτυξης. Τα εργαλεία αυτά περιλαμβάνουν πακέτα λογισμικού για προγράμματα σχεδίασης (computer-aided design, CAD), μεταφραστές (compilers), assemblers (μεταφράζουν την γλώσσα assembly σε ένα αρχείο), και προσομοιωτές (simulators) για επεξεργαστές. Το εύρος και η διαθεσιμότητα των εργαλείων συχνά εξαρτάται από την επιλογή του ενσωματωμένου επεξεργαστή. Επίσης είναι πολύ χρήσιμο να υπάρχει υποστήριξη από τρίτους, όπου υπάρχουν εναλλακτικές πηγές εργαλείων και την τεκμηρίωση. Καλή τεκμηρίωση και χρήσιμες συμβουλές από τον κατασκευαστή (αν χρειάζεται) είναι εξαιρετικά πολύτιμη.

- **Δυνατότητα δοκιμών και Αξιοπιστία:** Οι πλακέτες με τυπωμένα κυκλώματα είναι συχνά δύσκολο να δοκιμαστούν, ιδιαίτερα αν υπάρχουν πολλά τσιπ πάνω σε αυτές. Η διαδικασία δοκιμής απλοποιείται σε μεγάλο βαθμό αν το όλο σύστημα έχει σχεδιαστεί ώστε να είναι εύκολα ελέγξιμο. Ένα τσιπ μικροελεγκτή μπορεί να περιλαμβάνει κυκλώματα που καθιστούν ευκολότερο να δοκιμάσουν τις πλακέτες τυπωμένων κυκλωμάτων που περιέχουν αυτό το τσιπ.

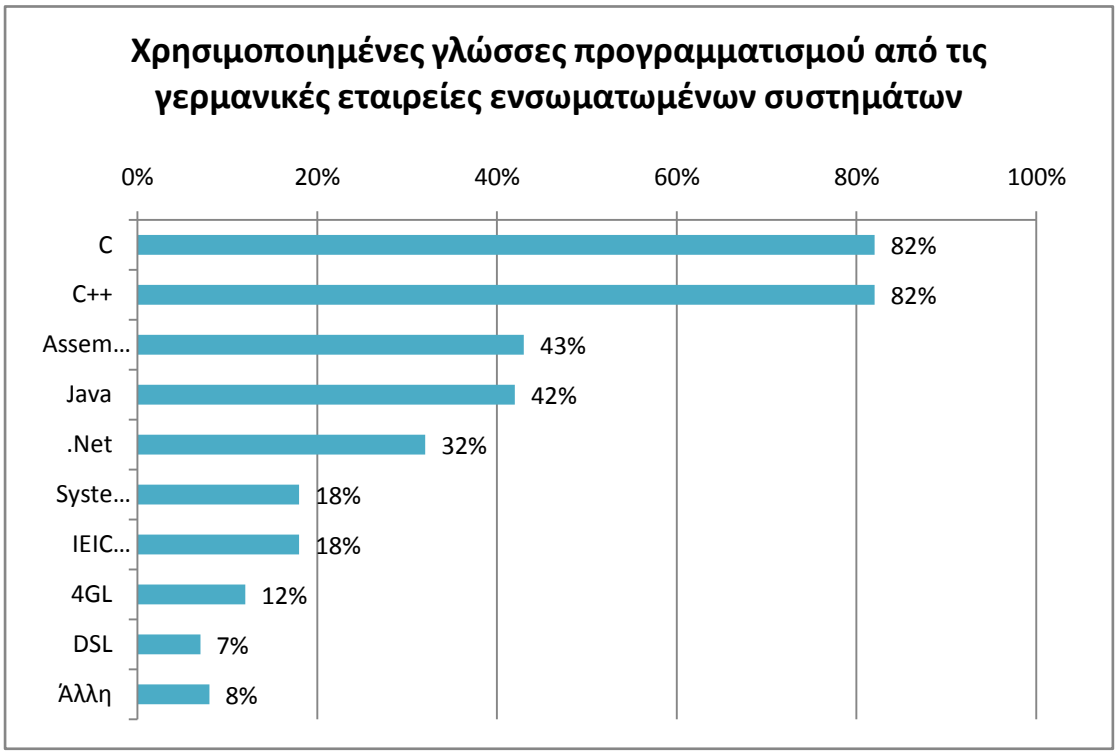
  Οι ενσωματωμένες εφαρμογές απαιτούν αντοχή και αξιοπιστία. Ο κύκλος της ζωής ενός τυπικού προϊόντος αναμένεται να είναι τουλάχιστον πέντε ετών. Αυτό έρχεται σε αντίθεση με τους προσωπικούς υπολογιστές, οι οποίοι τείνουν να θεωρηθούν παρωχημένοι σε μικρότερο χρονικό διάστημα.

1.6 Γλώσσες προγραμματισμού στα ενσωματωμένα συστήματα

Ένα καίριο ζήτημα για την ανάπτυξη του κλάδου ενσωματωμένων συστημάτων είναι η καθιέρωση προτύπων (standards) για τις χρησιμοποιούμενες αρχιτεκτονικές και γλώσσες προγραμματισμού, τα οποία να ισχύουν σε περισσότερους από έναν βιομηχανικούς κλάδους εφαρμογών. Μέχρι στιγμής, υπάρχουν μόνο πρότυπα που εφαρμόζονται σε συγκεκριμένους βιομηχανικούς κλάδους, όπως το EN 9100 στην αεροδιαστημική. Προς την καθιέρωση ενός

ανοιχτού προτύπου για την αυτοκινητοβιομηχανία εργάζεται η πρωτοβουλία Autosar – Automotive Open System Architecture. [4]

Οι γλώσσες προγραμματισμού που χρησιμοποιούνται στον κλάδο των ενσωματωμένων συστημάτων φαίνεται στο παρακάτω σχεδιάγραμμα [Σχεδιάγραμμα 1.2]:



Σχεδιάγραμμα 1.2: Χρησιμοποιούμενες γλώσσες προγραμματισμού στον κλάδο των ενσωματωμένων συστημάτων [4]

Επίσης υπάρχουν και οι γλώσσες περιγραφής VHDL και Verilog που θα περιγράψουμε στην παρακάτω ενότητα.

1.6.1 Γλώσσα Περιγραφής Υλικού VHDL

Η VHDL είναι μία γλώσσα περιγραφής υλικού, που χρησιμοποιείται για την ανάπτυξη ολοκληρωμένων ψηφιακών κυκλωμάτων και συστημάτων. Ο όρος VHDL είναι συντόμευση των λέξεων VHSIC Hardware Description Language, όπου VHSIC - Very High Speed Integrated Circuit. Η γλώσσα αυτή αναπτύχθηκε στις αρχές της

δεκαετίας του 1980 με χρηματοδότηση από το υπουργείο Άμυνας των ΗΠΑ και έγινε πρότυπη το 1987 από το ινστιτούτο IEEE ως *IEEE 1076*. Αργότερα, δημιουργήθηκε μια βελτιωμένη έκδοσή της, η *IEEE 1164* (1993). Ακολούθησαν και νεότερες αναβαθμίσεις του προτύπου, όπως η *IEEE 1076-2002* και *IEEE 1076-2008*. Ο κώδικας VHDL αποτελεί τον βασικό τύπο αρχείου που δέχονται ως είσοδο τα λογισμικά ψηφιακής σχεδίασης κυκλωμάτων (Computer Aided Design ή CAD), για τη δημιουργία σύνθετων ολοκληρωμένων κυκλωμάτων. Η γλώσσα VHDL χρησιμοποιείται ευρύτατα για την περιγραφή και υλοποίηση ψηφιακών συστημάτων σε προγραμματιζόμενες λογικές διατάξεις, τύπου CPLDs (Complex Programmable Logic Devices - Σύνθετες προγραμματιζόμενες λογικές διατάξεις) και FPGAs (Field Programmable Gate Arrays - διατάξεις πυλών προγραμματιζόμενες στο πεδίο). Επίσης, έχει καθιερωθεί σαν ένα πρότυπο (standard) στη σχεδίαση ηλεκτρονικών κυκλωμάτων ASICs (Application Specific Integrated Circuits). Η καθιέρωση της ως πρότυπο μας διαβεβαιώνει ότι και οι επόμενες εκδόσεις εργαλείων σχεδίασης θα υποστηρίζουν το πρότυπο αυτό. Έτσι, ένα κύκλωμα που περιγράφηκε και αναπτύχθηκε με τα σημερινά εργαλεία σχεδίασης θα είναι μεταφέρσιμο μελλοντικά σε νέα εργαλεία σχεδίασης, με ελάχιστες ή καθόλου αλλαγές.

Εκτός από τη γλώσσα VHDL ευρύτατη χρήση και αποδοχή έχει βρει διεθνώς και η γλώσσα περιγραφής υλικού Verilog.

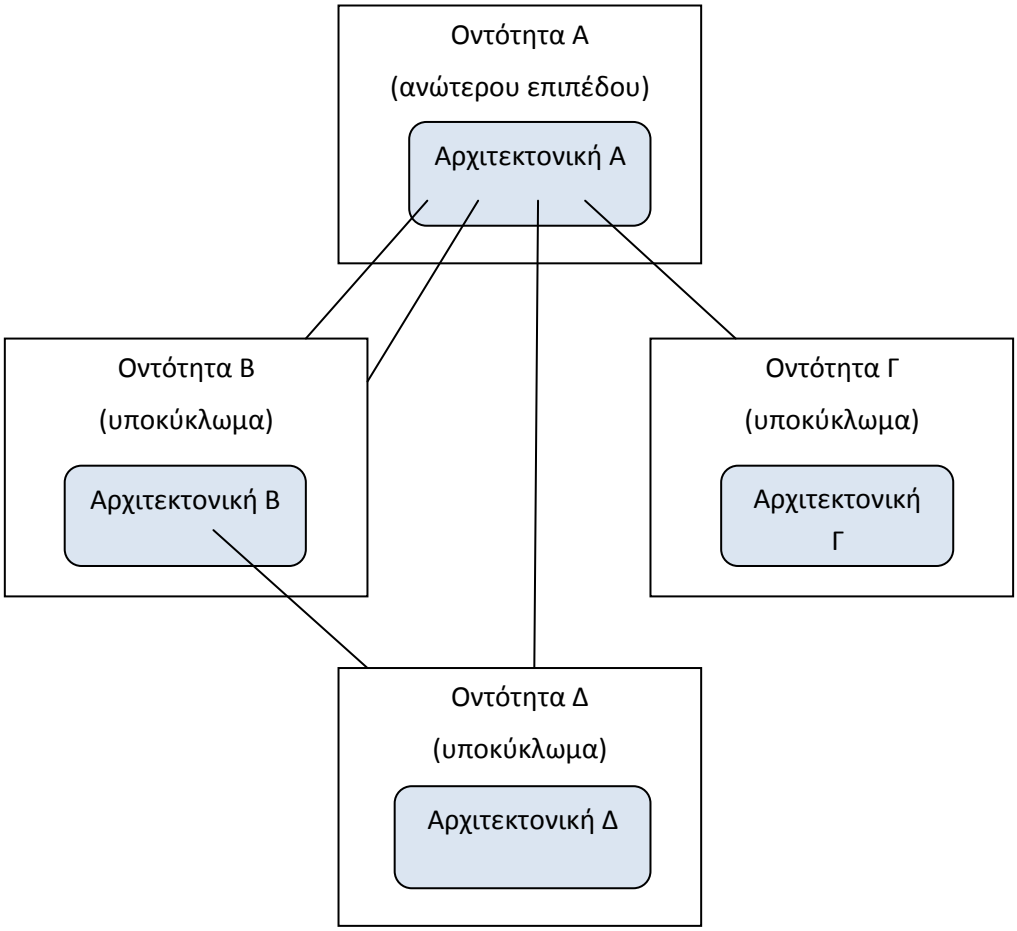
Χαρακτηριστικά της γλώσσας VHDL

Η VHDL υπακούει στις αρχές των παράλληλων γλωσσών και δεν είναι τυχαίο ότι έχει τις ρίζες της στην Ada, που χρησιμοποιείται για να προγραμματίσει παράλληλες διεργασίες. Ταυτόχρονα, υπακούει στις αρχές του δομημένου προγραμματισμού, που δίνει τη δυνατότητα της σχεδίασης ιεραρχικών κυκλωμάτων. Τέλος, περιέχει τις αρχές του συγχρονισμού και του χρονισμού, που είναι εγγενείς στα κυκλώματα. Ένα από τα χαρακτηριστικά της VHDL είναι ότι μοντελοποιεί με ακρίβεια τόσο τις λειτουργίες του κυκλώματος, όσο και τους χρόνους κατά τους οποίους οι λειτουργίες παράγουν αποτελέσματα.

Η VHDL διαφέρει από τις συμβατικές γλώσσες κατά το ότι δεν προορίζεται να περιγράψει λειτουργίες που εκτελούνται σειριακά, η μία μετά την άλλη. Κάθε πρόταση ή τμήμα κώδικα περιγράφει λειτουργίες, οι οποίες παράγουν αποτελέσματα σε συγχρονισμό με άλλες λειτουργίες,. Τα αποτελέσματα της προσομοίωσης παράγονται με βάση αυστηρές χρονικές προδιαγραφές σε διάφορα σημεία του κυκλώματος και εν τέλει στις εξόδους. Με την έννοια αυτή, είναι δυνατό ένα τμήμα κώδικα να μπορεί να γραφεί σε οποιοδήποτε σημείο του προγράμματος, αφού παράγει αποτελέσματα ανεξαρτήτως της σειράς του. Στο τέλος, ο compiler θα συνθέσει κυκλώματα που θα υλοποιούν στην πράξη τις σύγχρονες λειτουργίες που περιγράφει ο κώδικας.

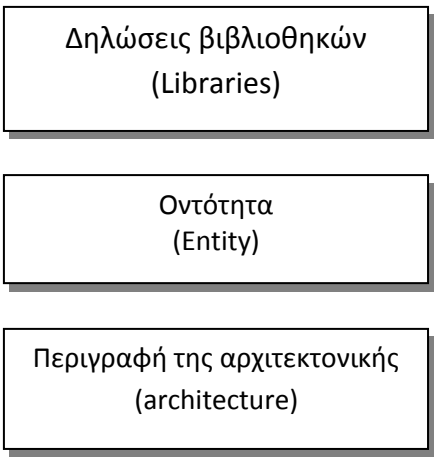
Η VHDL μπορεί να περιγράψει τόσο σύγχρονες όσο και ακολουθιακές λογικές λειτουργίες. Οι διεργασίες που περιγράφονται στη VHDL παράγουν αποτέλεσμα είτε ταυτόχρονα με την εφαρμογή των εισόδων, όπως συμβαίνει στα συνδυαστικά κυκλώματα, είτε σε συγχρονισμό με συμβάντα (π.χ. παλμούς ρολογιού), όπως συμβαίνει στα ακολουθιακά κυκλώματα. Για το λόγο αυτό η σύνταξη του κώδικα μπορεί να γίνει με σύγχρονες δομές (concurrent code) ή και με ακολουθιακές δομές (sequential code).

Ο ιεραρχικός τρόπος σχεδίασης γίνεται δυνατός στη VHDL εξαιτίας της δομής του κώδικα, που διακρίνει ανάμεσα στην περιγραφή του κυκλώματος ως βαθμίδα (block) και στη λειτουργική του περιγραφή. Τα δύο αυτά μέρη του κώδικα αναφέρονται ως «οντότητα» και «αρχιτεκτονική». Έχοντας σχεδιάσει ένα κύκλωμα σε VHDL, είναι δυνατό να το ενσωματώνουμε σε πιο περίπλοκα κυκλώματα, κάνοντας απλώς αναφορά στην «οντότητά» του, χωρίς να χρειάζεται να επαναλαμβάνουμε την περιγραφή της αρχιτεκτονικής. Μπορούμε να παράγουμε «στιγμιότυπα» (instances) ενός κυκλώματος, όσες φορές χρειάζεται, κάνοντας αναφορά σ’ αυτό μέσα στην αρχιτεκτονική μιας ανώτερης ιεραρχικής βαθμίδας [Σχήμα 1.1]. Η δυνατότητα της επανάληψης των στιγμιότυπων συνιστά ένα χαρακτηριστικό της γλώσσας VHDL που καθιστά την περιγραφή του υλικού (hardware) επαναχρησιμοποιήσιμη (reusable). Κάθε κύκλωμα που περιγράφεται μία φορά μπορεί να χρησιμοποιηθεί ξανά σαν βαθμίδα υποκυκλώματος, σε οποιοδήποτε άλλο ψηφιακό σύστημα.



Σχήμα 1.1: Ιεραρχική σύνδεση οντοτήτων στη δομημένη σχεδίαση με τη γλώσσα VHDL [5]

Δομή προγράμματος VHDL



Σχήμα 1.2: Βασικά τμήματα ενός αρχείου VHDL [5]

Το Σχήμα 1.2 δείχνει την βασική δομή ενός αρχείου VHDL. Τα βασικά μέρη ενός κώδικα που περιγράφει κύκλωμα σε VHDL είναι η *«οντότητα»* (entity) και η *«αρχιτεκτονική»* (architecture). Στις περισσότερες περιπτώσεις, πριν το τμήμα της οντότητας θα πρέπει να δηλωθούν και κάποια πακέτα βιβλιοθηκών, που περιγράφουν τους τύπους δεδομένων ή υποκυκλώματα που χρησιμοποιεί ο σχεδιαστής.

Οι δηλώσεις των *βιβλιοθηκών* (libraries) και των πακέτων (packages) επιτρέπουν τη χρήση συγκεκριμένων τμημάτων κώδικα, που έχουν ήδη δημιουργηθεί στο παρελθόν και χρησιμοποιούνται συχνά. Μια βιβλιοθήκη, λοιπόν, είναι συλλογή χρήσιμων τμημάτων κώδικα, οι οποίοι μπορούν να χρησιμοποιηθούν ξανά, σε άλλα σχέδια, κάνοντας απλά μια δήλωση της βιβλιοθήκης στην αρχή του κώδικα. Τέτοια τμήματα κώδικα είναι δηλώσεις τύπων δεδομένων, υποκυκλώματα, συναρτήσεις (functions), διαδικασίες (procedures). Ένα έτοιμο, τυποποιημένο πακέτο που χρησιμοποιείται πολύ συχνά είναι το *std_logic_1164* της βιβλιοθήκης *ieee*, το οποίο περιγράφει τον τύπο δεδομένων std_logic. Οι βιβλιοθήκες δηλώνονται με τη λέξη-κλειδί LIBRARY, ενώ τα πακέτα εισάγονται με τη λέξη-κλειδί USE. Εκτός από τις τυποποιημένες βιβλιοθήκες, ο κάθε χρήστης μπορεί να δημιουργεί τις δικές του.

Η *οντότητα* (entity) περιγράφει το κύκλωμα ως βαθμίδα, με εισόδους και εξόδους. Περιλαμβάνει μόνο τις διασυνδέσεις που έχει το κύκλωμα με άλλες βαθμίδες αλλά αποκρύπτει τη λειτουργία του κυκλώματος. Το αναγνωριστικό όνομα που δίνει ο σχεδιαστής στην οντότητα καθώς και τα σήματα εισόδου και εξόδου είναι καθοριστικά για κάθε υλοποίηση του κυκλώματος αυτού.

Η *αρχιτεκτονική* (architecture) περιλαμβάνει όλες τις λεπτομέρειες της λειτουργίας ενός κυκλώματος. Οι εντολές και οι δηλώσεις που περιλαμβάνονται στο σώμα της αρχιτεκτονικής περιγράφουν με ακρίβεια ποιές λογικές συναρτήσεις θα υλοποιεί το κύκλωμα και τι τιμές θα λαμβάνουν τα σήματα του κυκλώματος σε κάθε χρονική στιγμή.

Στον παραπάνω κώδικα φαίνεται ένα πλήρες πρόγραμμα σε VHDL. Ο κώδικας περιγράφει την απλή λειτουργία ενός δυαδικού αποκωδικοποιητή.

```
--decoder 2:4
ENTITY decoder1 IS
   PORT (
      x: IN bit_vector (1 downto 0);
      y: OUT bit_vector (3 downto 0));
END decoder1;
----------------------------------------
ARCHITECTURE decoder2_1 OF decoder1 IS
BEGIN
      y <=  "0001" WHEN x="00" ELSE
            "0010" WHEN x="01" ELSE
            "0100" WHEN x="10" ELSE
            "1000";
END decoder2_1;
```

1.7 Αρχιτεκτονικές και Μεθοδολογίες Ανάπτυξης για Ενσωματωμένα Συστήματα

Σε ενσωματωμένα συστήματα συχνά χρησιμοποιούμε μικροεπεξεργαστές ή μικροελεγκτές, επεξεργαστές σήματος (DSP – Digital Signal Processors) ή αναδιατασσόμενη λογική (FPGA – Field Programmable Gate Array). Χρησιμοποιούμε επίσης ολοκληρωμένα κυκλώματα συγκεκριμένης εφαρμογής (ASIC – Application Specific Integrated Circuits) με μικρό ή μεγάλο βαθμό ολοκλήρωσης.

1.7.1 Μικροεπεξεργαστές και Μικροελεγκτές

Παρότι μικροεπεξεργαστές και μικροελεγκτές είναι παραλλαγές στο ίδιο θέμα, δηλαδή, στην ολοκλήρωση ενός γενικής χρήσεως επεξεργαστή σε ένα ολοκληρωμένο κύκλωμα (IC - Integrated Circuit), υπάρχουν και διαφορές:

- *Μικροεπεξεργαστής* είναι η υλοποίηση ενός γενικής χρήσεως επεξεργαστή σε ένα ολοκληρωμένο κύκλωμα με κύριο κριτήριο την απόδοση. Η ολοκλήρωση σκοπό έχει την μεγιστοποίηση της ταχύτητας του επεξεργαστή. Εφόσον υπάρχει

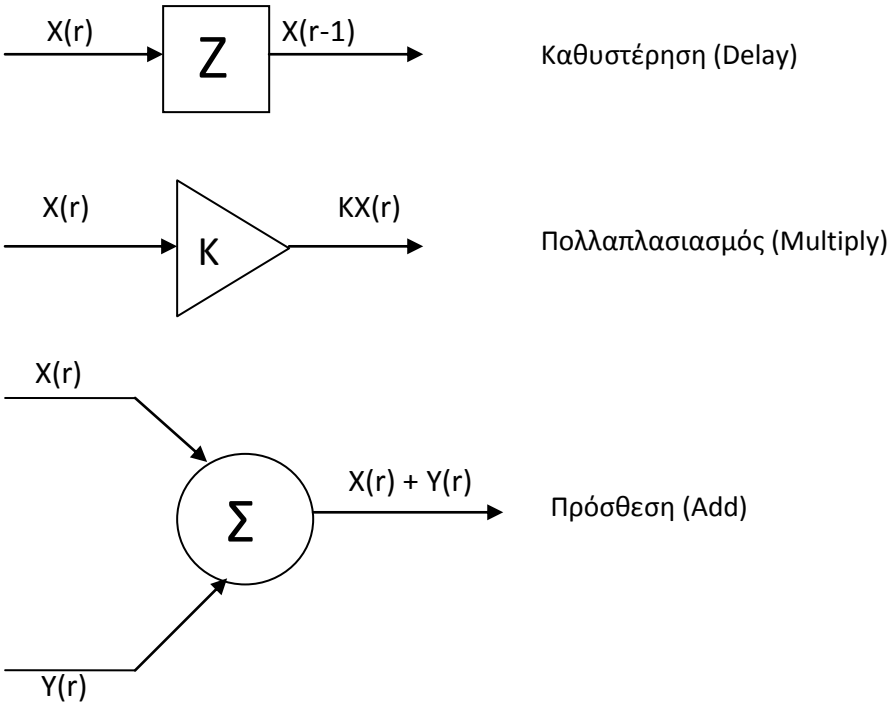
δυνατότητα (λόγω διαθέσιμων ημιαγωγών στοιχείων στο υπόστρωμα), επεκτείνεται με κριτήριο την απόδοση (π.χ. υποστήριξη πράξεων κινητής υποδιαστολής - Floating Point), με υποστήριξη της λειτουργικότητας της μνήμης (με ενσωμάτωση μονάδας διαχείρισης μνήμης MMU – Memory Management Unit), με υποστήριξη μέρους της ιεραρχίας μνήμης (με ενσωμάτωση κρυφής μνήμης ενός ή και δύο επιπέδων), κλπ.

- *Μικροελεγκτής* είναι η υλοποίηση ενός γενικής χρήσεως επεξεργαστή σε ένα ολοκληρωμένο κύκλωμα με κύριο γνώμονα την αυτοτέλεια. Η ολοκλήρωση σκοπό έχει την μεγιστοποίηση της δυνατότητας του μικροελεγκτή να χρησιμοποιηθεί με ελάχιστα ή και καθόλου κυκλώματα υποστήριξης. Επειδή σχεδόν όλοι οι μικροελεγκτές χρησιμοποιούνται σε ενσωματωμένες εφαρμογές, παρέχουν στήριξη σε επίπεδο υλικού αλλά και σε επίπεδο εντολών για πρωταρχικές λειτουργίες όπως η μέτρηση χρόνου (με χρονιστές – timers), η μέτρηση συμβάντων (με μετρητές – counters), η σύγκριση αναλογικού σήματος με κάποιο προγραμματιζόμενο κατώφλι (analog signal comparators), η υποστήριξη επικοινωνίας μέσω τυποποιημένου σειριακού πρωτοκόλλου (όπως RS232, USB, κλπ.), καθώς και υποστήριξη δομών που αυξάνουν την αξιοπιστία (π.χ. με μετρητές επανεκκίνησης - watchdog timers).

1.7.2 Επεξεργαστές Σήματος (DSP – Digital Signal Processors)

Οι επεξεργαστές σήματος είναι μία ειδική κατηγορία μικροεπεξεργαστών που θυσιάζει την βελτιστοποίηση για ευρεία γκάμα εφαρμογών που έχουν οι μικροεπεξεργαστές χάριν της συγκριτικά φθηνότερης υλοποίησης αριθμητικών πράξεων, συνήθως (αλλά όχι πάντα) κινητής υποδιαστολής. Εξ’ αρχής, αποσκοπούσαν στην γρήγορη εκτέλεση τριών βασικών λειτουργιών [Σχήμα 1.3], που είναι συνηθισμένες στην επεξεργασία σήματος:

- Καθυστέρηση z-1 όπου η είσοδος καθυστερείται κατά ένα κύκλο.
- Πολλαπλασιασμό
- Πρόσθεση



Σχήμα 1.3: Βασικές λειτουργίες που βελτιστοποιεί ένας επεξεργαστής σήματος (DSP) [6]

Επειδή τα σήματα είναι συνήθως πραγματικού χρόνου, οι μνήμες είναι φυσικές (δηλ. δεν υπάρχει ιδεατή μνήμη. Οι επεξεργαστές σήματος DSP έχουν από την δομή τους:

- Καθοριστική και εκπεφρασμένη χρήση των πόρων της μνήμης (σε αντίθεση με τους μικροεπεξεργαστές που λόγω κρυφής μνήμης και ιδεατής μνήμης ο χρήστης δεν έχει απ' ευθείας έλεγχο για εντολές και δεδομένα).
- Εντολές για γρήγορη αριθμητική και συνδυασμό πολλαπλασιασμού και πρόσθεσης, γνωστές και σαν MAC (multiply – accumulate).
- Χρήση ενός (ή και περισσότερων) accumulator επί πλέον από τους γενικής χρήσης καταχωρητές (κάτι που σε γενικής χρήσης επεξεργαστές δεν ισχύει πλέον, ήδη από την δεκαετία του 1980).
- Πολλαπλές αλλαγές δεικτών στην μνήμη (pointers).
- Σημαντικά μικρότερο κόστος σε σχέση με υψηλής απόδοσης μικροεπεξεργαστές.

1.8 Ενσωματωμένα συστήματα σε προγραμματιζόμενες ψηφίδες (SoPCs)

Σε μία ενσωματωμένη εφαρμογή, είναι επιθυμητή η χρήση όσο των δυνατόν λιγότερων τσιπ. Ιδανικά, ένα μόνο τσιπ θα πραγματοποιούσε ολόκληρο το σύστημα. Ο όρος σύστημα σε προγραμματιζόμενο τσιπ (System-on-a-Programmable-Chip, SoPC) έχει χρησιμοποιηθεί για να περιγράψει αυτήν την τεχνολογία. Σε απλές εφαρμογές, μερικοί διαθέσιμοι εμπορικά μικροελεγκτές μπορεί να πραγματοποιήσουν όλες τις απαραίτητες λειτουργίες. Αυτό όμως είναι απίθανο σε περιπτώσεις που οι εφαρμογές είναι πιο σύνθετες.

Σχεδιάζοντας ένα μικροελεγκτή για μία σύνθετη ενσωματωμένη εφαρμογή και τον υλοποιήσουμε με την μορφή ενός προσαρμοσμένου τσιπ είναι και ακριβό αλλά και αρκετά δύσκολο. Επίσης, είναι και αρκετά χρονοβόρο. Ακόμη, ο χρόνος ανάπτυξης για τα περισσότερα καταναλωτικά προϊόντα πρέπει να είναι σύντομος. Ένα τσιπ που υλοποιεί ένα ολόκληρο σύστημα για μία δεδομένη εφαρμογή μπορεί να αναπτυχθεί σε πολύ μικρότερο χρονικό διάστημα, εάν ο σχεδιαστής έχει πρόσβαση σε προσχεδιασμένες μονάδες κυκλώματος (*modules*), οι οποίες είναι διαθέσιμες σε μορφή που είναι εύκολη στη χρήση. Ένα κύκλωμα επεξεργαστή είναι ένα από τα αναγκαία modules. Τέτοια κυκλώματα ονομάζονται πυρήνες του επεξεργαστή σε τεχνική βιβλιογραφία. Άλλα modules που μπορούν να χρησιμοποιηθούν για την υλοποίηση μνήμης, διεπαφές εισόδου/εξόδου, κυκλώματα μετατροπής A/D και D/A, ή κυκλώματα DSP - digital signal processing (επεξεργασία ψηφιακού σήματος). Ο προγραμματιστής του συστήματος, στη συνέχεια ολοκληρώνει το σχεδιασμό χρησιμοποιώντας τα διαθέσιμα modules και σχεδιάζοντας το υπόλοιπο του απαιτούμενου κυκλώματος που είναι ειδικό για την εφαρμογή.

Το κόστος της εφαρμογής ενός ειδικά σχεδιασμένου τσιπ είναι ένας σημαντικός παράγοντας. Η κατασκευή αυτών των τσιπ είναι ακριβή και, ενώ προσφέρουν καλύτερες επιδόσεις και χαμηλότερη κατανάλωση ενέργειας, το κόστος τους μπορεί να δικαιολογηθεί μόνο εάν χρειάζεται ένα μεγάλος αριθμός από τσιπ για συγκεκριμένες εφαρμογές. Μια εναλλακτική δυνατότητα είναι η

χρησιμοποιηθεί τεχνολογία με Πίνακες Πυλών Προγραμματιζόμενων στο Πεδίο (Field Programmable Gate Arrays - FPGAs).

1.9 Διατάξεις Προγραμματιζόμενης Λογικής (PLDs)

Οι διατάξεις προγραμματιζόμενης λογικής (Programmable Logic Devices – PLDs) είναι τυποποιημένα λογικά κυκλώματα που χρησιμοποιούνται για την κατασκευή επαναπρογραμματιζόμενων ψηφιακών κυκλωμάτων και δημιουργήθηκαν την δεκαετία του 1970. Σε αντίθεση με μία λογική πύλη, η οποία έχει μία σταθερή λειτουργία, ένα PLD έχει μία απροσδιόριστη λειτουργία κατά τη στιγμή της κατασκευής. Πριν χρησιμοποιηθεί ένα PLD σε ένα κύκλωμα, πρέπει να προγραμματιστεί (δηλαδή να καθοριστεί η λογική που θα ακολουθεί το PLD), στην ουσία να επαναπρογραμματιστεί. Ο προγραμματισμός του PLD μπορεί να γίνει με τεχνολογία όπως αυτή των μνημών ROM (τηκόμενες ενώσεις-fuses).

Γιατί Προγραμματιζόμενη Λογική;

- Η παραγωγή ολοκληρωμένων κυκλωμάτων (ICs) σε μεγάλες ποσότητες συμφέρει οικονομικά.
- Πολλοί σχεδιασμοί απαιτούνται μόνο σε μικρές ποσότητες.
- Χρειάζεται ένα ολοκληρωμένο κύκλωμα που να μπορεί να παράγετε σε μεγάλες ποσότητες, αλλά και να υποστηρίζει πολλούς σχεδιασμούς σε μικρές ποσότητες.
- Μία προγραμματιζόμενη λογική μονάδα μπορεί να κατασκευαστεί σε μεγάλες ποσότητες και να προγραμματιστεί έτσι ώστε να υλοποιεί μεγάλο αριθμό διαφορετικών σχεδιασμών μικρής παραγωγής.

Επιπρόσθετα Πλεονεκτήματα

- Πολλά PLDs είναι *field-programmable*, δηλαδή μπορούν να προγραμματιστούν εκτός της εργοστασιακής κατασκευής.
- Τα περισσότερα PLDs είναι διαγράψιμα (**erasable**) και επαναπρογραμματιζόμενα (**re-programmable**).

- Επιτρέπει "ενημέρωση" μιας συσκευής ή διόρθωση λαθών.
- Επιτρέπει την επαναχρησιμοποίηση της συσκευής για διαφορετικό σχεδιασμό - καλύτερη δυνατή επαναχρησιμοποίηση.
- Ιδανικά για εργαστηριακά μαθήματα.

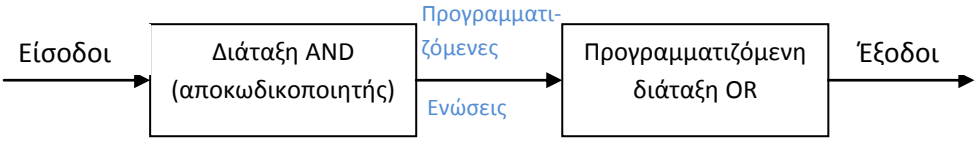
- Τα PLDs μπορούν να χρησιμοποιηθούν για ένα πρωτότυπο σχεδιασμό, ο οποίος θα υλοποιηθεί τελικά σε κανονικό ολοκληρωμένο κύκλωμα. Πολλά πρωτότυπα για την οικογένεια επεξεργαστών της Intel Pentium, υλοποιήθηκαν αρχικά από ειδικά συστήματα βασισμένα σε ένα μεγάλο αριθμό από προγραμματιζόμενες συσκευές VLSI!

Οι προγραμματιζόμενες διατάξεις που υπάρχουν και θα τις δούμε αναλυτικότερα στη συνέχεια είναι οι εξής:

- ✓ **Μνήμη Μόνο Ανάγνωσης (ROM - Read-Only Memory):** Σταθερός αριθμός από πύλες AND και προγραμματιζόμενη διάταξη πυλών OR.
- ✓ **Προγραμματιζόμενη Λογική Πίνακα (PAL - Programmable Array Logic):** Μία προγραμματιζόμενη διάταξη από πύλες AND που τροφοδοτούν μία σταθερή διάταξη πυλών OR.
- ✓ **Γενική Λογική Πίνακα (GAL - Generic Array Logic)**
- ✓ **Προγραμματιζόμενος Λογικός Πίνακας (PLA - Programmable Logic Array):** Μία προγραμματιζόμενη διάταξη από πύλες AND που τροφοδοτούν μία προγραμματιζόμενη διάταξη πυλών OR.
- ✓ **Σύνθετη Προγραμματιζόμενη Λογική Διάταξη (CPLD - Complex PLD):** Αποτελεί μια συλλογή από απλά PLDs.
- ✓ **Πίνακας Πυλών Προγραμματιζόμενος στο Πεδίο (FPGA - Field Programmable Gate Array):** Πιο πολύπλοκη δομή, την οποία θα αναλύσουμε παρακάτω.

1.9.1 Μνήμη Μόνο Ανάγνωσης (ROM - Read-Only Memory)

Πριν ανακαλυφθούν οι προγραμματιζόμενες λογικές διατάξεις, οι μνήμες μόνο για ανάγνωση (ROM) χρησιμοποιούνταν για τη δημιουργία αυθαίρετων συνδυαστικών λογικών συναρτήσεων ενός αριθμού εισόδων. Μία ROM αποτελείται από N εισόδους (γραμμές διευθύνσεων) και από M εξόδους (γραμμές δεδομένων). Δηλαδή αποτελείται συνολικά από 2^N κωδικοποιημένους ελαχιστόρους ή διευθύνσεις.



Σχήμα 1.4 : PROM - Programmable Read Only Memory [7]

Η προγραμματιζόμενη ROM (**PROM - Programmable ROM**) περιέχει αμετάβλητη διάταξη πυλών AND με 2^N εξόδους, που υλοποιεί όλους τους ελαχιστόρους (συνήθως με αποκωδικοποιητή (decoder) και προγραμματιζόμενη διάταξη OR με M εξόδους που σχηματίζει μέχρι και M αθροίσματα ελαχιστόρων [Σχήμα 1.4]. Ένα πρόγραμμα για μία ROM ή PROM είναι ένας πίνακας αληθείας με πολλαπλό αριθμό εξόδων. Δηλαδή, αν έχουμε είσοδο τότε γίνεται μία ένωση (σύνδεση) στον αντίστοιχο ελαχιστόρο για την αντίστοιχη έξοδο. Εάν έχουμε 0, δεν γίνεται καμία ένωση.

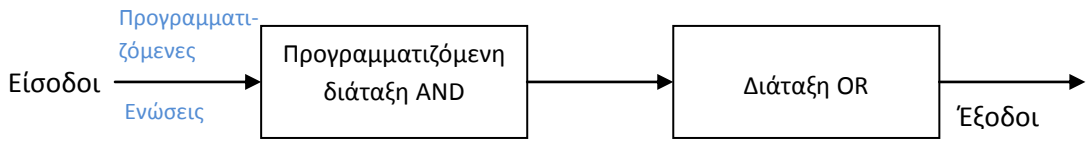
Το **πλεονέκτημα** της ROM είναι ότι κάθε πιθανή συνάρτηση όλων των δυνατών συνδυασμών των N εισόδων μπορεί να γίνει για να εμφανιστεί σε οποιαδήποτε από τις M εξόδους, καθιστώντας τη, στην πιο γενικής χρήσης συνδυαστική λογική διάταξη, διαθέσιμη για N εισόδους και M εξόδους.

Επίσης, διαθέσιμες εκτός από τις PROMs, είναι οι διαγράψιμες PROMs (**EPROMs - Erasable PROMs**) και οι ηλεκτρικά διαγράψιμες PROMs (**EEPROMs - Electrically Erasable PROMs**), οι οποίες μπορούν να προγραμματιστούν χρησιμοποιώντας μία πρότυπη PROM χωρίς να απαιτούνται εξειδικευμένο υλικό ή λογισμικό.

Ωστόσο, έχουν και αρκετά **μειονεκτήματα**:

- είναι συνήθως πολύ πιο αργές από ότι πρέπει στα λογικά κυκλώματα,
- δεν μπορούν να παρέχουν απαραίτητα ασφαλείς "καλύψεις" για ασύγχρονες λογικές μεταπτώσεις έτσι οι έξοδοι των PROMs μνημών μπορεί να πάθουν βλάβη καθώς οι είσοδοι εναλλάσσονται,
- καταναλώνουν περισσότερη ενέργεια,
- συχνά είναι πιο ακριβές από προγραμματιζόμενες λογικές, ειδικά αν απαιτείται υψηλή ταχύτητα.

1.9.2 Προγραμματιζόμενη Λογική Πίνακα (PAL - Programmable Array Logic)



Σχήμα 1.5: PAL - Programmable Array Logic [7]

Η διάταξη των PLAs είναι η αντίθετη των μνημών ROMs, δηλαδή αποτελούνται από μία προγραμματιζόμενη διάταξη από ANDs συνδυασμένη με αμετάβλητες πύλες ORs [Σχήμα 1.5].

Τα **μειονεκτήματα** σε σχέση με τις ROMs είναι ότι ενώ οι μνήμες ROMs εγγυούνται την υλοποίηση οποιονδήποτε M συναρτήσεων με N εισόδους, τα PALs είναι περιοριστικά, αφού έχουν συγκεκριμένο αριθμό εισόδων στις πύλες OR.

Τα **πλεονεκτήματα** της χρήσης της PAL είναι:

- Ένα PAL μπορεί να έχει μεγαλύτερες εισόδους N και συναρτήσεις M, για δεδομένη εσωτερική πολυπλοκότητα.
- Κάποια PALs έχουν εξόδους που μπορούν να συμπληρωθούν, προσθέτοντας συναρτήσεις POS.
- Δεν υπάρχουν υλοποιήσεις πολλαπλών επιπέδων στις ROMs (χωρίς εξωτερικές ενώσεις από την έξοδο στην είσοδο). Το PAL έχει εξόδους από όρους OR ως

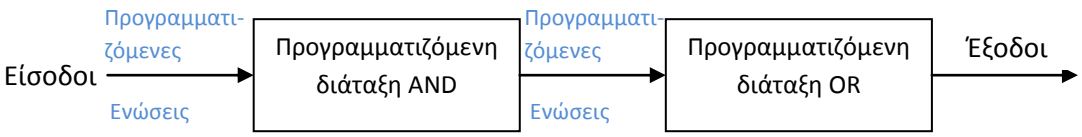
εσωτερικούς εισόδους σε όλους τους όρους AND, κάνοντας την υλοποίηση κυκλωμάτων πολλαπλών επιπέδων εφικτή.

1.9.3 Γενική Λογική Πίνακα (GAL - Generic Array Logic)

Μια καινοτομία του PAL ήταν η Γενική Λογική Πίνακα (GAL - Generic Array Logic), που εφευρέθηκε από την εταιρία Lattice Semiconductor το 1985. Η διάταξη του GAL έχει τις ίδιες λογικές ιδιότητες όπως το PAL, αλλά μπορεί να διαγραφεί και να επαναπρογραμματιστεί. Το GAL είναι πολύ χρήσιμο στο στάδιο της κατασκευής πρωτοτύπων ενός σχεδίου, όταν τυχόν σφάλματα στην λογική μπορεί να διορθωθεί με τον επαναπρογραμματισμό.

Τα GALs συνδυάζουν τεχνολογία CMOS και ηλεκτρικά διαγράψιμη τεχνολογία πυλών για μία χαμηλής και υψηλής ισχύος λογική διάταξη.

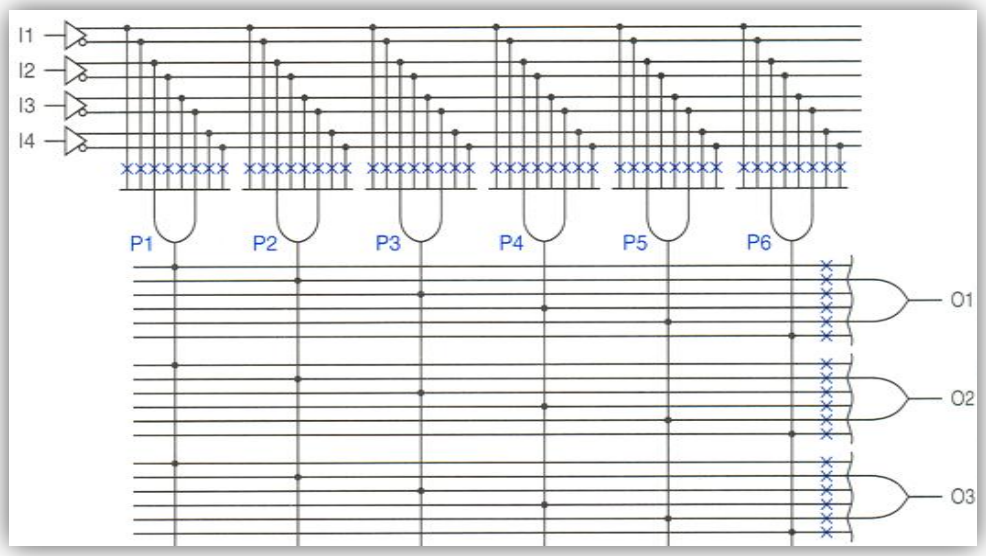
1.9.4 Προγραμματιζόμενος Λογικός Πίνακας (PLA - Programmable Logic Array)



Σχήμα 1.6: PLA - Programmable Logic Array [7]

Συγκριτικά με τα ROMs και PALs, ο Προγραμματιζόμενος Λογικός Πίνακας - PLA είναι πιο ευέλικτος, αφού έχει μία προγραμματιζόμενη διάταξη από πύλες AND συνδυασμένη με μία προγραμματιζόμενη διάταξη από πύλες OR [Σχήμα 1.6].

Επομένως, ένα PLA είναι ένας προγραμματιζόμενος πίνακας AND-OR που μπορεί να υλοποιήσει συναρτήσεις που μπορούν να γραφούν σαν αθροίσματα γινομένων. Έτσι, η παρακάτω διάταξη [Σχήμα 1.7] μπορεί να υλοποιήσει τρεις συναρτήσεις που περιέχουν μέχρι έξι όρους αθροισμάτων των τεσσάρων μεταβλητών εισόδου ο καθένας.



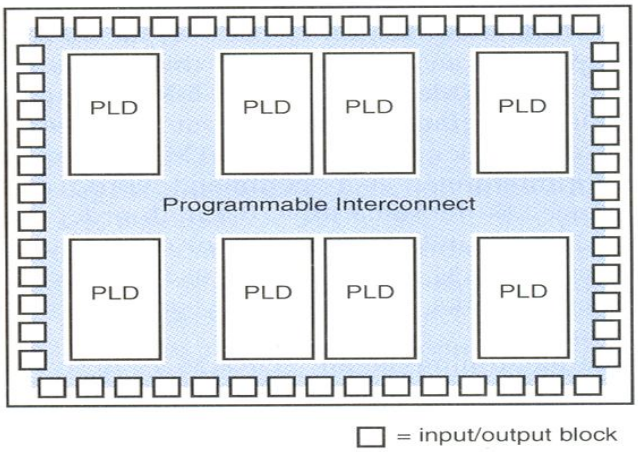
Σχήμα 1.7: Λογικό διάγραμμα ενός PLA [8]

Τα πλεονεκτήματα του PLA είναι τα εξής:

- Ένα PLA μπορεί να έχει μεγάλο αριθμό N εισόδων και M συναρτήσεων και έτσι επιτρέπει την υλοποίηση εξισώσεων που είναι μη πρακτικές για ένα ROM.
- Ένα PLA επιτρέπει σε όλους τους όρους γινομένου να μπορούν να ενωθούν με όλες τις εξόδους, ξεπερνώντας το πρόβλημα των λιγοστών εισόδων στα PALs.
- Κάποια PLAs έχουν εξόδους που μπορούμε να πάρουμε το συμπλήρωμα τους, προσθέτοντας έτσι συναρτήσεις POS.

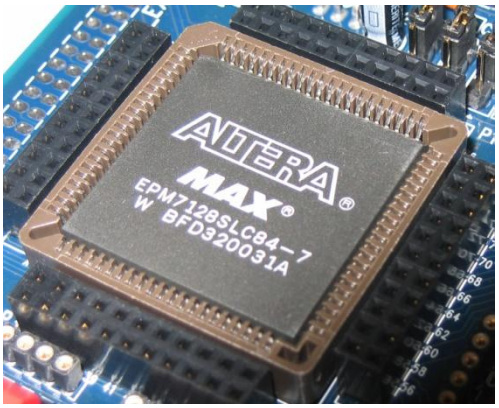
Ενώ ένα μειονέκτημα των PLAs είναι ότι συχνά, ο αριθμός όρων γινομένου μειώνει την εφαρμογή ενός PLA. Η βελτιστοποίηση 2-επιπέδων κυκλωμάτων με πολλαπλές εξόδους μειώνει τον αριθμό των όρων γινομένων σε μία υλοποίηση, βοηθώντας έτσι την προσαρμογή του σε ένα PLA.

1.9.5 Σύνθετη Προγραμματιζόμενη Λογική Διάταξη (Complex PLD - CPLD)



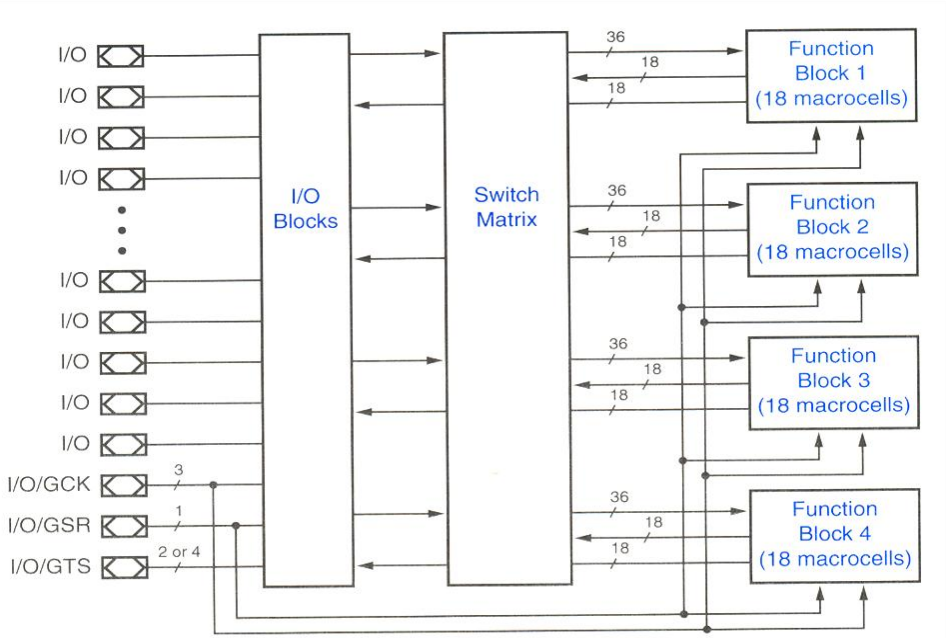
Εικόνα 1.4: Complex PLD - CPLD [8]

Τα PALs και τα GALS είναι διαθέσιμα μόνο σε μικρά μεγέθη, που είναι ισοδύναμα με μερικές εκατοντάδες λογικές πύλες. Για μεγαλύτερα λογικά κυκλώματα, μπορούν να χρησιμοποιηθούν τα σύνθετα PLDs (Complex PLDs) ή αλλιώς CPLDs. Τα CPLDs μπορούν να αντικαταστήσουν χιλιάδες, ή ακόμη και εκατοντάδες χιλιάδες, των λογικών πυλών.

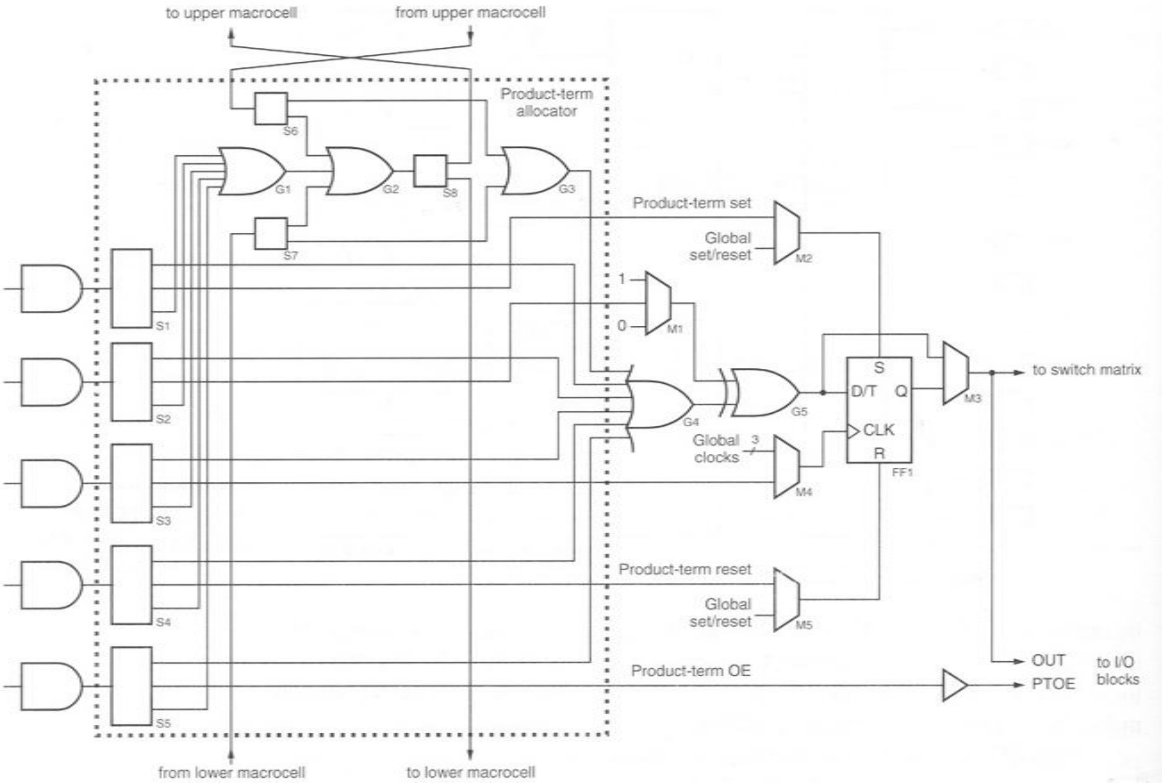


Εικόνα 1.5: MAX 7000-series CPLD με 2500 πύλες της ALTERA. [9]

Ένα CPLD είναι μια σύνθετη προγραμματιζόμενη λογική διάταξη, που αποτελεί μία συλλογή από απλά PLDs πάνω σε ένα μοναδικό chip [Εικόνα 1.4]. Αποτελείται από μία δομή προγραμματιζόμενων διασυνδέσεων, από έναν αριθμό κυκλωμάτων εισόδου/εξόδου, από ένα σύνολο ακροδεκτών εισόδου/εξόδου και από λειτουργικές βαθμίδες [Σχήμα 1.8]. Ο προγραμματιζόμενος πίνακας AND και ο σταθερός πίνακας OR αποτελούν τη λειτουργική βαθμίδα (Function Block). Η κάθε λειτουργική βαθμίδα υλοποιεί την δική της λογική συνάρτηση και αποτελείται από 18 ή περισσότερες μακροκυψέλες.



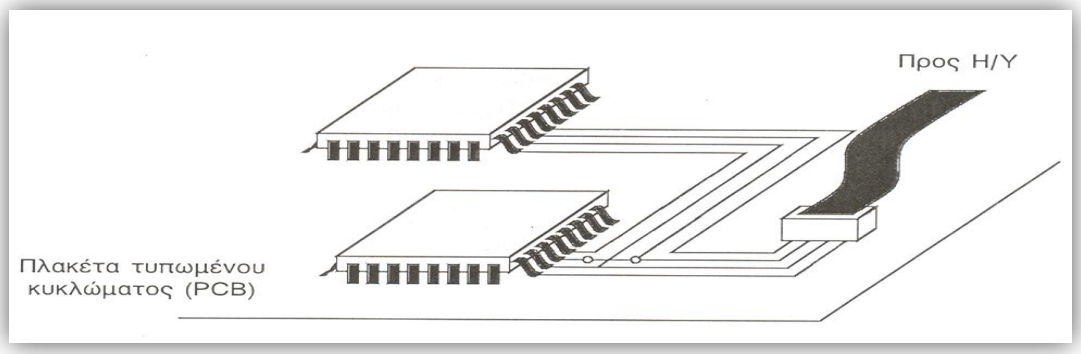
Σχήμα 1.8: Βασική αρχιτεκτονική ενός CPLD [8]



Σχήμα 1.9: Αρχιτεκτονική της μακροκυψέλης ενός CPLD [8]

Στην παραπάνω διάταξη [Σχήμα 1.9] παρουσιάζεται η αρχιτεκτονική της μακροκυψέλης (macrocell) που βρίσκεται μέσα στην λειτουργική βαθμίδα (function block) ενός CPLD. Αποτελείται από AND και OR λογικούς πίνακες και έναν "κατανεμητή ορών γινομένων" που στέλνει στον σταθερό πίνακα OR, όχι μόνο τα τελευταία γινόμενα από τον πίνακα AND της μακροκυψέλης, αλλά και γινόμενα που παράγονται από πίνακα AND άλλων γειτονικών μακροκυψελών. Στην εικόνα υπάρχει επίσης ένα D-FLIP FLOP που χρησιμοποιείται για την αποθήκευση του αποτελέσματος πριν αυτό εξαχθεί σε άλλες βαθμίδες. Επιπλέον, υπάρχει μία πύλη XOR, η οποία παράγει το συμπλήρωμα του αποτελέσματος της λογικής συνάρτησης. Λειτουργεί σαν μία προγραμματιζόμενη NOT, ώστε να γίνεται (αν χρειαστεί) η αντιστροφή. Τέλος, στην εικόνα 8 υπάρχει και ένας πολυπλέκτης, ο οποίος βγάζει αμέσως το αποτέλεσμα του πίνακα AND - OR ή το στέλνει μέσω του D-FLIP FLOP με καθυστέρηση.

Για τον προγραμματισμό των κυκλωμάτων με CPLDs, ορισμένοι κατασκευαστές (συμπεριλαμβανομένων των Altera και Microsemi) χρησιμοποιούν JTAG (Join Test Action Group - μέθοδος ελέγχου πλακετών τυπωμένων κυκλωμάτων χρησιμοποιώντας σάρωση ορίων, εντοπισμός σφαλμάτων) [Εικόνα 1.6].



Εικόνα 1.6: Προγραμματισμός της θύρας JTAG [8]

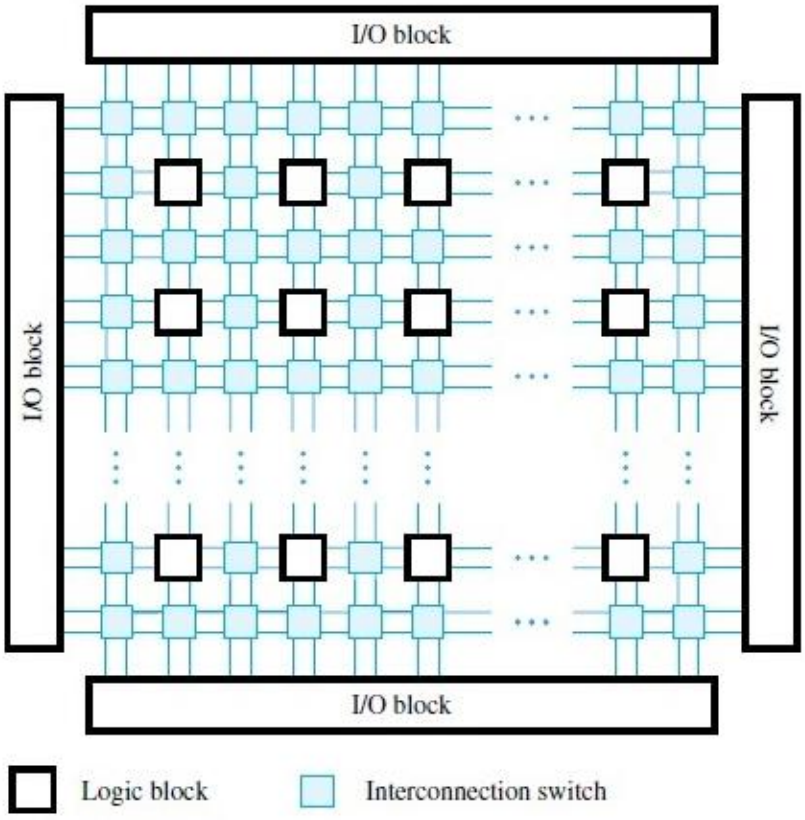
1.9.6 Πίνακας Πυλών Προγραμματιζόμενος στο Πεδίο (Field Programmable Gate Array - FPGA)

Εικόνα 1.7: Cyclone FPGA της εταιρείας ALTERA [10]

Και ενώ τα PALs αναπτύσσονταν σε GALs και CPLDs, ένα ξεχωριστό ρεύμα ανάπτυξης συνέβαινε. Αυτό το είδος των διατάξεων βασίζεται στην τεχνολογία με πίνακες πυλών και ονομάζεται πίνακας πυλών προγραμματιζόμενος στο πεδίο ή αλλιώς FPGA (Field Programmable Gate Array). Ο όρος "προγραμματιζόμενος στο πεδίο" (field programmable) σημαίνει ότι η διάταξη προγραμματίζεται από τον πελάτη και όχι από τον κατασκευαστή.

Πρώτα παραδείγματα των FPGAs είναι ο πίνακας 82S100, και ο 82S105 από την εταιρεία Signetics, οι οποίοι παρουσιάστηκαν στα τέλη της δεκαετίας του 1970. Ο πίνακας 82S100 ήταν ένας πίνακας που αποτελούνταν από όρους AND, ενώ ο 82S105 είχε επίσης και συναρτήσεις από FLIP FLOP. Στο παρελθόν τα FPGAs είχαν μεγαλύτερες απαιτήσεις σε ενέργεια σε σχέση με τις αντίστοιχες υλοποιήσεις σε ASICs και μικρότερες αποδόσεις. Η εξέλιξη των τεχνολογιών που τα συνοδεύει έχει εκμηδενίσει αυτές τις διαφορές ενώ έχει αναδειχθεί η αξία του επαναπρογραμματισμού και του γρήγορου σχεδιασμού, όπως θα αναλύσουμε παρακάτω.

Στα FPGAs η λογική κατακερματίζεται σε μικρά λογικά μπλοκ (logic blocks) που διασπείρονται σε όλη την έκταση του ολοκληρωμένου, μέσα σε μια "θάλασσα" από προγραμματιζόμενες διασυνδέσεις. Όλος ο πίνακας περιβάλλεται από προγραμματιζόμενα κυκλώματα εισόδου/εξόδου (I/O blocks) [Εικόνα 1.8].



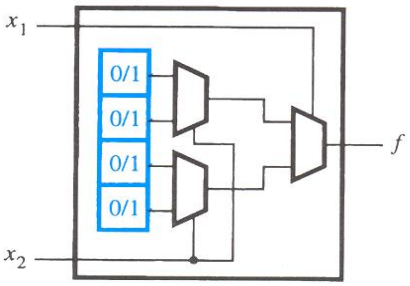
Εικόνα 1.8: Γενική δομή μίας διάταξης FPGA [11]

Πιο συγκεκριμένα, όπως φαίνεται και στο σχηματικό διάγραμμα ενός FPGA της Εικόνας 1.8, ένα FPGA αποτελείται από μία σειρά λογικών μπλοκ, τα οποία μπορούν να συνδεθούν μέσω των διακοπτών διασύνδεσης (interconnection switch). Αυτά τα λογικά μπλοκ περιέχουν συνήθως και ένα σημαντικό ποσό της μνήμης που μπορεί να χρησιμοποιηθεί για την εφαρμογή των μνημών RAM και ROM, τμήματα ενός ενσωματωμένου συστήματος, εάν οι απαιτήσεις μεγέθους της μνήμης δεν είναι πολύ μεγάλες. Πολλά FPGAs περιλαμβάνουν επίσης κυκλώματα πολλαπλασιαστών, τα οποία είναι ιδιαίτερα χρήσιμα σε εφαρμογές DSP (Digital signal processing).

Η διασύνδεση αποτελείται από τμήματα σύρματος και προγραμματιζόμενους διακόπτες. Οι διακόπτες χρησιμοποιούνται για την σύνδεση των λογικών μπλοκ στα τμήματα σύρματος και στην δημιουργία συνδέσεων μεταξύ διαφόρων τμημάτων του σύρματος. Αυτό επιτρέπει μεγαλύτερη ευελιξία στην δρομολόγηση του τσιπ. Οι ρυθμιστές (buffers) εισόδου και εξόδου παρέχονται για την πρόσβαση στα pins του τσιπ. Η διαδικασία προγραμματισμού της ρύθμισης των διακοπτών αναφέρεται ως διαμόρφωση του FPGA.

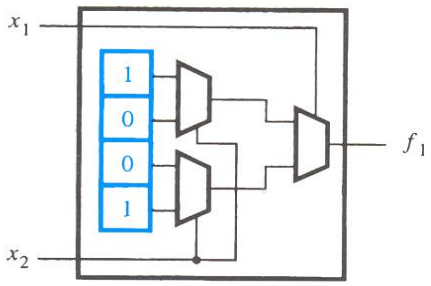
Στα περισσότερα FPGAs, η κατάσταση του κάθε προγραμματιζόμενου διακόπτη βρίσκεται σε ένα κελί μνήμης SRAM. Δεδομένου ότι τα κελιά των SRAM διατηρούν την κατάστασή τους μόνο εφ' όσον το ηλεκτρικό ρεύμα διαπερνά το FPGA, τέτοια FPGAs ονομάζονται πτητικά. Εάν δηλαδή η συσκευή απενεργοποιηθεί, το FPGA πρέπει να αναδιαμορφωθεί όταν αυτή ενεργοποιηθεί. Για να ρυθμιστεί το FPGA, οι πληροφορίες διαμόρφωσης πρέπει να φορτωθούν στην συσκευή. Αυτό γίνεται συνήθως χρησιμοποιώντας ένα άλλο τσιπ, το οποίο ονομάζεται συσκευή διαμόρφωσης, η οποία διατηρεί τις απαιτούμενες πληροφορίες σε μία μνήμη τύπου Flash. Κάθε φορά που η συσκευή είναι ενεργοποιημένη, το τσιπ διαμόρφωσης προγραμματίζει αυτόματα το FPGA.

Υπάρχει μία ποικιλία σχεδιαγραμμάτων για τα λογικά μπλοκ και την δομή των διασυνδέσεων. Ένα λογικό μπλοκ μπορεί να είναι ένα απλό κύκλωμα με βάση τον πολυπλέκτη. Ένα άλλος δημοφιλής τρόπος σχεδίασης λογικού μπλοκ είναι η χρήση ενός απλού πίνακα αναζήτησης (*lookup table - LUT*) ως ένα λογικό μπλοκ. Η λογική που εκτελεί ένα λογικό στοιχείο βρίσκεται αποθηκευμένο σε lookup table, που υλοποιείται με SRAM. Για παράδειγμα, ένας πίνακας LUT τριών εισόδων μπορεί να υλοποιηθεί με τη μορφή κυκλώματος μνήμης 8bit στην οποία αποθηκεύεται ο πίνακας αληθείας μιας λογικής συνάρτησης. Κάθε bit μνήμης αντιστοιχεί σε έναν συνδυασμό του πίνακα αληθείας ή συμπληρώνονται οι τιμές των μεταβλητών εισόδου. Ένας τέτοιος πίνακας LUT μπορεί να προγραμματιστεί για να εκτελέσει κάθε συνάρτηση των τριών μεταβλητών. (Στις παρακάτω εικόνες [Εικόνα 1.9, 1.10] υπάρχουν παραδείγματα πινάκων LUT δύο εισόδων και τριών εισόδων αντίστοιχα).



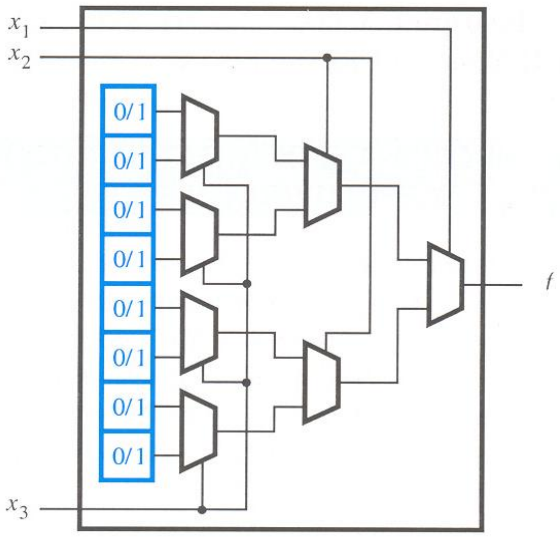
x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(a) Κύκλωμα ενός πίνακα LUT δύο εισόδων (b) $f_1 = \bar{x}_1 \bar{x}_2 + x_1 x_2$



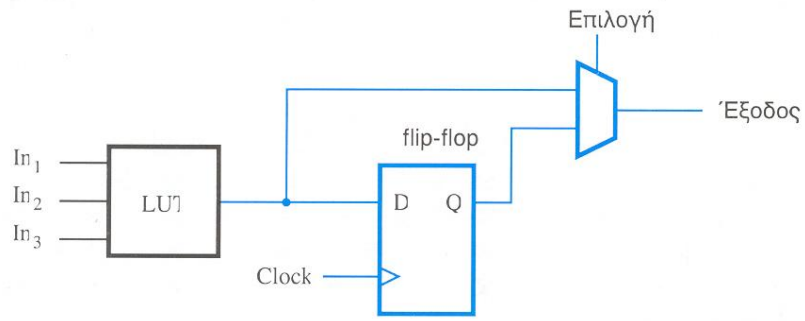
(c) Περιεχόμενα κυψελών αποθήκευσης στον πίνακα LUT

Εικόνα 1.9: Πίνακας LUT δύο εισόδων [8]



Εικόνα 1.10: Πίνακας LUT τριών εισόδων [8]

Τα λογικά μπλοκ μπορεί να περιέχουν flip-flops για να παρέχουν επιπλέον ευελιξία. [Σχήμα 1.10]

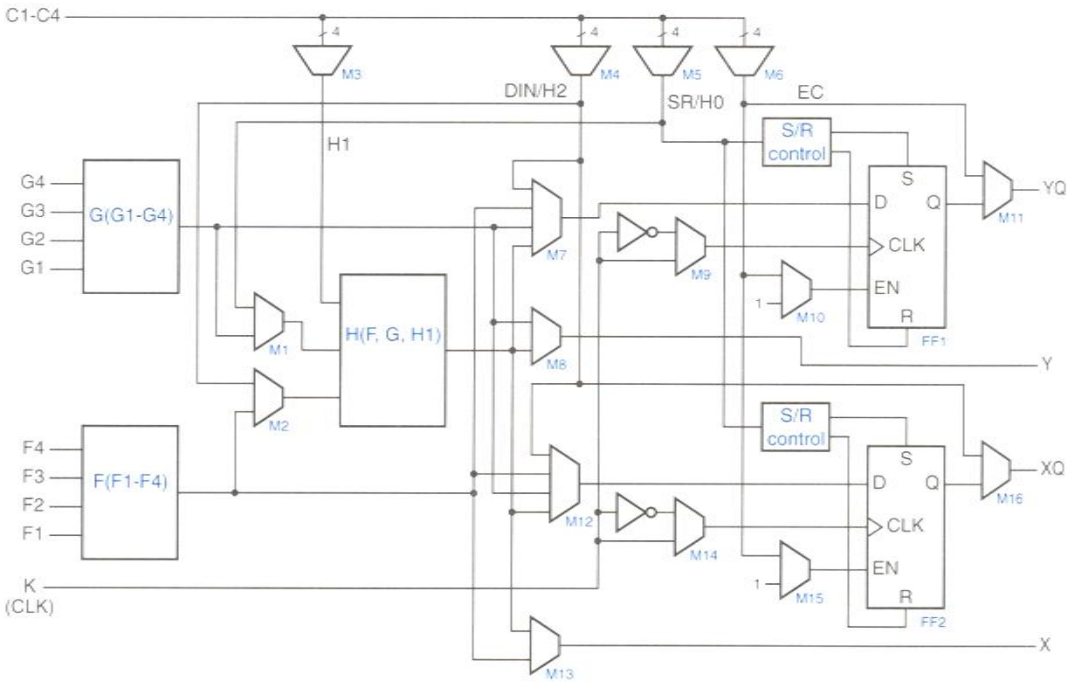


Σχήμα 1.10: Τοποθέτηση ενός flip-flop σε ένα λογικό μπλοκ FPGA [8]

Εκτός από τα λογικά μπλοκ, πολλά FPGAs περιλαμβάνουν ένα σημαντικό αριθμό κελιών μνήμης, που μπορούν να χρησιμοποιηθούν για παράδειγμα στις ουρές με τη μέθοδο first-in-first-out (FIFO) ή ως RAM και ROM μνήμες σε εφαρμογές system-on-a-chip.

Τα FPGAs είναι διαθέσιμα σε ένα ευρύ φάσμα μεγεθών. Τα μεγαλύτερα FPGAs περιέχουν δισεκατομμύρια τρανζίστορ και μπορούν να χρησιμοποιηθούν για να εκτελέσουν πολύ μεγάλα κυκλώματα. Η αυξανόμενη ζήτηση των FPGAs οφείλεται στο γεγονός ότι επιτρέπουν ένα σχεδιαστή να εφαρμόσει πολύ περίπλοκα λογικά κυκλώματα και μεγάλα ψηφιακά συστήματα σε ένα μόνο τσιπ, χωρίς να χρειάζεται να σχεδιάσει και να κατασκευάσει ένα προσαρμοσμένο VLSI chip, το οποίο είναι και δαπανηρό αλλά και χρονοβόρο. Χρησιμοποιώντας εργαλεία CAD, είναι δυνατό να δημιουργηθεί μία διάταξη FPGA μέσα σε λίγες ημέρες, αντί τους μήνες που χρειάζεται για να παραχθεί ένα πρότυπο σχεδιασμένο VLSI chip.

Το αντίστοιχο λογικό μπλοκ σε διατάξεις FPGA της εταιρείας Xilinx ονομάζεται διαμορφούμενο λογικό μπλοκ (*Configurable Logic Block* - CBL) ή αλλιώς στην βιβλιογραφία αναφέρεται και ως *slice* [Σχήμα 1.11].



Σχήμα 1.11: Διαμορφούμενη λογική βαθμίδα (Configuration Logic Block - CBL) [8]

Η *διαφορά μεταξύ των FPGAs και CPLDs* είναι ότι τα FPGAs βασίζονται κυρίως στους πίνακες αναζήτησης (look-up tables - LUTs), ενώ τα CPLDs σχηματίζουν τις λογικές τους συναρτήσεις με μία "θάλασσα" από πύλες. Άλλη μία διαφορά είναι ότι τα CPLDs προορίζονται για απλούστερα σχέδια ενώ τα FPGAs για πιο πολύπλοκα σχέδια. Σε γενικές γραμμές, τα CPLDs είναι μια καλή επιλογή για εφαρμογές ευρείας συνδυαστικής λογικής, ενώ τα FPGAs είναι πιο κατάλληλα για πολυπλοκότερες εφαρμογές (όπως οι μικροεπεξεργαστές). Επίσης, ο προγραμματισμός των FPGAs διαφέρει από αυτόν των CPLDs αφού στηρίζεται σε μνήμες SRAM και όχι σε τηκόμενες συνδέσεις.

Τα τελευταία χρόνια υπάρχει μια μεγάλη κλίση προς τον σκληρό επεξεργαστή, η οποία οφείλεται κυρίως στην ανάγκη για ταχύτερη επεξεργασία, την οποία οι μαλακοί πυρήνες δεν μπορούν να παρέχουν. Για παράδειγμα, όταν βάζουμε έναν επεξεργαστή με σκληρό πυρήνα, μπορούμε να ενεργοποιήσουμε τυπικά την επεξεργασία πολλών δεδομένων, τα οποία είναι απαραίτητα για εφαρμογές υποδομής επικοινωνιών (περνούν πολλά Giga Bytes δεδομένων).

Τα σύγχρονα FPGAs ενσωματώνουν στις αναπτυξιακές πλακέτες τους περιφερειακά, μνήμες, ενσωματωμένους επεξεργαστές και άλλα κυκλώματα ειδικού σκοπού. Έτσι επιτυγχάνεται η αύξηση των δυνατοτήτων τους δημιουργώντας υβριδικά συστήματα που συνδυάζουν τη χρήση υλικού με επαναπρογραμματιζόμενα στοιχεία.

Ο προγραμματισμός των FPGA γίνεται με τη χρήση των γλωσσών περιγραφής υλικού (Hardware Description Language - HDL) ή σχηματική περιγραφή του συστήματος. Οι γλώσσες περιγραφής υλικού [Ενότητα 1.6.1] που χρησιμοποιούνται είναι η VHDL και η Verilog, παρέχοντας ένα αφαιρετικό επίπεδο στον σχεδιαστή του συστήματος. Δίνουν τη δυνατότητα υλοποίησης των ολοκληρωμένων κυκλωμάτων με περιγραφή της επιθυμητής λογικής και χρονικής συμπεριφοράς και αρχιτεκτονικής τους.

Οι εταιρείες που κυριαρχούν στο χώρο των FPGA είναι η ALTERA και η Xilinx με τις αντίστοιχες οικογένειες FPGA FLEX και XC. Στην πτυχιακή αυτή εργασία χρησιμοποιείται η αναπτυξιακή πλακέτα DE2 της οικογένειας Cyclone II της εταιρείας ALTERA.

1.9.6.α. Επιλογή Επεξεργαστή

Το βασικό συστατικό κάθε συστήματος σε ένα τσιπ είναι ο πυρήνας του επεξεργαστή. Υπάρχουν δύο εναλλακτικές για συστήματα που βασίζονται σε FPGA. Η μία εναλλακτική περιλαμβάνει έναν επεξεργαστή, ο οποίος καθορίζεται σε λογισμικό και εφαρμόζεται στο FPGA με τον ίδιο τρόπο όπως και κάθε άλλο κύκλωμα. Η άλλη περιλαμβάνει ένα εξειδικευμένο τσιπ FPGA που έχει έναν πυρήνα επεξεργαστή που εφαρμόζεται στο τσιπ κατά τη στιγμή της κατασκευής του.

Μαλακός Πυρήνας Επεξεργαστή

Η πιο ευέλικτη λύση είναι να παρέχεται λογισμικό γραμμένο σε γλώσσα περιγραφής υλικού, όπως Verilog ή VHDL, το οποίο καθορίζει έναν παραμετροποιημένο επεξεργαστή. Ο σχεδιαστής ενός ενσωματωμένου συστήματος μπορεί να καθορίσει τις παραμέτρους για την εξασφάλιση ενός επεξεργαστή με κατάλληλα χαρακτηριστικά για την προβλεπόμενη εφαρμογή. Για παράδειγμα, μία παράμετρος σχετίζεται με την διαμόρφωση της κρυφής μνήμης (cache), όπου οι επιλογές μπορεί να είναι:

- Χωρίς κρυφή μνήμη
- Κρυφή μνήμη εντολών, αλλά να μην υπάρχει κρυφή μνήμη δεδομένων
- Να υπάρχει κρυφή μνήμη εντολών και δεδομένων.

Μία άλλη παράμετρος μπορεί να σχετίζεται με την ένταξη κυκλωμάτων πολλαπλασιαστών και διαιρετών στον επεξεργαστή. Οι λειτουργίες πολλαπλασιασμού και διαίρεσης μπορούν να εφαρμοστούν στο υλικό (hardware), αλλά μπορούν επίσης να υλοποιηθούν σε λογισμικό (software). Η υλοποίησή τους σε hardware χρησιμοποιεί περισσότερους πόρους από το FPGA, αλλά βελτιώνει σημαντικά την απόδοση.

Σκληρός Πυρήνας Επεξεργαστή

Μία εναλλακτική λύση στην προσέγγιση του μαλακού πυρήνα επεξεργαστή είναι η εφαρμογή του επεξεργαστή άμεσα, ως μέρος του hardware πάνω στο τσιπ, δημιουργώντας έτσι ένα εξειδικευμένο FPGA. Αυτό έχει ως αποτέλεσμα την καλύτερη απόδοση του συστήματος. Το κόστος βέβαια αυτών των FPGAs είναι υψηλότερο από το κόστος των κανονικών FPGAs.

Παρακάτω δίνονται μερικά παραδείγματα με αναπτυξιακές πλακέτες με FPGA των εταιρειών ALTERA και Xilinx. [Εικόνες 1.11, 1.12]

Εικόνα 1.11: Αναπτυξιακή πλακέτα Stratix II GX PCIe της εταιρείας ALTERA. [8]

Εικόνα 1.12: Αναπτυξιακή πλακέτα Digilent Atlys Spartan 6 FPGA της εταιρείας Xilinx [13]

1.9.6.β. Υπολογιστικά Εργαλεία Σχεδιασμού για το FPGA

Τα κύρια υπολογιστικά εργαλεία σχεδιασμού (CAD tools - Computer Aided Design tools) της εταιρείας Altera είναι γνωστά ως λογισμικό Quartus II. Περιλαμβάνουν μία πλήρη σειρά από εργαλεία που χρειάζονται για το σχεδιασμό και την υλοποίηση ενός ψηφιακού συστήματος σε ένα τσιπ FPGA. Ένα από αυτά τα εργαλεία ονομάζεται SOPC Builder ή Qsys tool, το οποίο μπορεί να χρησιμοποιηθεί για το σχεδιασμό συστημάτων που περιλαμβάνουν ένα πυρήνα επεξεργαστή. Περιλαμβάνει έναν αριθμό από παραμετροποιημένες ενότητες (modules), που μπορούν να χρησιμοποιηθούν στο σχεδιασμένο σύστημα. Για την απεικόνιση της φύσης τους, παρακάτω θα αναφέρουμε τέσσερα από αυτά τα modules. (Στο Κεφάλαιο 4 αναλύονται εκτενέστερα το λογισμικό Quartus II και οι ενότητές του.)

Επεξεργαστής Nios II

Για την εφαρμογή σε FPGAs, παρέχεται σε τρεις εκδόσεις: την οικονομική έκδοση (***Nios II/e****, "economy" version*), την πρότυπη (***Nios II/s****, "standard" version*) και την γρήγορη (***Nios II/f****, "fast" version*). Η "economy" version είναι και η απλούστερη και λιγότερο δαπανηρή για υλοποίηση (από άποψη πόρων του FPGA). Επίσης έχει και την χαμηλότερη απόδοση. Δεν περιλαμβάνει καθόλου κρυφές μνήμες, δεν έχει διοχέτευση και δεν χρησιμοποιεί πρόβλεψη διακλάδωσης. Καλύτερη απόδοση επιτυγχάνεται με την "standard" version, η οποία περιλαμβάνει μία εντολή κρυφής μνήμης, παρέχει διοχέτευση και χρησιμοποιεί στατική πρόβλεψη διακλάδωσης. Η καλύτερη απόδοση επιτυγχάνεται με την "fast" version, η οποία περιλαμβάνει και εντολές και δεδομένα για κρυφή μνήμη, και χρησιμοποιεί δυναμική πρόβλεψη διακλάδωσης.

Υπάρχουν αρκετές παράμετροι που ένας σχεδιαστής μπορεί να προσδιορίσει, συμπεριλαμβάνοντας τα μεγέθη των εντολών και των δεδομένων κρυφής μνήμης. Οι πυρήνες του επεξεργαστή Nios II είναι αρκετά μικροί ώστε να καταλαμβάνουν μόνο ένα μικρό μέρος του FPGA. Είναι δυνατόν να εφαρμοστούν μέχρι και δέκα

πυρήνες Nios II σε ένα σχετικά μικρό FPGA. (Ο επεξεργαστής Nios II περιγράφεται αναλυτικότερα στο επόμενο κεφάλαιο)

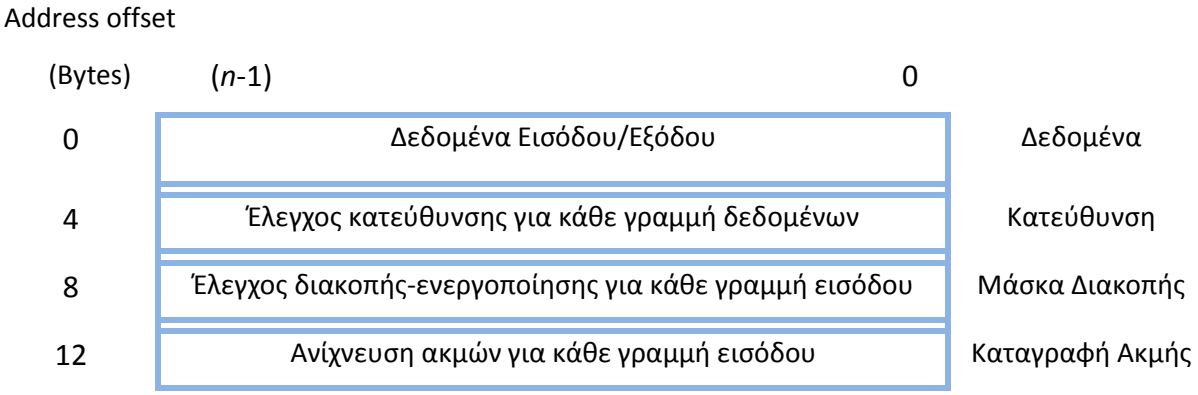
Μνήμη

Τα μπλοκ της μνήμης σε ένα τσιπ FPGA μπορούν να χρησιμοποιηθούν για την υλοποίηση κρυφής μνήμης και ενός τμήματος της κύριας μνήμης. Το τμήμα της κύριας μνήμης, που πραγματοποιείται με αυτόν τον τρόπο, αναφέρεται ως μνήμη πάνω σε τσιπ (*on-chip memory*). Αυτή η μνήμη μπορεί να ρυθμιστεί με διάφορους τρόπους. Μπορεί να υλοποιηθεί είτε ως μνήμη τύπου RAM ή τύπου ROM. Το μέγεθος και το μήκος λέξης του μπορούν να καθοριστούν κατά τη στιγμή του σχεδιασμού. Ο SOPC Builder ή το εργαλείο Qsys καθιστά εύκολη την δημιουργία των ελεγκτών και των διασυνδέσεων που χρειάζεται για να συνδεθεί ένα σύστημα, που υλοποιείται στο FPGA, με ένα πλήθος εξωτερικών εξαρτημάτων της μνήμης, όπως SRAMs, SDRAMs και συσκευές Flash.

Παράλληλη Διασύνδεση Εισόδου/Εξόδου

Μία παράλληλη διασύνδεση, που ονομάζεται αλλιώς *PIO* (Parallel Input/Output), είναι μία παραμετροποιημένη μονάδα που μπορεί να χρησιμοποιηθεί για σκοπούς εισόδου αλλά και εξόδου. Η θύρα δεδομένων της μπορεί να επιλεγεί κατά την στιγμή του σχεδιασμού για να χρησιμεύσει ως:

- Θύρα εισόδου
- Θύρα εξόδου
- Αμφίδρομη θύρα



Σχήμα 1.12: Καταχωρητές στην διασύνδεση PIO [11]

Εάν η αμφίδρομη επιλογή είναι επιλεγμένη, τότε οι γραμμές δεδομένων της PIO πρέπει να συνδεθούν με τις ακίδες (pins) του FPGA, που έχουν ικανότητα τριών καταστάσεων (tristate).

Ο επεξεργαστής αποκτά πρόσβαση σε μία διασύνδεση PIO ως μία διασύνδεση απεικονισμένη στη μνήμη. Οι καταχωρητές PIO φαίνονται στο Σχήμα 1.12. Το μέγεθος του καταχωρητή *n* είναι μια παράμετρος που καθορίζεται κατά τη στιγμή σχεδίασης, και μπορεί να κυμαίνεται από το 1 έως 32. Οι καταχωρητές χρησιμοποιούνται ως εξής:

- Ο καταχωρητής δεδομένων (*Data* register) κρατά τα *n* bits των δεδομένων, που μεταφέρονται μεταξύ του επεξεργαστή και της διασύνδεσης PIO.
- Ο καταχωρητής κατεύθυνσης (*Direction* register) καθορίζει την κατεύθυνση της μεταφοράς (εισόδου ή εξόδου) για κάθε μία από τις *n* γραμμές δεδομένων, όταν μία αμφίδρομη θύρα είναι εφαρμοσμένη.
- Ο καταχωρητής μάσκας διακοπής (*Interrupt-mask* register) χρησιμοποιείται για την ενεργοποίηση διακοπών από τις γραμμές εισόδου συνδέονται με την PIO. Μεμονωμένες αιτήσεις διακοπής μπορούν να τεθούν σε κάθε μία από τις *n* πιθανές γραμμές εισόδου.
- Ο καταχωρητής ανίχνευσης ακμών (*Edge-capture* register) υποδεικνύει αλλαγές στις λογικές τιμές που εντοπίζονται στα σήματα στις γραμμές εισόδου που συνδέονται με την PIO. Το είδος της ακμής (ανοδική ή καθοδική) που εντοπίζονται καθορίζεται κατά τη στιγμή της σχεδίασης.

Οι γραμμές που συνδέουν την PIO σε μία συσκευή I/O μπορούν να ρυθμιστούν μεμονωμένα. Εάν η διασύνδεση PIO εξυπηρετεί μόνο ως θύρα εισόδου, τότε όλες οι *n* γραμμές ρυθμίζονται κατά τη στιγμή της σχεδίασης ως είσοδοι. Ομοίως, , για μία θύρα εξόδου, όλες οι γραμμές ρυθμίζονται ως έξοδοι. Σε αυτές τις περιπτώσεις, ο καταχωρητής Direction δεν είναι απαραίτητος, και δεν εφαρμόζεται στο κύκλωμα που δημιουργείται. Ο καταχωρητής Direction περιλαμβάνεται στην περίπτωση που η θύρα είναι αμφίδρομη και λειτουργεί με τον εξής τρόπο: όταν η τιμή του *k* bit είναι ίση με 1, η γραμμή *k* της θύρας λειτουργεί ως μία έξοδος στη συνδεδεμένη συσκευή I/O. Ενώ αντίθετα όταν η τιμή του *k* bit είναι ίση με 0, η γραμμή *k* της θύρας λειτουργεί ως μία είσοδος από τη συνδεδεμένη συσκευή I/O.

Όταν η διασύνδεση PIO χρησιμοποιείται ως θύρα εισόδου, ο καταχωρητής Data περιέχει τις λογικές τιμές που υπάρχουν δεδομένη στιγμή στις γραμμές εισόδου. Αλλαγές στις λογικές τιμές στις γραμμές εισόδου μπορούν να εντοπιστούν χρησιμοποιώντας τον καταχωρητή Edge-capture. Κατά το σχεδιασμό, είναι δυνατό να προσδιοριστεί ότι τα bits σε αυτόν τον καταχωρητή ορίζονται ως 1 όταν συμβεί μία ακμή. Η ακμή μπορεί να καθοριστεί ως: ανοδική, καθοδική, ή άλλη ακμή. Τα bits στον καταχωρητή ανίχνευσης ακμών εκκαθαρίζονται από μία εντολή, η οποία κάνει τον καταχωρητή 0.

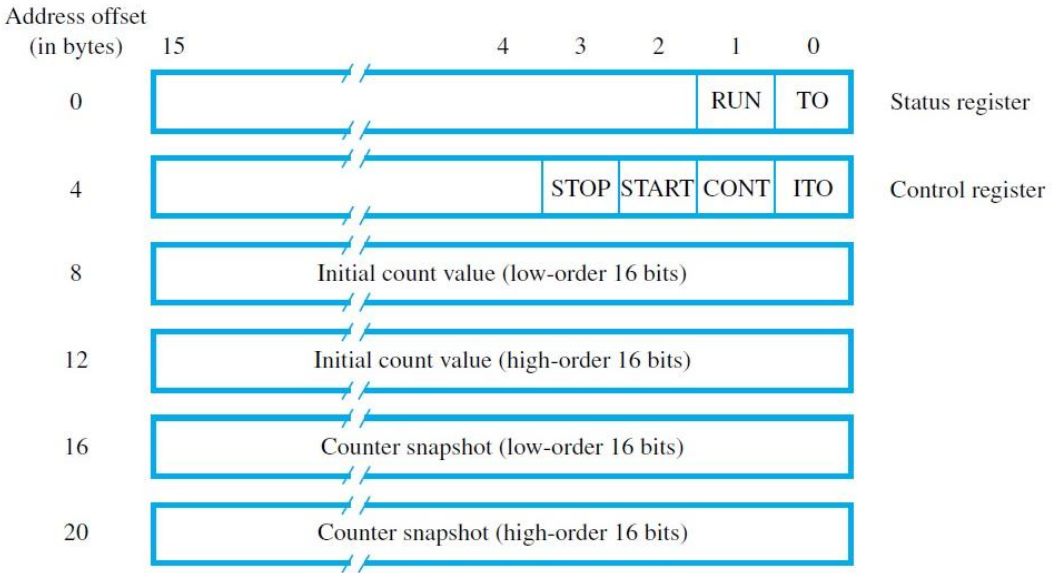
Ο καταχωρητής Interrupt-mask επιτρέπει την ενεργοποίηση και τη απενεργοποίηση των διακοπών. Γράφοντας 1 στη θέση του *k* bit του καταχωρητή ενεργοποιούνται οι διακοπές που προκαλούνται από την δραστηριότητα στην *k* γραμμή εισόδου. Οι διακοπές μπορεί να είναι ως εξής:

- *Ευαίσθητο επίπεδο*, στην περίπτωση αυτή μία αίτηση διακοπής αυξάνεται όταν το σήμα σε κάθε ενεργοποιημένη γραμμή εισόδου έχει την τιμή 1
- *Ευαίσθητη ακμή*, στην περίπτωση αυτή μία αίτηση διακοπής αυξάνεται όταν κάθε ενεργοποιημένο bit στον καταχωρητή ανίχνευσης ακμών έχει την τιμή 1

Χρονομετρητής Διαστημάτων (Interval timer)

Βασικό μέρος της μονάδας του χρονομετρητή είναι ο απαριθμητής (counter) του οποίου τα περιεχόμενα μειώνονται κατά ένα σε κάθε κύκλο ρολογιού. Ο counter μπορεί να καθοριστεί να είναι είτε 32 ή 64 bits μήκος. Ας υποθέσουμε ότι ο counter έχει 32 bits. Στο παρακάτω σχήμα απεικονίζονται οι καταχωρητές της διασύνδεσης του Interval timer [Σχήμα 1.13]. Κάθε καταχωρητής έχει μήκος 16 bits. Στον καταχωρητή Status, μόνο δύο bits χρησιμοποιούνται:

- Το bit *RUN* ισούται με 1 όταν ο counter λειτουργεί. Αλλιώς αυτό είναι ίσο με 0. Αυτό το bit δεν επηρεάζεται εάν ο επεξεργαστής γράφει στον Status register.
- Το bit *TO* είναι το bit του χρονικού ορίου. Ορίζεται σε 1 όταν ο counter φθάνει το 0. Παραμένει στο 1 μέχρι ο επεξεργαστής να το καθαρίσει γράφοντας 0 στη θέση του.



Σχήμα 1.13: Καταχωρητές της διασύνδεσης Interval timer [11]

Στον καταχωρητή ελέγχου (*Control* register) τέσσερα bits χρησιμοποιούνται:

- Το bit *STOP* ορίζεται 1 για να σταματήσει τον counter.
- Το bit *START* ορίζεται 1 για να προκαλέσει τον counter να αρχίσει να λειτουργεί.
- Το bit *CONT* καθορίζει την συμπεριφορά του counter όταν φθάνει στο 0. Εάν το CONT = 0, ο counter σταματάει όταν φθάσει στο 0. Εάν το CONT = 1, ο counter φορτώνει πάλι την αρχική τιμή του και συνεχίζει να λειτουργεί.

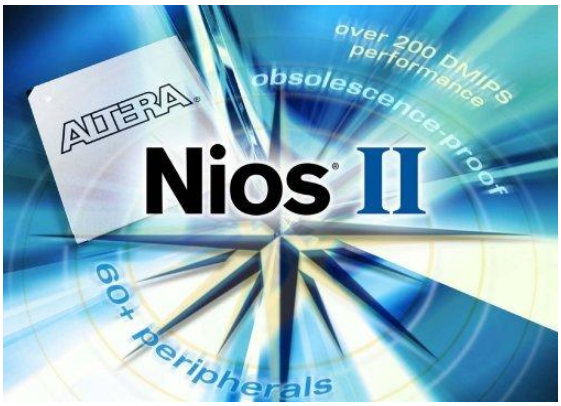
- Το bit *ITO* επιτρέπει τις διακοπές όταν έχει οριστεί σε 1.

Η αρχική τιμή του counter (*Initial count value*) πρέπει να φορτωθεί σε δύο λειτουργίες εγγραφής των 16-bit [Σχήμα 1.13]. Οι καταχωρητές του στιγμιότυπου του μετρητή (*Counter Snapshot*) χρησιμοποιούνται για να πάρουμε ένα στιγμιότυπο των περιεχομένων του counter ενώ βρίσκεται σε λειτουργία. Η λειτουργία εγγραφής σε οποιοδήποτε από τους Status register προκαλεί την λήψη ενός στιγμιότυπου, που σημαίνει ότι τα τρέχοντα περιεχόμενα του counter φορτώνονται στους καταχωρητές Snapshot. Στη συνέχεια, αυτοί οι καταχωρητές μπορούν να διαβαστούν με το συνηθισμένο τρόπο.

Εκτός από την ικανότητα της χρήσης της Initial count value για να οριστεί μία περίοδο με χρονικό όριο, μία περίοδος με προκαθορισμένο χρονικό όριο μπορεί να καθοριστεί κατά το χρόνο σχεδίασης. Ο προκαθορισμένος χρόνος της περιόδου χρησιμοποιείται εάν η Initial count value είναι 0.

Μία αίτηση διακοπής τίθεται όταν το bit TO = 1 και το ITO έχει οριστεί σε 1. Για να καταργηθεί αυτήν η αίτηση, ο επεξεργαστής πρέπει να γράψει το 0 στο bit TO.

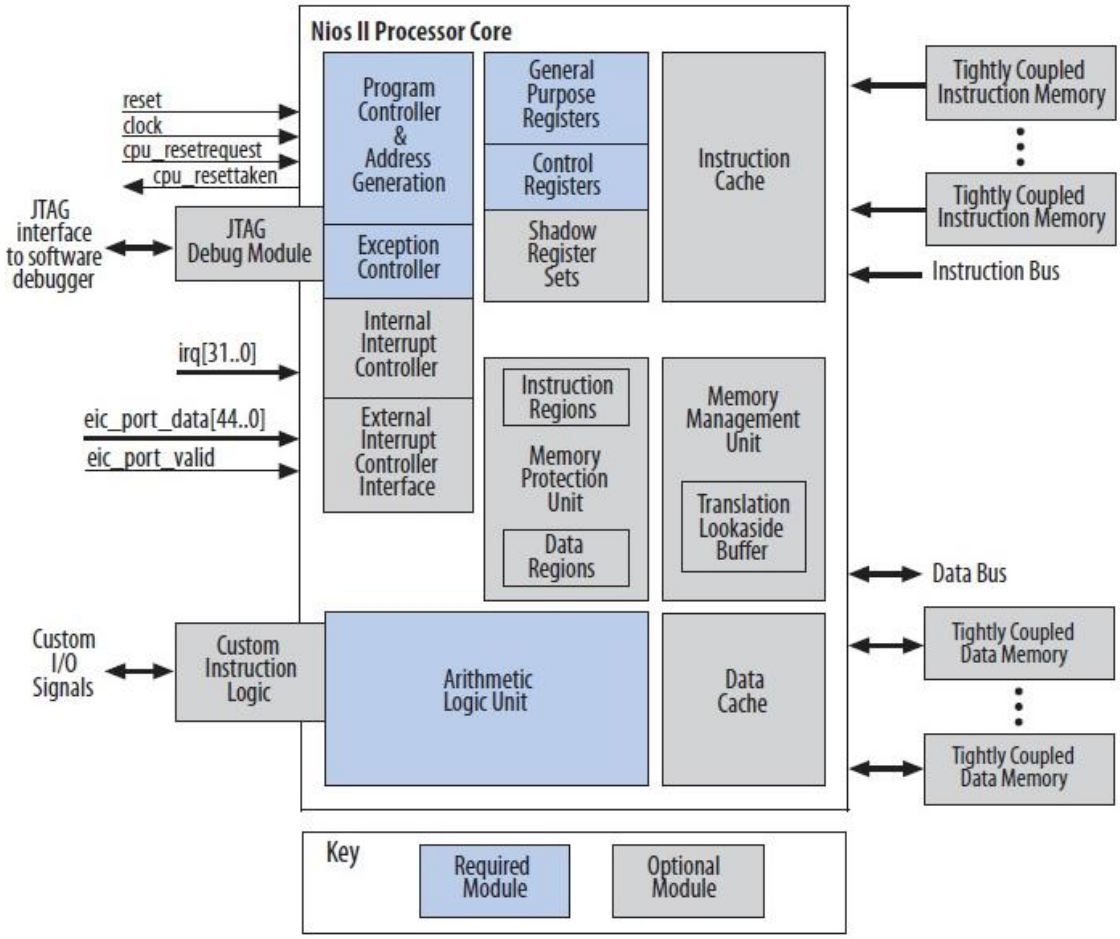
Κεφάλαιο 2 : Ο Επεξεργαστής Nios II Της Εταιρείας Altera



Ο Nios II είναι ένας επεξεργαστής των 32-bit, ιδιοκτησία της εταιρείας Altera και προορίζεται για εφαρμογή σε τσιπ Πινάκων πυλών προγραμματιζόμενων στο πεδίο (Field Programmable Gate Array - FPGA). Παρέχεται σε μορφή λογισμικού, το οποίο καθιστά εύκολη την ενσωμάτωσή του σε ένα σύστημα υπολογιστή χρησιμοποιώντας το λογισμικό Quartus II και υπολογιστικά εργαλεία σχεδίασης - CAD (Computer-Aided Design) tools - που παρέχονται από την εταιρεία Altera. Ο επεξεργαστής Nios II μπορεί να ρυθμιστεί έτσι ώστε να παρέχει έναν αριθμό από διαφορετικά χαρακτηριστικά, τα οποία απαιτούν διαφορετικές ποσότητες του λογικού κυκλώματος για την εφαρμογή τους, επηρεάζοντας έτσι την απόδοση και το κόστος του τελικού συστήματος. Δεδομένου ότι ο χρήστης μπορεί να προσαρμόσει το σχεδιασμό του τελικού κυκλώματος, μπορούμε να πούμε ότι ο Nios II είναι ένα μαλακός επεξεργαστής (soft processor).

Η ικανότητα, που παρέχει ο επεξεργαστής Nios II στον χρήστη, να σχεδιάσει ένα προσαρμοσμένο σύστημα υπολογιστή, το οποίο μπορεί να εφαρμοστεί σε ένα μόνο τσιπ FPGA, κάνει τον Nios II να είναι ίσως ο πιο διαδεδομένος soft επεξεργαστής στις ενσωματωμένες εφαρμογές.

2.1 Αρχιτεκτονική του Επεξεργαστή Nios II



Σχήμα 2.1: Block διάγραμμα του πυρήνα του επεξεργαστή Nios II [14]

Η αρχιτεκτονική του Nios II περιγράφει ένα σετ εντολών (Instruction Set Architecture - ISA). Η ISA με τη σειρά της απαιτεί ένα σύνολο λειτουργικών μονάδων που υλοποιούν τις εντολές. Στο Σχήμα 2.1 παρουσιάζεται το block διάγραμμα του πυρήνα του Nios II. Ένας πυρήνας επεξεργαστή Nios II είναι ένα σχέδιο hardware που υλοποιεί το σύνολο εντολών Nios II και υποστηρίζει τις λειτουργικές μονάδες, που αναφέρονται παρακάτω. Μία λειτουργική μονάδα μπορεί είτε να υλοποιηθεί σε hardware, είτε να εξομοιωθεί σε software ή και να παραβλεφθεί εντελώς. Ο πυρήνας του επεξεργαστή δεν περιλαμβάνει περιφερειακά ή λογική σύνδεση με τον έξω κόσμο. Περιλαμβάνει μόνο τα κυκλώματα που απαιτούνται για την υλοποίηση της αρχιτεκτονικής Nios II. Οι λειτουργικές μονάδες της αρχιτεκτονικής του Nios II είναι:

- Αρχείο καταχωρητών (register file)
- Αριθμητική και λογική μονάδα (Arithmetic Logic Unit - ALU)
- Διασύνδεση σε προσαρμοσμένη λογική εντολή (Interface to custom instruction logic)
- Δίαυλος εντολών (instruction bus)
- Δίαυλος δεδομένων (data bus)
- Ελεγκτής εξαιρέσεων (exception controller)
- Ελεγκτής εσωτερικών ή εξωτερικών διακοπών (Internal or external interrupt controller)
- Μονάδα διαχείρισης μνήμης (Memory management unit - MMU)
- Μονάδα προστασίας μνήμης (Memory protection unit - MPU)
- Κρυφές μνήμες δεδομένων και εντολών (Instruction and data cache memories)
- Στενά συνδεδεμένες διασυνδέσεις μνήμης για εντολές και δεδομένα (Tightly-coupled memory interfaces for instructions and data)
- Μονάδα αποσφαλμάτωσης JTAG (JTAG debug module)

2.2 Χαρακτηριστικά Του Επεξεργαστή Nios II

Ο Nios II είναι επεξεργαστής αρχιτεκτονικής RISC (Reduced Instruction Set Computers). Έτσι, τα χαρακτηριστικά του είναι πολύ παρόμοια με εκείνα της αρχιτεκτονικής RISC-style.

Μεγέθη Δεδομένων

Το μήκος λέξης είναι 32 bits. Τα δεδομένα ανάλογα με το μήκος τους χωρίζονται σε *λέξεις* (*words*) των 32-bit, "*μισές λέξεις*" (*halfwords*) των 16-bit ή *bytes* των 8-bit. Οι διευθύνσεις Byte σε μία λέξη αποδίδονται στην διάταξη little-endian, όπου οι χαμηλότερες διευθύνσεις byte χρησιμοποιούνται για τα λιγότερο σημαντικά bytes.

Πρόσβαση Μνήμης

Τα δεδομένα στη μνήμη είναι προσβάσιμα μόνο από το τις εντολές Load και Store, οι οποίες φορτώνουν τα δεδομένα σε καταχωρητές γενικού σκοπού ή τα αποθηκεύουν από τους καταχωρητές αυτούς. Οι εντολές Load και Store μπορούν να μεταφέρουν δεδομένα σε word, halfword, and byte μεγέθη.

Καταχωρητές

Υπάρχουν 32 καταχωρητές γενικού σκοπού και μια σειρά από καταχωρητές ελέγχου. Όλοι οι καταχωρητές έχουν μέγεθος 32 bits.

Εντολές

Όλες οι εντολές έχουν μέγεθος 32 bits. Και έχουν όλες τις λειτουργίες της αρχιτεκτονικής RISC.

Τρόποι λειτουργίας

Ο επεξεργαστής Nios II μπορεί να λειτουργήσει με δύο διαφορετικούς τρόπους:

- *Supervisor mode (λειτουργία επόπτη)***:** ο επεξεργαστής μπορεί να εκτελεί όλες τις εντολές και να παρουσιάζει όλες τις διαθέσιμες λειτουργίες. Όταν εκτελείται η λειτουργία του *reset* επανέρχεται σε αυτή τη μορφή.
- *User mode (λειτουργία χρήστη):* δεν μπορούν να εκτελεστούν κάποιες εντολές ελέγχου.

Σε μια βασική διαμόρφωση ενός επεξεργαστή Nios II, όλα τα προγράμματα τρέχουν σε Supervisor mode. Ο User mode είναι διαθέσιμος όταν ο επεξεργαστής έχει ρυθμιστεί ώστε να περιλαμβάνει τη μονάδα διαχείρισης μνήμης (MMU). Μοναδικός σκοπός του είναι να υποστηρίξει λειτουργικά συστήματα, έτσι ώστε το λογισμικό του λειτουργικού συστήματος να μπορεί να τρέξει σε Supervisor mode, ενώ τα προγράμματα εφαρμογών εκτελούνται σε User mode.

2.3 Περιφερειακά

Οι περιφερειακές συσκευές συνδέονται στους Nios II πυρήνες μέσω του Avalon Switch Fabric, που είναι ένα σύνολο από σημείο σε σημείο, master to slave συνδέσεις. Ένας master μπορεί να συνδεθεί σε πολλαπλούς slaves και ένας slave μπορεί να συνδεθεί σε πολλαπλούς master. Αν ένας slave είναι συνδεδεμένος σε πολλούς masters τότε παράγεται από τους SOPC builders ένας προγραμματισμός επιλογής, που συνήθως χρησιμοποιεί round robin αλγόριθμους επιλογής του master και επιδέχεται τον ορισμό προτεραιοτήτων.

Τα περισσότερα από αυτά τα περιφερειακά παρέχονται από την εταιρεία Altera ως IPs (Intellectual Properties) που μπορούν να προστεθούν στο σύστημα. Τα IPs είναι κώδικας σε γλώσσα περιγραφής υλικού HDL που περιγράφει τη λειτουργία τους και τους επιτρέπει να ενσωματώνονται δυναμικά κατά τον σχεδιασμό των SOPCs.

Πέρα από τα IPs της Altera δίνεται η δυνατότητα στο χρήστη να ενσωματώσει κάποιο δικό του περιφερειακό ή να χρησιμοποιήσει IPs τρίτων που εκτελούν συγκεκριμένες λειτουργίες. Έτσι έχουμε τα custom (προσαρμοσμένα) περιφερειακά, τα οποία επιτρέπουν τη μεταφορά συναρτήσεων, που επαναλαμβάνεται, μεταφράζοντας την ίδια λειτουργία σε υλικό, αυξάνοντας έτσι την συνολική απόδοση του συστήματος. Η αύξηση της απόδοσης προέρχεται τόσο από το γεγονός πως η υλοποίηση σε hardware είναι γρηγορότερη από την αντίστοιχη σε λογισμικό αλλά και ταυτόχρονα απελευθερώνεται ο επεξεργαστής ώστε να εκτελέσει άλλες διεργασίες παράλληλα με αυτή που διεκπεραιώνεται στο περιφερειακό.

2.4 Καταχωρητές του Επεξεργαστή Nios II

2.4.1 Καταχωρητές Γενικού Σκοπού

Καταχωρητές	Όνομα	Λειτουργία
r0	zero	0x00000000
r1	at	Assembler Temporary
r2		
r3		
-	-	-
-	-	-
-	-	-
r23		
r24	et	Exception Temporary
r25	bt	Breakpoint Temporary
r26	gp	Global Pointer
r27	sp	Stack Pointer
r28	fp	Frame Pointer
r29	ea	Exception Return Address
r30	ba	Breakpoint Return Address
r31	ra	Return Address

Πίνακας 2.1: Καταχωρητές γενικού σκοπού του επεξεργαστή Nios II [11]

Οι καταχωρητές γενικού σκοπού είναι 32 και ονομάζονται από το r0 έως r32, τα οποία είναι τα ονόματα που χρησιμοποιούνται στις εντολές της γλώσσας assembly. Μερικοί καταχωρητές χρησιμοποιούνται για ειδικούς σκοπούς και συνεπώς δίνονται επίσης ονόματα τα οποία είναι περισσότερο ενδεικτικά της λειτουργικότητάς τους. Ο Πίνακας 2.1 εμφανίζει τους 32 καταχωρητές γενικού σκοπού του επεξεργαστή.

Αναλυτικά, οι καταχωρητές που προορίζονται για ένα συγκεκριμένο σκοπό, που φαίνονται και στον παραπάνω πίνακα, είναι οι εξής:

- **r0:** περιέχει πάντα την σταθερά 0. Διαβάζοντας αυτόν τον καταχωρητή επιστρέφει η τιμή 0. Γράφοντας σε αυτόν δεν έχει καμία επίδραση.
- **r1 (Assembler Temporary):** χρησιμοποιείται από τον assembler ως προσωρινός καταχωρητής. Δεν θα πρέπει να χρησιμοποιείται σε προγράμματα χρηστών.
- **r24 (Exception Temporary) και r29 (Exception Return Address):** χρησιμοποιούνται για την επεξεργασία των εξαιρέσεων.
- **r25 (Breakpoint Temporary) και r30 (Breakpoint Return Address):** χρησιμοποιούνται αποκλειστικά από ένα εργαλείο εντοπισμού σφαλμάτων, που ονομάζεται JTAG Debug Module.
- **r26 (Global Pointer):** είναι παγκόσμιος δείκτης για τα δεδομένα σε ένα πρόγραμμα χρήστη.
- **r27 (Stack Pointer):** είναι ο δείκτης στοίβας του επεξεργαστή.
- **r28 (Frame Pointer):** είναι ο δείκτης πλαισίου
- **r31 (Return Address):** κρατά την επιστρεφόμενη διεύθυνση όταν μια υπορουτίνα καλείται.

Οι άλλοι καταχωρητές χρησιμοποιούνται για γενικό σκοπό. Επειδή ο καταχωρητής r0 περιέχει πάντα το 0, μπορεί να συμπεριληφθεί ως τελεστής σε μία εντολή κάθε φορά που χρειάζεται την τιμή 0. Αυτό μπορεί να συμβεί με διάφορους τρόπους. Για παράδειγμα, η εντολή " add r5, r0, r0" μπορεί να χρησιμοποιηθεί για να καθαρίσει τον καταχωρητή r5.

2.4.2 Καταχωρητές Ελέγχου

Στη βασική διαμόρφωση ενός επεξεργαστή Nios II, υπάρχουν έξι καταχωρητές ελέγχου. Επιπλέον καταχωρητές ελέγχου παρέχονται όταν προηγμένες μονάδες hardware υλοποιούνται, όπως η μονάδα διαχείρισης μνήμης (MMU) ή ο ελεγκτής εξωτερικής διακοπής.

Οι βασικοί καταχωρητές ελέγχου αναφέρονται στον Πίνακα 2.2. Ονομάζονται *ctl0* έως *ctl5*. Οι καταχωρητές ελέγχου διαβάζονται και γράφονται από τις ειδικές εντολές *rdctl* και *wrctl* αντίστοιχα.

Register	Name	$b_{31} \ldots b_2$	b_1	b_0
ctl0	status	Reserved	U	PIE
ctl1	estatus	Reserved	EU	EPIE
ctl2	bstatus	Reserved	BU	BPIE
ctl3	ienable	Interrupt-enable bits		
ctl4	ipending	Pending-interrupt bits		
ctl5	cpuid	Processor identifier		

Πίνακας 2.2: Βασικοί καταχωρητές ελέγχου [11]

Οι καταχωρητές ελέγχου του επεξεργαστή Nios II χρησιμοποιούνται ως εξής:

- Ο καταχωρητής *ctl0* είναι ο καταχωρητής κατάστασης (*Status register*) που υποδεικνύει την τρέχουσα κατάσταση του επεξεργαστή. Στη βασική διαμόρφωση, μόνο δύο bits χρησιμοποιούνται:
  - Το bit *PIE* είναι το bit ενεργοποίησης διακοπών του επεξεργαστή. Ο επεξεργαστής θα δεχτεί αίτηση διακοπής από συσκευές I/O όταν *PIE* = 1, και θα τις αγνοήσει όταν *PIE* = 0.
  - Το *U* είναι το bit των λειτουργιών User/Supervisor mode. Είναι 0 για την Supervisor mode και 1 για την User mode.
- Ο Καταχωρητής *ctl1* χρησιμοποιείται για να αποθηκεύει αυτόματα τα περιεχόμενα του status register όταν μία ρουτίνα διακοπής - ή εξαίρεσης - υπηρεσίας εκτελείται. Τα bits *EU* και *EPIE* είναι τα αποθηκευμένα status bits *U* και *PIE.*
- Ο καταχωρητής *ctl2* χρησιμοποιείται για να αποθηκεύει τα περιεχόμενα του status register κατά τη διάρκεια της επεξεργασίας διακοπής εντοπισμού σφαλμάτων. Τα bits *BU* και *BPIE* είναι τα αποθηκευμένα status bits *U* και *PIE*.
- Ο καταχωρητής *ctl3* χρησιμοποιείται για την ενεργοποίηση μεμονωμένων διακοπών από συσκευές I/O. Κάθε bit αντιστοιχεί σε μία από τις διακοπές *irq0* έως *irq31*. Οι τιμές του bit 1 και 0 ενεργοποιούνται και απενεργοποιούνται σε κάθε διακοπή, αντίστοιχα.

- Ο καταχωρητής *ctl4* υποδεικνύει ποιες αιτήσεις διακοπών εκκρεμούν. Η τιμή ενός δεδομένου bit, $ctl4_k$, έχει οριστεί σε 1, αν η διακοπή *irqk* είναι ενεργή και επίσης ενεργοποιημένη από το bit ενεργοποίησης διακοπών $ctl3_k$ γίνεται ίσο με 1.
- Ο καταχωρητής *ctl5* χρησιμοποιείται για να κρατήσει μια τιμή που προσδιορίζει μοναδικά τον επεξεργαστή όταν είναι μέρος ενός συστήματος πολλαπλών επεξεργαστών.

2.5 Τρόποι Διευθυνσιοδότησης

Ο επεξεργαστής Nios II υποστηρίζει πέντε τρόπους διευθυνσιοδότησης:

- *Immediate mode* (άμεσος τρόπος) - Ένας 16-bit τελεστής δίνεται στην εντολή. Αυτή η τιμή έχει επεκταθεί για να παράγει έναν 32-bit τελεστή για εντολές που εκτελούν αριθμητικές πράξεις.
- *Register mode* (τρόπος καταχώρησης) - Ο τελεστής είναι τα περιεχόμενα του καταχωρητή γενικού σκοπού.
- *Register indirect mode* (τρόπος έμμεσης καταχώρησης) - Η αποτελεσματική διεύθυνση (effective address) του τελεστή είναι τα περιεχόμενα του καταχωρητή.
- *Displacement mode* (τρόπος μετατόπισης) - Η αποτελεσματική διεύθυνση του τελεστή δημιουργείται με την πρόσθεση των περιεχομένων του καταχωρητή και μία προσημασμένη 16-bit τιμή μετατόπισης δίνεται στην εντολή.
- *Absolute mode* (απόλυτος τρόπος) - Μία 16-bit απόλυτη διεύθυνση του τελεστή μπορεί να καθοριστεί χρησιμοποιώντας τον τρόπο μετατόπισης με τον καταχωρητή r0.

Οι τρόποι διευθυνσιοδότησης και η σύνταξη του assembler τους δίνονται στον Πίνακα 2.3.

Name	Assembler syntax	Addressing function
Immediate	*Value*	*Operand = Value*
Register	*ri*	*EA = ri*
Register indirect	*(ri)*	*EA = [ri]*
Displacement	*X(ri)*	*EA = [ri] + X*
Absolute	*LOC(r0)*	*EA = LOC*

EA = effective address
Value = a 16-bit signed number
X = a 16-bit signed displacement value

Πίνακας 2.3: Τρόποι διευθυνσιοδότησης του επεξεργαστή Nios II [11]

Παρατηρούμε ότι και οι δυο τρόποι διευθυνσιοδότησης, Άμεσος και Απόλυτος, μπορούν να χρησιμοποιηθούν μόνο εάν η άμεση τιμή (*Value*) ή η διεύθυνση μπορεί να παρασταθεί στα 16 bits.

2.6 Εντολές

Όλες οι εντολές του επεξεργαστή Nios II είναι μήκους 32-bit. Ο συμβολισμός που χρησιμοποιείται σε προγράμματα της γλώσσας assembly διέπεται από τους περιορισμούς. Πολλοί assemblers δέχονται ότι στις δηλώσεις ενός προγράμματος κώδικα υπάρχει διάκριση πεζών-κεφαλαίων. Επομένως, οι δηλώσεις "ADD R2, R3, R4" και "add r2, r3, r4" είναι ισοδύναμες. Ο assembler του Nios II, που παρέχεται από την Altera,επιτρέπει την διάκριση πεζών-κεφαλαίων, αλλά απαιτεί πεζά γράμματα για τα ονόματα των καταχωρητών. Έτσι οι καταχωρητές πρέπει να προσδιορίζονται από τα ονόματα που δίνονται στους Πίνακες 2.1, 2.2. Στην τεκμηρίωση της Altera, τα πεζά γράμματα χρησιμοποιούνται για τον προσδιορισμό του OP-code.

Το σετ εντολών του επεξεργαστή Nios II είναι αρκετά εκτεταμένο. Για αυτό το λόγο, θα παρουσιαστεί ένα μόνο υποσύνολο που είναι επαρκής για την κατανόηση των δυνατοτήτων του επεξεργαστή Nios II.

Υπάρχουν τρεις τύποι εντολών του επεξεργαστή Nios II: είναι οι I-Type, R-Type, και J-Type εντολές.

I-Type Εντολές

Πεδία των bits			
31 – 27	26 – 22	21 – 6	5 – 0
A	B	IMM16	OP

Πίνακας 2.4: Μορφή εντολών **I-Type** του Nios II [14]

Το καθοριστικό χαρακτηριστικό των I-Type εντολών είναι ότι περιέχουν μία άμεση τιμή εντός του Instruction word τους. Οι εντολές I-Type περιλαμβάνουν αριθμητικές και λογικές πράξεις, branch λειτουργίες, load και store λειτουργίες και λειτουργίες διαχείρισης κρυφής μνήμης. Περιέχουν [Πίνακας 2.4]:

- Ένα πεδίο opcode (*OP*) των 6-bit
- Δύο πεδία καταχωρητών *A* και *B* των 5-bit
- Ένα πεδίο άμεσων δεδομένων - immediate data (*IMM16*) των 16-bit

Στις περισσότερες περιπτώσεις, τα πεδία A και IMM16 καθορίζουν τους τελεστές της πηγής, και το πεδίο B καθορίζει τον καταχωρητή προορισμού (destination register). Το IMM16 θεωρείται προσημασμένο εκτός από τις λογικές πράξεις και τις χωρίς-πρόσημο συγκρίσεις.

R-Type Εντολές

Πεδία των bits				
31 – 27	26 – 22	21 – 17	16 – 6	5 – 0
A	B	C	OPX	OP

Πίνακας 2.5: Μορφή εντολών R-Type του Nios II [14]

Το καθοριστικό χαρακτηριστικό των εντολών R-Type είναι ότι όλα τα αποτελέσματα και τα επιχειρήματα καθορίζονται ως καταχωρητές. Οι εντολές R-Type περιλαμβάνουν αριθμητικές και λογικές πράξεις, συγκρίσεις, προσαρμοσμένες (custom) εντολές και άλλες λειτουργίες που χρειάζονται μόνο τελεστές καταχωρητών. Περιέχουν [Πίνακας 2.5]:

- Ένα πεδίο opcode (*OP*) των 6-bit
- Τρία πεδία καταχωρητών *A, B* και *C* των 5-bit
- Ένα πεδίο opcode-extension (*OPX*) των 11-bit

Στις περισσότερες περιπτώσεις, τα πεδία A και B προσδιορίζουν τους τελεστές πηγής, και το πεδίο C καθορίζει τον καταχωρητή προορισμού (destination register). Μερικές εντολές R-Type ενσωματώνουν μια μικρή άμεση τιμή στα πέντε bits χαμηλής τάξης των OPX. Τα αχρησιμοποίητα bits στο OPX είναι πάντα μηδέν.

J-Type Εντολές

Πεδία των bits																														
31																									6	5					0
IMM26																										OP				

Πίνακας 2.6: Μορφή εντολών J-Type του Nios II [14]

Οι εντολές J-Type περιέχουν [Πίνακας 2.6]:

- Ένα πεδίο opcode (OP) των 6-bit
- Ένα πεδίο άμεσων δεδομένων (immediate data) των 26-bit

Οι εντολές J-Type, όπως οι call και jmpi, μεταφέρουν την εκτέλεση οπουδήποτε μέσα σε ένα εύρος των 256MB.

2.6.1 Εντολές Load και Store

Οι εντολές Load (φόρτωση) και Store (αποθήκευση) μετακινούν τα δεδομένα ανάμεσα στη μνήμη, ή της διασυνδέσεις I/O, και τους καταχωρητές γενικού σκοπού. Ανήκουν στις εντολές I-Type. Η εντολή Load Word έχει την γενική μορφή

ldw rB, byte_μετατόπισης (rA)

η οποία φορτώνει την 32-bit λέξη από την μνήμη ή από περιφερειακό I/O. Αναλυτικότερα, η διεύθυνση μνήμης είναι το άθροισμα του byte_μετατόπισης και των περιεχομένων του καταχωρητή rA. Η 16-bit byte_μετατόπιση μπορεί με προσημασμένη τιμή να επεκταθεί στα 32 bits. Με την εντολή αυτή ο τελεστής μνήμης των 32 bits φορτώνεται στον καταχωρητή rB. Η αποτελεσματική διεύθυνση πρέπει να είναι word-aligned, δηλαδή εάν η διεύθυνση δεν είναι πολλαπλάσια του 4, η πράξη είναι απροσδιόριστη.

Η εντολή Store Word έχει τη μορφή

stw rB, byte_offset(rA)

η οποία αποθηκεύει τα περιεχόμενα του καταχωρητή rB στην διεύθυνση μνήμης που είναι το άθροισμα της τιμής της byte_μετατόπισης και των περιεχομένων του καταχωρητή rA. Η αποτελεσματική διεύθυνση πρέπει να είναι word-aligned, δηλαδή εάν η διεύθυνση δεν είναι πολλαπλάσια του 4, η πράξη είναι απροσδιόριστη.

Εκτός από τους τελεστές μεγέθους λέξης (word - 32bits), οι εντολές Load και Store μπορούν επίσης να χειριστούν τελεστές μεγέθους byte (8 bit) και halfword (16-bit). Τέτοιες εντολές είναι οι εξής:

- **ldb** (Load Byte - φόρτωση byte)
- **ldbu** (Load Byte Unsigned - φόρτωση μη προσημασμένου byte)
- **ldh** (Load Halfword - φόρτωση μισής λέξης)
- **ldhu** (Load Halfword Unsigned - φόρτωση μη προσημασμένης μισής λέξης)

Όταν ένας μικρότερος τελεστής φορτώνεται σε έναν 32-bit καταχωρητή, η τιμή του πρέπει να προσαρμοστεί για να ταιριάξει στον καταχωρητή.

Οι αντίστοιχες εντολές Store είναι:

- **stb** (Store Byte - αποθήκευση Byte): αποθηκεύει το byte χαμηλής τάξης του καταχωρητή πηγής στο byte που ορίζεται από την effective address.
- **sth** (Store halfword - αποθήκευση μισής λέξης): αποθηκεύει την halfword χαμηλής τάξης του καταχωρητή της πηγής στη halfword μνήμης που ορίζεται από την effective address.

Για κάθε εντολή Load και Store υπάρχει επίσης μία εκδοχή που προορίζεται για την πρόσβαση σε διασυνδέσεις I/O. Τέτοιες εντολές είναι οι εξής:

- **ldwio** (Load Word I/O - φόρτωση λέξης εισόδου/εξόδου)
- **ldbio** (Load Byte I/O - φόρτωση byte I/O)
- **ldbuio** (Load Byte Unsigned I/O - φόρτωση μη προσημασμένου byte I/O)
- **ldhio** (Load Halfword I/O - φόρτωση μισής λέξης I/O)
- **ldhuio** (Load Halfword Unsigned I/O - φόρτωση μη προσημασμένης μισής λέξης I/O)
- **stwio** (Store Word I/O - αποθήκευση λέξης I/O)
- **stbio** (Store Byte I/O - αποθήκευση byte I/O)
- **sthio** (Store Halfword I/O - αποθήκευση μισής λέξης I/O)

Οι εκδοχές των εντολών Load και Store για συσκευές I/O απαιτούνται όταν ο επεξεργαστής Nios II χρησιμοποιείται με κρυφή μνήμη. Η κρυφή μνήμη είναι μια σχετικά μικρή μνήμη που μπορεί να προσπελαστεί γρηγορότερα από την κύρια μνήμη. Φορτώνεται με πρόσφατα χρησιμοποιημένες εντολές και δεδομένα από την κύρια μνήμη, έτσι ώστε να μπορούν να έχουν πρόσβαση πιο γρήγορα όταν τα χρειαστούν πάλι. Αυτό είναι πλεονέκτημα για τις εντολές και τα δεδομένα που βρίσκονται κανονικά στην κύρια μνήμη. Οι εκδοχές των εντολών Load και Store για συσκευές I/O παρακάμπτουν την προσωρινή μνήμη, αν υπάρχει, και πάντα εισχωρούν στη θέση I / O.

2.6.2 Αριθμητικές Εντολές

Οι αριθμητικές εντολές χειρίζονται δεδομένα που είτε είναι καταχωρητές γενικής χρήσης ή δίνεται ως μια άμεση τιμή στην εντολή. Ανήκουν στις μορφές εντολών R-Type και I-Type. Αυτές οι εντολές είναι οι εξής:

- **add:** Add Registers **-** πρόσθεση καταχωρητών
- **addi**: Add Immediate - πρόσθεση άμεσης διεύθυνσης
- **sub**: Subtract Registers - αφαίρεση καταχωρητών
- **subi:** Subtract Immediate - αφαίρεση άμεσης διεύθυνσης
- **mul:** Multiply - πολλαπλασιασμός
- **muli:** Multiply Immediate - πολλαπλασιασμός άμεσης διεύθυνσης
- **div:** Divide - διαίρεση
- **divu:** Divide Unsigned - μη προσημασμένη διαίρεση

add **r*i*, r*j*, r*k*:** η εντολή Add προσθέτει τα περιεχόμενα των καταχωρητών r*j* και r*k,* και τοποθετεί το άθροισμά τους στον καταχωρητή r*i*.

addi **ri, r*j*, 85:** η άμεση εντολή addi προσθέτει τα περιεχόμενα του καταχωρητή r*j* και της άμεσης τιμής 85 (16-bit προσημασμένος τελεστής), και τοποθετεί το αποτέλεσμα στον καταχωρητή r*i*.

sub **r*i*, r*j*, r*k*:** η εντολή της αφαίρεσης αφαιρεί τα περιεχόμενα του καταχωρητή r*k* από τον καταχωρητή r*j*, και τοποθετεί το αποτέλεσμα στον καταχωρητή r*i*.

mul **r*i*, r*j*, r*k*:** η εντολή του πολλαπλασιασμού πολλαπλασιάζει τα περιεχόμενα των καταχωρητών r*j* και r*k*, και τοποθετεί το χαμηλής τάξης 32bit αποτέλεσμα στον καταχωρητή r*i*. Ο πολλαπλασιασμός αντιμετωπίζει τους τελεστές ως μη προσημασμένους αριθμούς. Το αποτέλεσμα είναι σωστό εάν το παραγόμενο προϊόν μπορεί να εκπροσωπείται στα 32 bits, ανεξάρτητα από το αν οι τελεστές είναι προσημασμένοι ή μη.

muli **r*i*, r*j*, Value16**: Στην άμεση εκδοχή του πολλαπλασιασμού, ο 16-bit άμεσος τελεστής Value16 είναι προσημασμένος και επεκτείνεται μέχρι τα 32 bits.

div **r*i*, r*j*, r*k*:** η εντολή της διαίρεσης διαιρεί τα περιεχόμενα του καταχωρητή rj από τα περιεχόμενα του καταχωρητή rk και τοποθετεί το ακέραιο μέρος του πηλίκου στον καταχωρητή ri. Οι τελεστές αντιμετωπίζονται ως προσημασμένοι ακέραιοι.

Η εντολή **divu** εκτελείται με τον ίδιο τρόπο εκτός του ότι οι τελεστές αντιμετωπίζονται ως μη προσημασμένοι ακέραιοι.

2.6.3 Λογικές Εντολές

Οι λογικές εντολές παρέχουν τις πράξεις AND, OR, XOR, και NOR. Διαχειρίζονται δεδομένα που είναι είτε σε καταχωρητές γενικής χρήσης ή ως άμεση τιμή στην εντολή. Ανήκουν στις μορφές εντολών R-Type και I-Type.

and **r*i*, r*j*, r*k*:** εκτελεί την λογική πράξη AND στα περιεχόμενα των καταχωρητών rj και rk, και αποθηκεύει το αποτέλεσμα στον καταχωρητή ri. Ομοίως, οι εντολές or, xor, και nor εκτελούν τις λογικές πράξεις OR, XOR, και NOR, αντίστοιχα.

andi **r*i*, r*j*, Value16**: η άμεση εκδοχή της εντολής and εκτελεί τη λογική πράξη AND στα περιεχόμενα του καταχωρητή rj και του 16-bit άμεσου τελεστή Value16, ο οποίος επεκτείνεται έως τα 32 bits, και αποθηκεύει το αποτέλεσμα στον καταχωρητή ri. Ομοίως, οι εντολές ori, xori, και nori εκτελεί τις λογικές πράξεις OR, XOR, και NOR, αντίστοιχα, χρησιμοποιώντας τους άμεσους τελεστές.

Είναι επίσης δυνατόν να χρησιμοποιηθεί ένας άμεσος τελεστής των 16-bit, ως τα 16 bits υψηλής τάξης στις λογικές πράξεις, στην οποία περίπτωση τα χαμηλής τάξης 16 bits του τελεστή είναι μηδενικά. Αυτό επιτυγχάνεται με τις εξής οδηγίες:

- **andhi** (AND High Immediate)
- **orhi** (OR High Immediate)
- **xorhi** (XOR High Immediate)

Αυτό παρέχει ένα μηχανισμό για τη φόρτωση μιας άμεσης αξίας 32-bit σε ένα καταχωρητή, τοποθετώντας πρώτα τα 16 bits υψηλής τάξης στον καταχωρητή

χρησιμοποιώντας την εντολή orhi και, στη συνέχεια μέσου της λογικής πράξης OR στα χαμηλής τάξης 16 bits χρησιμοποιώντας την εντολή ori.

2.6.4 Εντολές Μεταφοράς (Move)

Οι εντολές μεταφοράς αντιγράφουν τα περιεχόμενα ενός καταχωρητή σε έναν άλλο, ή τοποθετούν μία άμεση τιμή σε έναν καταχωρητή. Αυτές είναι ψευδοεντολές που παρέχονται για την διευκόλυνση του προγραμματιστή.

mov r*i*, r*j*: αντιγράφει τα περιεχόμενα του καταχωρητή r*j* στον καταχωρητή r*i*. Υλοποιείται όπως η εντολή add r*i*, r*j*, r0.

movi r*i*, Value16: η άμεση εντολή μεταφοράς επεκτείνει την προσημασμένη 16-bit άμεση τιμή (Value16) στα 32 bits και την φορτώνει στον καταχωρητή r*i*. Υλοποιείται όπως η εντολή addi r*i*, r0, Value16.

movui r*i*, Value16: η εντολή μεταφοράς για μη προσημασμένη άμεση τιμή επεκτείνει την 16-bit άμεση, μη προσημασμένη τιμή (Value16) στα 32 bits και την φορτώνει στον καταχωρητή r*i*. Υλοποιείται όπως η εντολή ori r*i*, r0, Value16.

movia r*i*, LABEL: η εντολή μεταφοράς άμεσης διεύθυνσης φορτώνει μία 32-bit τιμή, που αντιστοιχεί στην ετικέτα (LABEL) της διεύθυνσης, στον καταχωρητή r*i*. Υλοποιείται χρησιμοποιώντας δύο εντολές

```
orhi    ri, r0, LABEL_HIGH
ori     ri, ri, LABEL_LOW
```

όπου LABEL_HIGH και LABEL_LOW είναι τα 16 bits υψηλής και χαμηλής τάξης της ετικέτας, αντίστοιχα.

2.6.5 Εντολές Branch και Jump

Η ροή εκτέλεσης ενός προγράμματος αλλάζει χρησιμοποιώντας τις εντολές Branch και Jump. Η άνευ όρων εντολή Branch

br LABEL

μεταφέρει την εκτέλεση άνευ όρων στην εντολή της ετικέτας της διεύθυνσης. Ο στόχος της branch καθορίζεται στη μορφή της προσημασμένης 16-bit μετατόπισης (offset), που περιλαμβάνεται στην εντολή. Η μετατόπιση (offset) είναι η απόσταση σε bytes από την εντολή που ακολουθεί αμέσως το br στην ετικέτα της διεύθυνσης.

Η υπό όρους μεταφορά της εκτέλεσης επιτυγχάνεται με υπό όρους εντολές διακλάδωσης (Branch), οι οποίες συγκρίνουν τα περιεχόμενα των δύο καταχωρητών γενικού σκοπού και προκαλεί μία διακλάδωση εάν το αποτέλεσμα ικανοποιεί τον όρο της εντολής branch.

blt *ri*, *rj*, LABEL: η εντολή Branch if Less Than για προσημασμένους αριθμούς εκτελεί την σύγκριση [r*i*] < [r*j*], αντιμετωπίζοντας τα περιεχόμενα των καταχωρητών ως προσημασμένους αριθμούς.

bltu *ri*, *rj*, LABEL: η εντολή Branch if Less Than για μη προσημασμένους αριθμούς εκτελεί τη σύγκριση [r*i*] < [r*j*], αντιμετωπίζοντας τα περιεχόμενα των καταχωρητών ως μη προσημασμένους αριθμούς.

Οι άλλες υπό όρους εντολές Branch της ίδιας μορφής είναι οι εξής:

- **beq** (Σύγκριση [r*i*] = [r*j*])
- **bne** (Σύγκριση [r*i*] ≠ [r*j*])
- **bge** (Προσημασμένη σύγκριση [r*i*] ≥ [r*j*])
- **bgeu** (Μη προσημασμένη σύγκριση [r*i*] ≥ [r*j*])
- **bgt** (Προσημασμένη σύγκριση [r*i*] > [r*j*])
- **bgtu** (Μη προσημασμένη σύγκριση [r*i*] > [r*j*])
- **ble** (Προσημασμένη σύγκριση [r*i*] ≤ [r*j*])
- **bleu** (Μη προσημασμένη σύγκριση [r*i*] ≤ [r*j*])

Ο στόχος της κάθε εντολής διακλάδωσης πρέπει να είναι εντός του εύρους που μπορεί να καθοριστεί σε ένα offset 16-bit. Για έναν στόχο εκτός αυτού του εύρους, είναι απαραίτητο να χρησιμοποιηθεί η εντολή Jump

JMP r*i*

η οποία μεταφέρει την εκτέλεση άνευ όρων στη διεύθυνση που περιέχεται στον καταχωρητή r*i*.

2.6.6 Εντολές Κλήσης Υπορουτίνας

Ο επεξεργαστής Nios II έχει δύο εντολές για την κλήση υπορουτινών. Η εντολή κλήσης υπορουτίνας

call LABEL

περιλαμβάνει μία 26-bit μη προσημασμένη άμεση τιμή. Η εντολή αποθηκεύει την διεύθυνση επιστροφής (η οποία είναι η διεύθυνση της επόμενης εντολής) στον καταχωρητή r31. Στη συνέχεια, μεταφέρει τον έλεγχο στην εντολή της διεύθυνσης του LABEL. Αυτή η διεύθυνση καθορίζεται από την αλληλουχία των τεσσάρων bits υψηλής τάξης του μετρητή προγράμματος με την άμεση τιμή, Value26, και των δύο bits χαμηλής τάξης ως εξής: **Jump Address = PC_{31-28} : Value26 : 00**
Τα δύο λιγότερο σημαντικά bits είναι 0, διότι οι εντολές του Nios II πρέπει να ευθυγραμμιστεί με τα όρια των λέξεων.

Η εντολή κλήσης υπορουτίνας σε καταχωρητή

callr r*i*

αποθηκεύει τη διεύθυνση επιστροφής στον καταχωρητή r31 και στη συνέχεια μεταφέρει τον έλεγχο στην εντολή που βρίσκεται στη διεύθυνση που περιέχεται στον καταχωρητή r*i*.

Επιστροφή από υπορουτίνα εκτελείται με την εντολή **ret.** Αυτή η εντολή μεταφέρει την εκτέλεση στη διεύθυνση που περιέχονται στον καταχωρητή r31.

2.6.7 Εντολές Σύγκρισης

Οι εντολές σύγκρισης (Comparison) συγκρίνουν τα περιεχόμενα δύο καταχωρητών ή τα περιεχόμενα ενός καταχωρητή και μίας άμεσης τιμής και γράφουν στον καταχωρητή το αποτέλεσμα, είτε 1 (εάν η συνθήκη είναι αληθής) ή 0 (εάν η συνθήκη είναι ψευδής).

cmplt *ri, rj, rk*: η εντολή Compare Less Than για προσημασμένους αριθμούς εκτελεί τη σύγκριση προσημασμένων αριθμών στους καταχωρητές r*j* και r*k*, [r*j*]<[r*k*], και γράφει 1 στον καταχωρητή r*i* εάν το αποτέλεσμα είναι αληθές. Διαφορετικά, γράφει ένα 0.

cmpltu *ri, rj, rk*: η εντολή Compare Less Than για μη προσημασμένους αριθμούς εκτελεί την ίδια λειτουργία με την εντολή cmplt, αλλά αντιμετωπίζει τους τελεστές ως μη προσημασμένους αριθμούς.

Άλλες εντολές αυτού του τύπου είναι οι εξής:

- **cmpeq** (Σύγκριση [r*i*] = [r*j*])
- **cmpne** (Σύγκριση [r*i*] ≠ [r*j*])
- **cmpge** (Προσημασμένη σύγκριση [r*i*] ≥ [r*j*])
- **cmpgeu** (Μη προσημασμένη σύγκριση [r*i*] ≥ [r*j*])
- **cmpgt** (Προσημασμένη σύγκριση [r*i*] > [r*j*])
- **cmpgtu** (Μη προσημασμένη σύγκριση [r*i*] > [r*j*])
- **cmple** (Προσημασμένη σύγκριση [r*i*] ≤ [r*j*])
- **cmpleu** (Μη προσημασμένη σύγκριση [r*i*] ≤ [r*j*])

Οι άμεσες εκδοχές των εντολών σύγκρισης περιλαμβάνουν έναν 16-bit άμεσο τελεστή.

cmplti *ri, rj,* Value16: η εντολή Compare Less Than Signed Immediate (για άμεσες προσημασμένες τιμές) συγκρίνει τον προσημασμένο αριθμό στον καταχωρητή r*j* με τον προσημασμένο άμεσο τελεστή. Γράφει το 1 στον καταχωρητή r*i* εάν [r*j*]<Value16. Αλλιώς, γράφει το 0.

cmpltui ***ri, rj,* Value16:** η εντολή Compare Less Than Unsigned Immediate (για άμεσες μη προσημασμένες τιμές) συγκρίνει τον μη προσημασμένο αριθμό στον καταχωρητή r*j* με τον άμεσο τελεστή. Γράφει το 1 στον καταχωρητή r*i* εάν [r*j*]<Value16. Αλλιώς, γράφει το 0.

Άλλες άμεσες εντολές αυτού του τύπου είναι οι εξής:

- **cmpeqi** (Σύγκριση [r*i*] = Value16)
- **cmpnei** (Σύγκριση [r*i*] ≠ Value16)
- **cmpgei** (Προσημασμένη σύγκριση [r*i*] ≥ Value16)
- **cmpgeui** (Μη προσημασμένη σύγκριση [r*i*] ≥ Value16)
- **cmpgti** (Προσημασμένη σύγκριση [r*i*] > Value16)
- **cmpgtui** (Μη προσημασμένη σύγκριση [r*i*] > Value16)
- **cmplei** (Προσημασμένη σύγκριση [r*i*] ≤ Value16)
- **cmpleui** (Μη προσημασμένη σύγκριση [r*i*] ≤ Value16)

2.6.8 Εντολές Shift

Οι εντολές ολίσθησης (Shift) μετακινούν τα περιεχόμενα ενός συγκεκριμένου καταχωρητή είτε προς τα δεξιά ή προς τα αριστερά.

srl ***ri, rj, rk*:** η εντολή Shift Right Logical (λογική δεξιά ολίσθηση) μετακινεί το περιεχόμενο του καταχωρητή r*j* προς τα δεξιά τόσες φορές, όσες ο αριθμός των θέσεων των bits που ορίζονται από τα πέντε λιγότερο σημαντικά bits (εύρος αριθμού 0 έως 31) του καταχωρητή r*k*, και αποθηκεύει το αποτέλεσμα στον καταχωρητή r*i*. Τα κενά bits στην αριστερή πλευρά του μετατοπισμένου τελεστή γεμίζουν με μηδενικά.

srli ***ri, rj,* Value5:** η άμεση εντολή Shift Right Logical Immediate (άμεση λογική δεξιά ολίσθηση) μεταθέτει το περιεχόμενο του καταχωρητή r*j* προς τα δεξιά από τον αριθμό των θέσεων των bits που ορίζεται από την 5-bit μη προσημασμένη τιμή, Value5, δίνεται στην εντολή, και αποθηκεύει το αποτέλεσμα στον καταχωρητή r*i*. Τα κενά bits στην αριστερή πλευρά του μετατοπισμένου τελεστή γεμίζουν με μηδενικά.

Οι άλλες εντολές Shift είναι οι:

- **sra** (Shift Right Arithmetic - Δεξιά αριθμητική ολίσθηση)
- **srai** (Shift Right Arithmetic Immediate - Δεξιά άμεση αριθμητική ολίσθηση)
- **sll** (Shift Left Logical - Αριστερή λογική ολίσθηση)
- **slli** (Shift Left Logical Immediate - Αριστερή άμεση λογική ολίσθηση)

Οι εντολές **sra** και **srai** λειτουργούν όπως οι εντολές srl και srli, εκτός του ότι το προσημασμένο bit, rj_{31}, επαναλαμβάνεται στα κενά bits στην αριστερή πλευρά του μετατοπισμένου τελεστή.

Οι εντολές **sll** και **slli** λειτουργούν όπως οι εντολές srl και srli, αλλά μετατοπίζουν τον τελεστή στον καταχωρητή r*j* προς τα αριστερά και γεμίζουν τα κενά bits στην δεξιά πλευρά με μηδενικά.

2.6.9 Εντολές Rotate

Υπάρχουν τρεις εντολές περιστροφής (Rotate), οι οποίες ανήκουν στις εντολές R-Type:

ror r*i*, r*j*, r*k*: η εντολή Rotate Right (δεξιά περιστροφή) περιστρέφει τα bits του καταχωρητή r*j* με κατεύθυνση από τα αριστερά προς τα δεξιά τόσες φορές, όσες ο αριθμός των θέσεων bits που ορίζονται από τα πέντε λιγότερο σημαντικά bits (εύρος αριθμού 0 έως 31) του καταχωρητή r*k*, και αποθηκεύει το αποτέλεσμα στον καταχωρητή r*i*.

rol r*i*, r*j*, r*k*: η εντολή Rotate Left (αριστερή περιστροφή) είναι παρόμοια με την εντολή ror, αλλά περιστρέφει σε κατεύθυνση από τα δεξιά προς τα αριστερά.

roli r*i*, r*j*, Value5: η εντολή Rotate Left Immediate (αριστερή άμεση περιστροφή) περιστρέφει τα bits του καταχωρητή r*j* με κατεύθυνση από τα δεξιά προς τα αριστερά τόσες θέσεις, όσες ο αριθμός των θέσεων των bits που ορίζονται από τα την 5-bit μη προσημασμένη τιμή, Value5, δίνεται στην εντολή, και αποθηκεύει το αποτέλεσμα στον καταχωρητή r*i*.

2.6.10 Εντολές Control

Υπάρχουν δύο ειδικές εντολές για την ανάγνωση και την εγγραφή των καταχωρητών ελέγχου [Ενότητα 2.4.2].

rdctl ***ri*, ctl*j*:** η εντολή Read Control Register (ανάγνωση καταχωρητή ελέγχου) αντιγράφει τα περιεχόμενα του καταχωρητή ελέγχου ctl*j* στον καταχωρητή γενικού σκοπού r*i*.

wrctl ctlj, ri: η εντολή Write Control Register (εγγραφή καταχωρητή ελέγχου) αντιγράφει τα περιεχόμενα του καταχωρητή γενικού σκοπού r*i* στον καταχωρητή ελέγχου ctl*j*.

Υπάρχουν δύο εντολές που παρέχονται για την αντιμετώπιση εξαιρέσεων: **trap** και **eret**. Είναι παρόμοιες με τις εντολές call και ret, αλλά χρησιμοποιούνται για τις εξαιρέσεις. Υπάρχουν, επίσης, εντολές για την διαχείριση της κρυφής μνήμης: **flushd** (Flush Data Cache Line), **flushi** (Flush Instruction Cache Line), **initd** (Initialize Data Cache Line), and **initi** (Initialize Instruction Cache Line). Επίσης, οι εντολές break και bret, οι οποίες δημιουργούν breaks και επιστρέφουν από breaks, αντίστοιχα.

2.6.11 Κρατούμενο και Υπερχείλιση

Όταν εκτελείτε μια αριθμητική πράξη, όπως πρόσθεση ή η αφαίρεση, είναι σημαντικό να γνωρίζουμε αν η πράξη δημιούργησε ένα κρατούμενο (carry) από την πιο σημαντική θέση bit ή αν συνέβη αριθμητική υπερχείλιση (overflow). Ένας επεξεργαστής που χρησιμοποιεί κωδικούς κατάστασης θέτει αυτόματα τις σημαίες C και V να αναφέρουν εάν συνέβησαν κρατούμενο ή υπερχείλιση. Ωστόσο, ο επεξεργαστής Nios II δεν περιλαμβάνει σημαίες κατάστασης. Οι εντολές Πρόσθεση (Add) και Αφαίρεση (Subtract) του Nios II εκτελούν τις αντίστοιχες λειτουργίες με τον ίδιο τρόπο και για προσημασμένους και για μη προσημασμένους τελεστές. Πρόσθετες εντολές πρέπει να χρησιμοποιηθούν για την ανίχνευση του κρατούμενου και της υπερχείλισης. Το κρατούμενο εξόδου (carry out) από τη θέση bit 31

παρουσιάζει ενδιαφέρον όταν προστίθενται ή αφαιρούνται αριθμοί χωρίς πρόσημο. Η υπερχείλιση είναι ενδιαφέρον όταν εμπλέκονται προσημασμένοι τελεστές.

Κρατούμενο και Υπερχείλιση στην Πρόσθεση

Παράδειγμα πρόσθεσης:

```
add    r4, r2, r3
```

Μετά την εκτέλεσή της μια πιθανή εμφάνιση κρατουμένου μπορεί να ανιχνευτεί ελέγχοντας αν το μη προσημασμένο άθροισμα, στον καταχωρητή r4, είναι μικρότερο από κάποιον από τους μη προσημασμένους τελεστές. Εάν αυτή η εντολή ακολουθείται από την εντολή

```
cmpltu    r5, r4, r2
```

το bit carry (κρατούμενο) θα γραφτεί στον καταχωρητή r5.

Εάν είναι επιθυμητό να συνεχιστεί η εκτέλεση με διακλάδωση (branch) στη θέση CARRY όταν ένα κρατούμενο του 1 ανιχνεύεται, αυτό μπορεί να επιτευχθεί με τη χρήση των εντολών

```
add     r4, r2, r3
bltu    r4, r2, CARRY
```

Η αριθμητική υπερχείλιση μπορεί να ανιχνευτεί ελέγχοντας τα πρόσημα των προσθετέων και του αθροίσματος. Υπερχείλιση συμβαίνει εάν δύο θετικοί αριθμοί προστεθούν και το αποτέλεσμα είναι αρνητικό, ή αν προστεθούν δύο αρνητικοί αριθμοί και το αποτέλεσμα είναι θετικό. Για παράδειγμα έχουμε την εξής αλληλουχία εντολών:

```
Add     r4, r2, r3
Xor     r5, r4, r2
Xor     r6, r4, r3
And     r5, r5, r6
Blt     r5, r0, OVERFLOW
```

η οποία προκαλεί διακλάδωση στην υπερχείλιση εάν η πρόσθεση καταλήγει σε αριθμητική υπερχείλιση. Οι δύο *xor* εντολές χρησιμοποιούνται για την σύγκριση των πρόσημων του αθροίσματος και κάθε έναν από τους δύο προσθετέους. Ενώ αυτές οι εντολές εκτελούν την XOR σε όλα τα 32 bits, μόνο το πρόσημο της θέσης του b31

εξετάζεται στην επόμενη εντολή διακλάδωσης. Αυτό το bit ορίζεται σε 1 μόνο αν τα πρόσημα των bits του αθροίσματος και των προσθετέων είναι διαφορετικά. Η εντολή *and* και προκαλεί το bit $r5_{31}$ να γίνει 1 μόνο αν τα πρόσημα των δύο τελεστών είναι οι ίδια, αλλά το πρόσημο του αθροίσματος είναι διαφορετικό. Η εντολή *blt* προκαλεί διακλάδωση εάν ο προσημασμένος αριθμός στον καταχωρητή r5 είναι αρνητικό, που φαίνεται αν το bit $r5_{31}$ είναι ίσο με 1.

Κρατούμενο και Υπερχείλιση στην Αφαίρεση

Το κρατούμενο και η υπερχείλιση στην αφαίρεση λειτουργίες μπορούν να ανιχνευθούν χρησιμοποιώντας μία παρόμοια προσέγγιση. Το κρατούμενο από το πιο σημαντικό bit της διαφοράς μπορεί να ανιχνευτεί ελέγχοντας εάν ο αφαιρέτης είναι μικρότερος από τον αφαιρετέο. Για παράδειγμα, οι εντολές

```
sub     r4, r2, r3
bltu    r2, r3, CARRY
```

θα προκαλέσουν διακλάδωση (branch) στην θέση CARRY, εάν ένα κρατούμενο δημιουργηθεί από την αφαίρεση.

Αριθμητική υπερχείλιση συμβαίνει αν ο αφαιρέτης και αφαιρετέος έχουν διαφορετικά πρόσημα και το πρόσημο της διαφοράς δεν είναι το ίδιο με το σήμα του αφαιρέτη. Αυτή η υπερχείλιση μπορεί να ανιχνευτεί από την εξής αλληλουχία εντολών:

```
sub     r4, r2, r3
xor     r5, r2, r3
xor     r6, r2, r4
and     r5, r5, r6
blt     r5, r0, OVERFLOW
```

Οι δύο εντολές *xor* συγκρίνουν το πρόσημο του αφαιρέτη με τα πρόσημα του αφαιρετέου και της διαφοράς. Η εντολή *and* προκαλεί το bit $r5_{31}$ να οριστεί σε 1 μόνο αν η παραπάνω προϋπόθεση για την υπερχείλιση είναι αληθής.

2.7 Custom Εντολές και Ψευδοεντολές

a) Custom Εντολές

Στον επεξεργαστή Nios II δίνεται η δυνατότητα της δημιουργίας custom (προσαρμοσμένων) εντολών. Η υλοποίηση κρίσιμων για το σύστημα λειτουργιών με custom εντολές αυξάνει κατά πολύ την απόδοση. Η εξειδικευμένη λογική που επιθυμούμε ενσωματώνεται στην αριθμητική και λογική μονάδα (ALU). Έχουν την ίδια μορφή με τις εντολές του Nios II. Παρέχεται στους σχεδιαστές ένα υψηλό επίπεδο προσπέλασης των εντολών, χρησιμοποιώντας μακροεντολές σε γλώσσες προγραμματισμού Assembly ή συναρτήσεις σε C.

b) Ψευδοεντολές

Για ευκολία στον προγραμματισμό, είναι χρήσιμο να υπάρχει μια ποικιλία διαφορετικών εντολών. Από την μεριά του hardware, ένας μεγάλος αριθμός εντολών απαιτεί εκτενέστερα κυκλώματα για την υλοποίησή τους. Συχνά, η λειτουργία μερικών επιθυμητών εντολών μπορεί να επιτευχθεί αποτελεσματικά με τη χρήση άλλων εντολών. Εάν αυτές οι επιθυμητές εντολές δεν εφαρμόζονται σε hardware, ονομάζονται ψευδοεντολές. Ο assembler αντικαθιστά τις ψευδοεντολές με πραγματικές εντολές που εφαρμόζονται σε hardware.

Όπως είδαμε στην ενότητα 2.6.4, οι εντολές Move είναι ψευδοεντολές. Άλλες ψευδοεντολές είναι οι εξής:

subi **r*i*, r*j*, Value16**: η εντολή Subtract Immediate υλοποιείται ως "addi r*i*, r*j*, −Value16"

bgt **r*i*, r*j*, LABEL:** η εντολή Branch Greater Than Signed υλοποιείται με την εντολή blt εναλλάσσοντας τους τελεστές των καταχωρητών.

Κατά τη σύνταξη ενός προγράμματος, ο προγραμματιστής δεν χρειάζεται να γνωρίζει τις ψευδοεντολές. Όμως, η επίγνωση γίνεται σημαντική, αν κάποιος προσπαθεί να εξετάσει τον κώδικα στον assembler, ίσως κατά τη διάρκεια της διαδικασίας εντοπισμού σφαλμάτων.

2.8 Εντολές του Assembler

Οι εντολές του assembler του Nios II προσαρμόζονται σε εκείνες που ορίζονται από το ευρέως χρησιμοποιημένο GNU assembler, το οποίο είναι διαθέσιμο λογισμικό στο κοινό. Οι εντολές αυτές ξεκινούν με μία περίοδο. Μερικές από τις συχνά χρησιμοποιούμενες εντολές περιγράφονται παρακάτω:

.equ ***symbol, expression*****:** θέτει την τιμή του symbol στην τιμή του expression.

.byte ***expressions*****:** τοποθετεί δεδομένα μεγέθους byte στην μνήμη. Τα δεδομένα ορίζονται από εκφράσεις, οι οποίες χωρίζονται με κόμμα.

.ascii ***"string" ...*** **:** Μια σειρά (string) από χαρακτήρες ASCII φορτώνεται σε διαδοχικές byte διευθύνσεις στη μνήμη. Πολλαπλά stings, χωρισμένα με κόμμα.

.asciz ***"string" ...*** **:** Αυτή η εντολή είναι η ίδια όπως η .ascii, εκτός από το ότι κάθε string ή ακολουθείται (τερματίζεται) από ένα zero byte.

.data: Προσδιορίζει τα δεδομένα που πρέπει να τοποθετηθούν στο τμήμα δεδομένων της μνήμης. Η επιθυμητή θέση μνήμης για το τμήμα των δεδομένων μπορεί να καθοριστεί στο σύστημα διαμόρφωσης του Altera Monitor Program.

.end: Σηματοδοτεί το τέλος του αρχείου πηγαίου κώδικα. Οτιδήποτε μετά από αυτήν την εντολή αγνοείται από τον assembler.

.global ***symbol*****:** Κάνει το symbol ορατό έξω από το object file του assembler.

.hword ***expressions*****:** είναι ίδια με την εντολή .byte, εκτός του ότι οι εκφράσεις συγκροτούνται σε διαδοχικές 16-bit halfwords.

.word ***expressions*****:** είναι ίδια με την εντολή .byte, εκτός του ότι οι εκφράσεις συγκροτούνται σε διαδοχικές 32-bit words.

.include ***"filename"*****:** Παρέχει ένα μηχανισμό για την συμπερίληψη αρχείων υποστήριξης σε ένα πηγαίο πρόγραμμα.

.org ***new-lc*****:** Προχωράει τον μετρητή θέσης από το new-lc, όπου το new-lc χρησιμοποιείται ως ένα offset (μετατόπιση) από την αρχική θέση που καθορίζεται στο σύστημα διαμόρφωσης του Altera Monitor Program. Η εντολή .org μπορεί να αυξήσει μόνο το μετρητή τοποθεσίας, ή να το αφήσει αμετάβλητο. Δεν μπορεί να μετακινήσει τον μετρητή θέσης προς τα πίσω.

.skip ***size*****:** Διατηρεί στη μνήμη τον αριθμό των bytes που ορίζεται ως Size. Η τιμή του κάθε byte είναι μηδέν.

.text: Προσδιορίζει τον κώδικα που θα πρέπει να τοποθετείται στο τμήμα κειμένου της μνήμης. Η επιθυμητή θέση μνήμης για το τμήμα του κειμένου μπορεί να καθορίζεται στο σύστημα διαμόρφωσης του Altera Monitor Program.

2.9 Διακοπές και Εξαιρέσεις

Η χρήση διακοπών είναι ένας αποτελεσματικός τρόπος για την εκτέλεση των μεταφορών I/O. Ένα σύστημα Nios II μπορεί να ασχοληθεί με δύο τύπους διακοπών. Μία αίτηση για εξυπηρέτηση από μία συσκευή I/O θεωρείται ότι είναι μία *διακοπή υλικού* (hardware interrupt). Οποιαδήποτε άλλη διακοπή δεν ονομάζεται διακοπή, αλλά *εξαίρεση* (exception). Στην πραγματικότητα, ο όρος εξαίρεση χρησιμοποιείται γενικά για να περιγράψει οποιαδήποτε απόκλιση που ξεκίνησε από το υλικό ή απόκλιση που ξεκίνησε από το λογισμικό από την κανονική εκτέλεση.

Μία εξαίρεση στην κανονική ροή της εκτέλεσης του προγράμματος μπορεί να προκληθεί από:

- Software trap (παγίδα λογισμικού)
- Hardware interrupt (διακοπή υλικού)
- Unimplemented instruction (ανεφάρμοστες εντολές)

Ως αποτέλεσμα μίας εξαίρεσης, ο επεξεργαστής Nios II εκτελεί αυτόματα τις παρακάτω ενέργειες:

1. Αποθηκεύει τις υπάρχουσες πληροφορίες κατάστασης του επεξεργαστή αντιγράφοντας τα περιεχόμενα του καταχωρητή κατάστασης (*status*) (*ctl0*) στον καταχωρητή *estatus* (*ctl1*).
2. Καθαρίζει το bit *U* στον καταχωρητή *status*, για να βεβαιωθεί ότι ο επεξεργαστής είναι σε Supervisor mode.
3. Διαγράφει το bit *PIE* στον καταχωρητή *status*, απενεργοποιώντας έτσι τις επιπλέον εξωτερικές διακοπές του επεξεργαστή.
4. Γράφει την διεύθυνση της εντολής μετά την εξαίρεση στον καταχωρητή *ea* (r29).
5. Μεταφέρει την εκτέλεση στην διεύθυνση του χειριστή εξαίρεσης, η οποία καθορίζει την αιτία της εξαίρεσης και στέλνει μια κατάλληλη ρουτίνα εξαίρεσης να ανταποκριθεί στην εξαίρεση.

Ο *χειριστής εξαιρέσεων* (*exception handler*) είναι ένα πρόγραμμα που ασχολείται με τις εξαιρέσεις. Είναι φορτωμένο σε μία προκαθορισμένη θέση στη μνήμη, στην οποία μεταφέρεται ο έλεγχος εκτέλεσης όταν παρουσιάζεται μια εξαίρεση. Η διεύθυνση του χειρισμού εξαιρέσεων καθορίζεται κατά τον χρόνο δημιουργίας του συστήματος με τη χρήση του SOPC Builder, και δεν μπορεί να αλλάξει από το λογισμικό κατά το χρόνο εκτέλεσης. Αυτή η διεύθυνση μπορεί να παρέχεται από τον σχεδιαστή, αλλιώς η προεπιλεγμένη διεύθυνση είναι 2016 από την αρχική διεύθυνση της κύριας μνήμης. Για παράδειγμα, αν η μνήμη ξεκινάει στη διεύθυνση 0, τότε η προεπιλεγμένη διεύθυνση του χειρισμού εξαιρέσεων είναι 0x00000020.

2.9.1 Παγίδα Λογισμικού

Μια εξαίρεση λογισμικού συμβαίνει όταν μια εντολή παγίδα συναντάται σε ένα πρόγραμμα. Αυτή η εντολή αποθηκεύει τη διεύθυνση της επόμενης εντολής στον καταχωρητή *ea* (r29). Στη συνέχεια, απενεργοποιεί τις διακοπές και μεταφέρει την εκτέλεση στον χειρισμό εξαιρέσεων.

Στην ρουτίνα exception-service η τελευταία εντολή είναι η eret (Επιστροφή Εξαίρεσης), η οποία επιστρέφει τον έλεγχο της εκτέλεσης στην εντολή που

ακολουθεί την εντολή παγίδα που προκάλεσε την εξαίρεση. Η διεύθυνση επιστροφής δίνεται από τα περιεχόμενα του καταχωρητή *ea*. Η εντολή eret επαναφέρει την προηγούμενη κατάσταση του επεξεργαστή αντιγράφοντας τα περιεχόμενα του καταχωρητή *estatus* στον καταχωρητή *status.*

Μια κοινή χρήση της παγίδας λογισμικού (software trap) είναι να μεταφέρει τον έλεγχο σε ένα διαφορετικό πρόγραμμα, όπως ένα λειτουργικό σύστημα.

2.9.2 Διακοπές Υλικού

Οι διακοπές υλικού (Hardware interrupt) μπορούν να αυξηθούν από εξωτερικές πηγές, όπως οι συσκευές I/O, διεκδικώντας μία από τις 32 εισόδους του επεξεργαστή με αίτημα για διακοπή, από irq0 εως irq31. Μια διακοπή δημιουργείται μόνο εάν πληρούνται οι ακόλουθες τρεις συνθήκες:

- Το bit *PIE* στον καταχωρητή status έχει οριστεί 1.
- Μία είσοδος αίτησης διακοπής, *irqk* , είναι βεβαιωμένη.
- Το αντίστοιχο bit ενεργοποίησης διακοπής, $ctl3_k$, τίθεται 1.

Τα περιεχόμενα του καταχωρητή *ipending* (ctl4) υποδεικνύουν ποιες αιτήσεις διακοπών εκκρεμούν. Μια ρουτίνα εξαίρεσης καθορίζει ποιες από τις εν αναμονή διακοπές έχει την υψηλότερη προτεραιότητα, και μεταφέρει τον έλεγχο στην αντίστοιχη ρουτίνα εξυπηρέτησης διακοπής.

Μετά την ολοκλήρωση της ρουτίνας εξυπηρέτησης διακοπής, ο έλεγχος εκτέλεσης επιστρέφεται στο διακεκομμένο πρόγραμμα μέσω της εντολής eret, όπως εξηγήθηκε παραπάνω. Ωστόσο, δεδομένου ότι η εξωτερική αίτηση διακοπής αντιμετωπίζεται χωρίς προηγουμένως να έχει ολοκληρώσει την εντολή που εκτελείται όταν εμφανίζεται η διακοπή, η διακεκομμένη εντολή πρέπει να εκτελεστεί πάλι κατά την επιστροφή από τη ρουτίνα εξυπηρέτησης διακοπής. Για να επιτευχθεί αυτό, η ρουτίνα εξυπηρέτησης διακοπής πρέπει να προσαρμόσει τα περιεχόμενα του καταχωρητή *ea,* τα οποία αυτή τη στιγμή δείχνουν στην επόμενη

εντολή του διακεκομμένου προγράμματος. Ως εκ τούτου, η τιμή στον καταχωρητή *ea* πρέπει να μειωθεί κατά 4 πριν από την εκτέλεση της εντολής *eret*.

2.9.3 Ανεφάρμοστες Εντολές

Αυτή η εξαίρεση συμβαίνει όταν ο επεξεργαστής συναντά μια έγκυρη εντολή που δεν έχει υλοποιηθεί σε hardware. Αυτό μπορεί να συμβαίνει με τις εντολές όπως οι mul και div. Ο χειριστής εξαιρέσεων μπορεί να καλέσει μια ρουτίνα που προσομοιώνει την απαιτούμενη πράξη στο λογισμικό.

2.9.4 Προσδιορισμός του Τύπου της Εξαίρεσης

Όταν συμβαίνει μια εξαίρεση, η ρουτίνα χειρισμού εξαιρέσεων πρέπει να προσδιορίσει το είδος της εξαίρεσης που έχει συμβεί. Η σειρά με την οποία θα πρέπει να ελέγχονται οι εξαιρέσεις είναι η εξής:

1. Διαβάζει τον καταχωρητή *ipending*, για να δει αν έχει συμβεί μια διακοπή hardware. Αν ναι, τότε πηγαίνει στην κατάλληλη ρουτίνα εξυπηρέτησης διακοπών.
2. Διαβάζει την εντολή που εκτελούνταν όταν συνέβη η εξαίρεση. Η διεύθυνση αυτής της εντολής είναι η τιμή στον καταχωρητή *ea* μείον 4. Αν αυτή είναι η εντολή παγίδα, τότε πηγαίνει στην ρουτίνα χειρισμού παγίδων λογισμικού.
3. Διαφορετικά, η εξαίρεση οφείλεται σε ανεφάρμοστη εντολή.

2.9.5 Επανεκκίνηση (Reset)

Ένα σύστημα Nios II πρέπει να περιλαμβάνει μια δυνατότητα *επανεκκίνησης* (*reset*) για να καταστεί δυνατή η ανάκτηση από μια λανθασμένη κατάσταση που δεν μπορεί να αντιμετωπίζεται ως εξαίρεση. Αυτό μπορεί να γίνει με την παροχή ενός πλήκτρου επαναφοράς (reset key). Όταν πατηθεί το πλήκτρο, ο επεξεργαστής κάνει

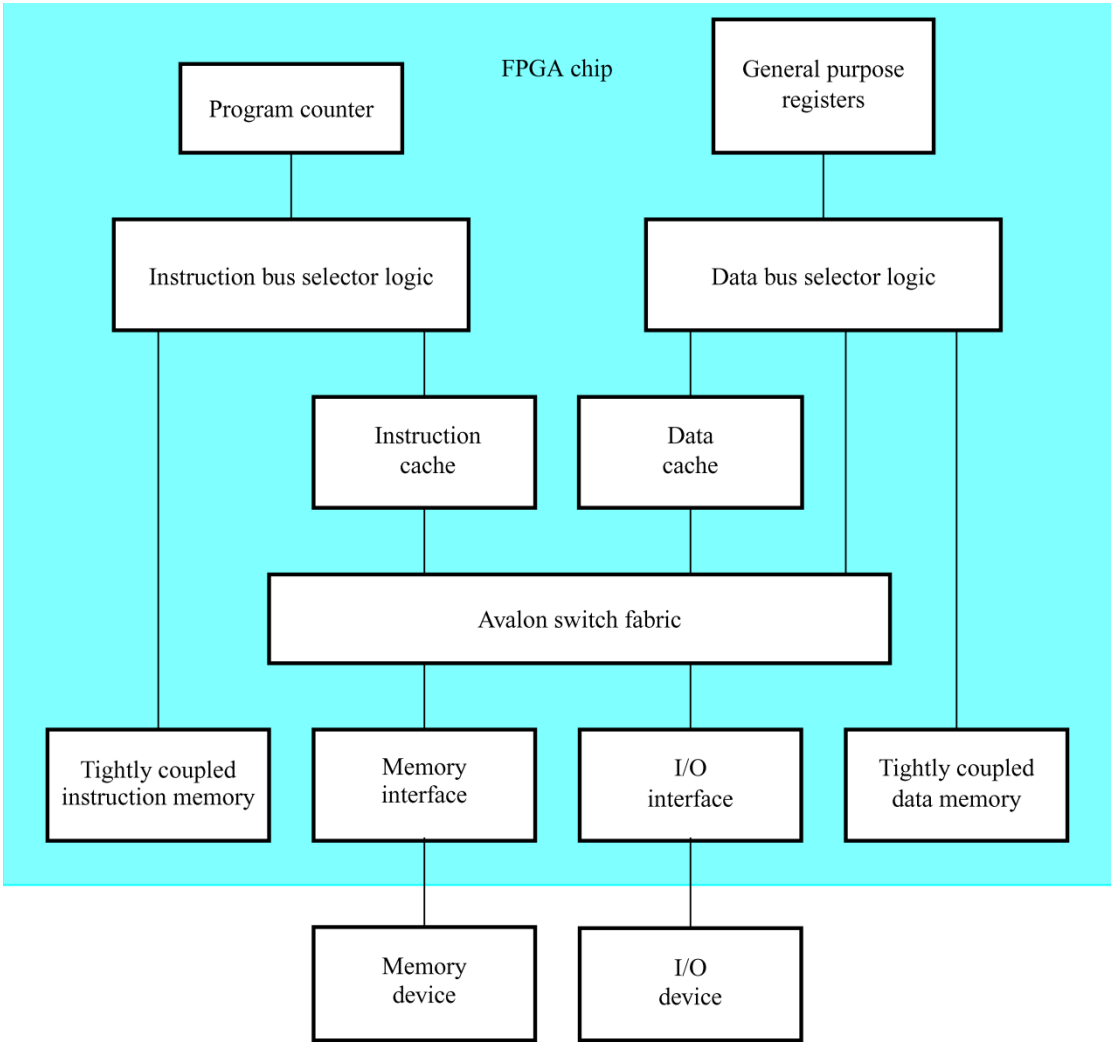
επανεκκίνηση και ένα κατάλληλο πρόγραμμα εκτελείται. Εάν η μνήμη ξεκινάει στη διεύθυνση 0, τότε είναι φυσικό να χρησιμοποιήσει αυτή τη διεύθυνση ως τη θέση επαναφοράς, η οποία μπορεί να γίνει όταν το σύστημα Nioss II έχει τεθεί σε εφαρμογή. Όταν ο επεξεργαστής κάνει επανεκκίνηση, ο μετρητής προγράμματος και οι καταχωρητές ελέγχου γίνονται μηδέν. Έτσι, η εκτέλεση αρχίζει με την εντολή στη διεύθυνση 0. Για να εκτελεστεί το κύριο πρόγραμμα μετά την επαναφορά, είναι απαραίτητο να τοποθετηθεί μία εντολή διακλάδωσης (branch) στη διεύθυνση 0 με την πρώτη εντολή του κύριου προγράμματος ως στόχου διακλάδωσης.

2.10 Κρυφή Μνήμη (Cache)

Όπως φαίνεται στο Σχήμα 2.2, ένα σύστημα Nios II μπορεί να περιλαμβάνει κρυφές μνήμες εντολών και δεδομένων, οι οποίες εφαρμόζονται στα block μνήμης στο τσιπ του FPGA. Οι κρυφές μνήμες ή προσωρινές (caches) μπορούν να καθοριστούν όταν ένα σύστημα σχεδιάζεται χρησιμοποιώντας το λογισμικό SOPC Builder. Οι κρυφές μνήμες βελτιώνουν σημαντικά την απόδοση ενός συστήματος Nios II, ειδικά όταν το μεγαλύτερο μέρος της κύριας μνήμης παρέχεται από ένα εξωτερικό τσιπ SDRAM, όπως στην περίπτωση των πλακετών (board) σειράς DE της Altera. Οι κρυφές μνήμες εντολών και δεδομένων είναι απευθείας χαρτογραφημένες στην μνήμη.

Η *κρυφή μνήμη των εντολών* (instruction cache) μπορεί να εφαρμοστεί στις εκδόσεις "fast" και "standard" των συστημάτων του επεξεργαστή Nios II. Είναι οργανωμένη σε 8 λέξεις ανά γραμμή κρυφής μνήμης, και το μέγεθός της είναι μία παράμετρος σχεδιασμού την οποία επιλέγει ο χρήστης.

Η *κρυφή μνήμη δεδομένων* (data cache) μπορεί να υλοποιηθεί μόνο με την "fast" έκδοση του επεξεργαστή Nios II. Διαθέτει ένα ρυθμιζόμενο μέγεθος της γραμμής των 4, 16 ή 32 bytes ανά γραμμή κρυφής μνήμης. Το συνολικό μέγεθος του είναι επίσης μία παράμετρος σχεδιασμού την οποία επιλέγει ο χρήστης.



Σχήμα 2.2: Οργάνωση μνήμης και συσκευών I/O [15]

2.10.1 Διαχείριση της Κρυφής Μνήμης

Η διαχείριση της κρυφής μνήμης γίνεται μέσω του λογισμικού. Για το σκοπό αυτό το σύνολο των εντολών του Nios II περιλαμβάνει τις ακόλουθες εντολές:

- **initd IMMED16 (rA)** (Initialize data-cache line): Ακυρώνει την γραμμή στην κρυφή μνήμη δεδομένων που συνδέεται με την διεύθυνση που καθορίζεται προσθέτοντας την προσημασμένη και επεκταμένη τιμή, IMMED16, με τα περιεχόμενα του καταχωρητή rA.
- **initi rA** (Initialize instruction-cache line): Ακυρώνει τη γραμμή στην κρυφή μνήμη εντολών που συνδέεται με την διεύθυνση που περιέχεται στον καταχωρητή rA.
- **flushd IMMED16(rA)** (Flush data-cache line): Υπολογίζει την ενεργή διεύθυνση, προσθέτοντας την προσημασμένη και επεκταμένη τιμή, IMMED16, με τα

περιεχόμενα του καταχωρητή rA. Στη συνέχεια, προσδιορίζει τη γραμμή της κρυφής μνήμης που συνδέεται με αυτή την ενεργή διεύθυνση, γράφει κάθε άχρηστο δεδομένο στη γραμμή κρυφής μνήμης πίσω στη μνήμη, και ακυρώνει τη γραμμή της κρυφής μνήμης.

- **flushi rA** (Flush instruction-cache line): Ακυρώνει τη γραμμή στην κρυφή μνήμη εντολών που συνδέεται με τη διεύθυνση που περιέχεται στον καταχωρητή rA.

2.10.2 Μέθοδοι Παράκαμψης Κρυφής Μνήμης

Ένας επεξεργαστής Nios II χρησιμοποιεί την κρυφή μνήμη δεδομένων του με τον τυπικό τρόπο. Όμως, επιτρέπει επίσης την κρυφή μνήμη να παρακαμφθεί με δύο τρόπους. Οι εντολές Load και Store έχουν μια εκδοχή που προορίζεται για την πρόσβαση σε συσκευές I / O, όπου η ενεργή διεύθυνση προσδιορίζει μια θέση σε μια διασύνδεση συσκευών I / O. Αυτές οι εντολές είναι οι: ldwio, ldbio, lduio, ldhio, ldhuio, stwio, stbio, και sthio. Αυτές παρακάμπτουν την κρυφή μνήμη δεδομένων.

Ένας άλλος τρόπος παράκαμψης της κρυφής μνήμης δεδομένων, χρησιμοποιώντας το bit31 της διεύθυνση ως μια ετικέτα που δείχνει αν ο επεξεργαστής θα πρέπει να μεταφέρει τα δεδομένα από/προς την κρυφή μνήμη, ή την παρακάμπτει. Αυτή η λειτουργία είναι διαθέσιμη μόνο στον επεξεργαστή Nios II/f.

2.11 Στενά Συνδεδεμένη Μνήμη

Ένας επεξεργαστής Nios II μπορεί να έχει πρόσβαση στα μπλοκ μνήμης του τσιπ FPGA ως στενά συνδεδεμένη μνήμη (*tightly coupled memory*). Η διάταξη αυτή δεν χρησιμοποιεί το Avalon network. Αντ' αυτού, η στενά συνδεδεμένη μνήμη συνδέεται απευθείας στον επεξεργαστή. Τα δεδομένα στη στενά συνδεδεμένη μνήμη είναι προσβάσιμα χρησιμοποιώντας τις εντολές Load και Store τις οδηγίες. Τα κυκλώματα ελέγχου του Nios II καθορίζουν εάν η διεύθυνση μιας θέσης μνήμης είναι στη στενά συνδεδεμένη μνήμη. Για τη διεύθυνση της στενά συνδεδεμένης μνήμης, ο επεξεργαστής λειτουργεί σαν οι κρυφές μνήμες να μην υπάρχουν.

2.12 Προηγμένες Ρυθμίσεις του Επεξεργαστή Nios II

Στις προηγούμενες ενότητες αναπτύχθηκαν τα χαρακτηριστικά των βασικών ρυθμίσεων του επεξεργαστή Nios II. Είναι δυνατή η υλοποίηση μεγαλύτερων επεξεργαστών Nios II, που περιλαμβάνουν πρόσθετες μονάδες υλικού που παρέχουν αυξημένες δυνατότητες. Παρακάτω δίνονται τρεις από τις πιθανές αναβαθμίσεις.

2.12.1 Εξωτερικός Ελεγκτής Διακοπών

Η Ενότητα 2.9 περιγράφει τον μηχανισμό που χρησιμοποιείται από τον εσωτερικό ελεγκτή διακοπών, στον οποίο το λογισμικό χρησιμοποιείται για να καθορίζει την προτεραιότητα των αιτημάτων διακοπών. Για τη μείωση του χρόνου αναμονής, ο επεξεργαστής είναι δυνατόν να περιλαμβάνει ένα εξωτερικό κύκλωμα ελέγχου διακοπών που χρησιμοποιεί vectored διακοπές. Σε αυτήν την περίπτωση, ο ελεγκτής παρέχει τη διεύθυνση της ρουτίνας εξυπηρέτησης διακοπής για κάθε αίτηση διακοπής.

Για περαιτέρω μείωση του χρόνου αναμονής των διακοπών, ο επεξεργαστής μπορεί να περιλαμβάνει καταχωρητές shadow για τους 32 καταχωρητές γενικής χρήσης. Αρκετοί καταχωρητές shadow μπορούν να υλοποιηθούν και να συνδέονται με διαφορετικές διακοπές. Ο εξωτερικός ελεγκτής διακοπών προσδιορίζει το σύνολο των καταχωρητών shadow να χρησιμοποιείται με μία εισερχόμενη αίτηση διακοπής. Στη συνέχεια, το σύνολο αυτών των εντολών χρησιμοποιείται αντί του κανονικού συνόλου καταχωρητών. Αυτό παρακάμπτει την ανάγκη για αποθήκευση των περιεχομένων των καταχωρητών που χρησιμοποιούνται στη ρουτίνα εξυπηρέτησης διακοπών.

Τα επίπεδα προτεραιότητας συνδέονται με διαφορετικές διακοπές. Όταν ο επεξεργαστής εξυπηρετεί μια διακοπή ενός δεδομένου επιπέδου προτεραιότητας, μπορεί να διακόπτεται μόνο από άλλη διακοπή με υψηλότερο επίπεδο προτεραιότητας. Το τρέχον επίπεδο προτεραιότητας του επεξεργαστή και μια

ένδειξη του ενεργού συνόλου των καταχωρητών shadow είναι ένα μέρος του επεξεργαστή, το οποίο είναι στον καταχωρητή ελέγχου κατάστασης, ctl0. Αυτή η πληροφορία βρίσκεται στα *reserved* bits κομμάτια του Πίνακας 2.2.

Όταν ένας επεξεργαστής Nios II ορίζεται, έχει διαμορφωθεί είτε με τον εσωτερικό ελεγκτή διακοπών ή τον εξωτερικό ελεγκτή διακοπών.

2.12.2 Διαχείριση Μονάδας Μνήμης

Ένα σύστημα Nios II μπορεί να περιλαμβάνει μια μονάδα διαχείρισης μνήμης (Memory Management Unit - MMU), η οποία παρέχει λειτουργικότητα. Η MMU έχει ως στόχο να υποστηρίξει λειτουργικά συστήματα που χρησιμοποιούν τη δυνατότητα διαχείρισης μνήμης. Ένα σύστημα που ενσωματώνει μία MMU μπορεί να χρησιμοποιήσει τους τρόπους λειτουργίας Supervisor και User. Η MMU είναι μια προαιρετική μονάδα η οποία πρέπει να καθορίζεται για να συμπερίληψη στο σύστημα κατά το χρόνο σχεδίασης.

2.12.3 Υλικό Κινητής Υποδιαστολής

Οι προσαρμοσμένες εντολές (custom instructions), όπως είδαμε στην Ενότητα 2.7, μπορούν να χρησιμοποιηθούν για να καθορίσουν μια ποικιλία από πράξεις, οι οποίες μπορεί να απαιτούν τη χρήση επιπλέον κυκλωμάτων. Ένα σύνολο από προκαθορισμένες προσαρμοσμένες εντολές είναι διαθέσιμο για την υλοποίηση της κινητής υποδιαστολής σε αριθμητικές πράξεις. Το απαραίτητο hardware περιλαμβάνεται σε ένα σύστημα Nios II κατά το χρόνο σχεδίασης αν είναι επιθυμητές πράξεις κινητής υποδιαστολής.

2.13 Παραδείγματα Προγραμμάτων στον Nios II

1ο Παράδειγμα: Ας υποθέσουμε ότι υπάρχει μια συμβολοσειρά (*string*) από ASCII κωδικοποιημένους χαρακτήρες που αποθηκεύονται στη μνήμη ξεκινώντας από τη διεύθυνση STRING. Η συμβολοσειρά τελειώνει με τον χαρακτήρα Carriage Return (CR). Το παρακάτω πρόγραμμα του Nios II θα καθορίζει το μήκος της συμβολοσειράς.

```
      movia r2, STRING  /*ο r2 δείχνει στην αρχή του string.*/
      add   r3, r0, r0  /*ο r3 είναι ένας μετρητής που μηδενίζεται*/
      addi  r4, r0, 0x0D /*Φόρτωση του ASCII code για τον CR.*/
LOOP: ldb   r5, (r2)    /* Λαμβάνει τον επόμενο χαρακτήρα.*/
      beq   r5, r4, DONE /*Σταματάει εάν ο χαρακτήρας είναι ο CR.*/
      addi  r2, r2, 1   /* Αύξηση του δείκτη του string κατά 1.*/
      addi  r3, r3, 1   /* Αύξηση του μετρητή κατά 1.*/
      br    LOOP        /* Εάν δεν έχει τελειώσει, πάει ξανά στο
                            loop.*/
DONE: movia r2, LENGTH  /* Αποθηκεύει την τιμή στην θέση*/
      stw   r3, (r2)    /* μνήμης LENGTH. */
```

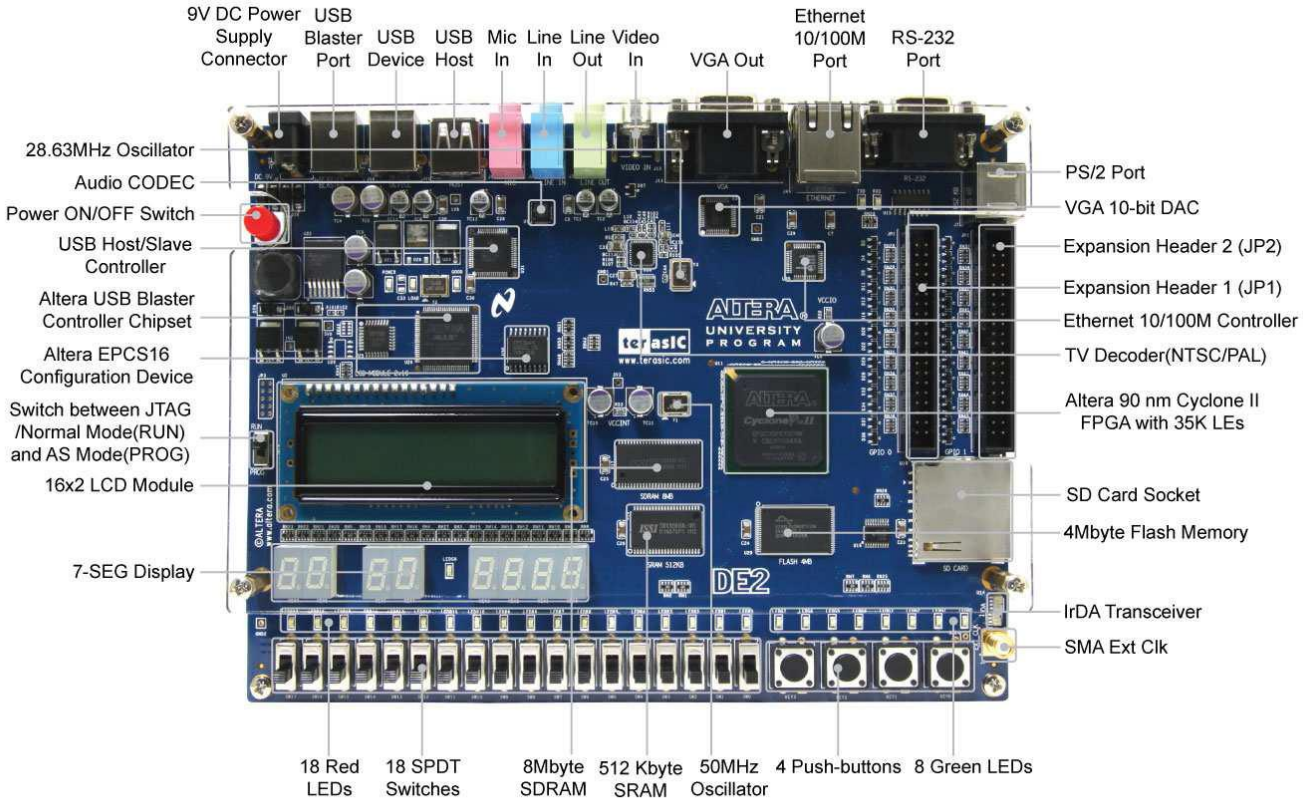
Στο παραπάνω πρόγραμμα κάθε χαρακτήρας στη συμβολοσειρά (string) συγκρίνεται με το Carriage Return - CR (0x0D κώδικας ASCII), και ένας μετρητής αυξάνεται μέχρι να φτάσει το τέλος του string. Το αποτέλεσμα αποθηκεύεται στη θέση LENGTH.

2ο Παράδειγμα: Ας υποθέσουμε ότι έχουμε να προσθέσουμε δύο ακέραιους που είναι πολύ μεγάλοι ώστε να χωρέσουν σε 32-bit καταχωρητές. Αυτό μπορεί να γίνει με τη φόρτωση των αριθμών σε δύο διαφορετικούς καταχωρητές και, στη συνέχεια, την εκτέλεση της πρόσθεσης με ανίχνευση κρατουμένου. Θα χρησιμοποιήσουμε δεκαεξαδικούς αριθμούς ώστε να είναι εύκολο να δούμε πώς ένας αριθμός μπορεί να αναπαρασταθεί σε δύο καταχωρητές. Έστω A=10A72C10F8 και B = 4A5C00FE04. Η πρόσθεση, C = A + B, μπορεί να υπολογιστεί όπως φαίνεται παρακάτω.

```
orhi   r2, r0, 0xA72C        /* Ο r2 περιέχει τον A72C0000.*/
ori    r2, r2, 0x10F8        /* Ο r2 περιέχει τον A72C10F8.*/
ori    r3, r0, 0x10          /* Ο r3 περιέχει τον 10.*/
orhi   r4, r0, 0x5C00        /* Ο r4 περιέχει τον 5C000000.*/
ori    r4, r4, 0xFE04        /* Ο r4 περιέχει τον 5C00FE04.*/
ori    r5, r0, 0x4A          /* Ο r5 περιέχει τον 4A.*/
add    r6, r2, r4      /* Πρόσθεση των 32 bits χαμηλής τάξης. */
cmpltu r7, r6, r2            /* Έλεγχος αν υπάρχει κρατούμενο.*/
add    r7, r7, r3            /* Πρόσθεση του κρατούμενου και */
add    r7, r7, r5            /* των bits υψηλής τάξης.*/
```

Οι καταχωρητές r2 και r3 φορτώνονται με τα χαμηλής και υψηλής τάξης 32 bits του αριθμού A, αντίστοιχα. Οι καταχωρητές r4 και r5 χρησιμοποιούνται για να κρατήσουν τον B με τον ίδιο τρόπο. Παρατηρούμε ότι οι 32-bit τιμές στους καταχωρητές r2 και r4 φορτώνονται χρησιμοποιώντας δύο άμεσους τελεστές των 16-bit. Μετά την πρόσθεση των 32 bits χαμηλής τάξης του A και του B, η διεξαγωγή το εξαγόμενο κρατούμενο (carry out) περιλαμβάνεται στην πρόσθεση των 32 bits υψηλής τάξης. Το άθροισμα C = 5B032D0EFC τοποθετείται στους καταχωρητές r6 και r7.

Κεφάλαιο 3 : Η Αναπτυξιακή Πλακέτα DE2 Της Altera



Εικόνα 3.1: Η αναπτυξιακή πλακέτα DE2 [16]

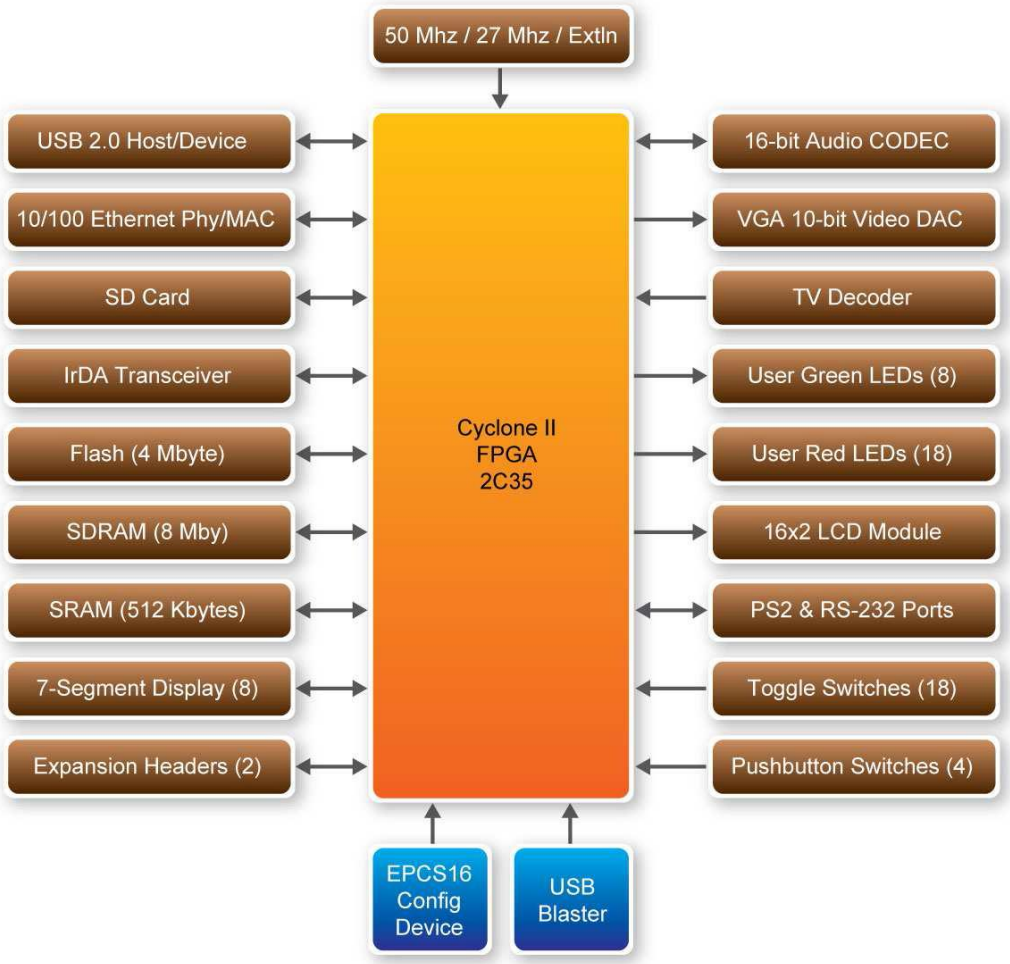
3.1 Σκοπός της πλακέτας DE2

Η αναπτυξιακή και εκπαιδευτική πλακέτα DE2 της Altera [Εικόνα 3.1] έχει αναπτυχθεί για την εκμάθηση ψηφιακής λογικής και την υπολογιστική οργάνωση σε εργαστηριακό περιβάλλον. Σχεδιάστηκε για να χρησιμοποιείται σε εργαστηριακά ακαδημαϊκά μαθήματα που έχουν σχέση με τον σχεδιασμό λογικών κυκλωμάτων και την οργάνωση του υπολογιστή. Η πλακέτα DE2 μπορεί να είναι κατάλληλη για μια ποικιλία εργασιών σχεδιασμού, αλλά ταυτόχρονα και για την ανάπτυξη εξελιγμένων ψηφιακών συστημάτων.

3.2 Περιγραφή της πλακέτας DE2 της Altera

Η παραπάνω εικόνα [Εικόνα 4.1] απεικονίζει την διάταξη της πλακέτας DE2 και δείχνει τη θέση των διασυνδέσεων και τα βασικά υποσυστήματα. Η καρδιά του συστήματος είναι το Cyclone II FPGA chip. Όλα τα σημαντικά επιμέρους υποσυστήματα στην πλακέτα συνδέονται με τους ακροδέκτες του τσιπ, επιτρέποντας στο χρήστη τη ρύθμιση των διασυνδέσεων μεταξύ των διαφόρων υποσυστημάτων όπως επιθυμεί.

Για την διασύνδεση και την χρήση του αναπτυξιακού είναι απαραίτητη η σύνδεση με υπολογιστή μέσω της θύρας USB. Γι' αυτό χρειάζεται και η εγκατάσταση στον υπολογιστή των USB-Blaster drivers κατά την εγκατάσταση του προγράμματος Quartus II, μέσω του οποίου γίνεται ο σχεδιασμός συστημάτων και παρέχεται από την Altera.



Σχήμα 3.1: Block διάγραμμα του αναπτυξιακού DE2 [16]

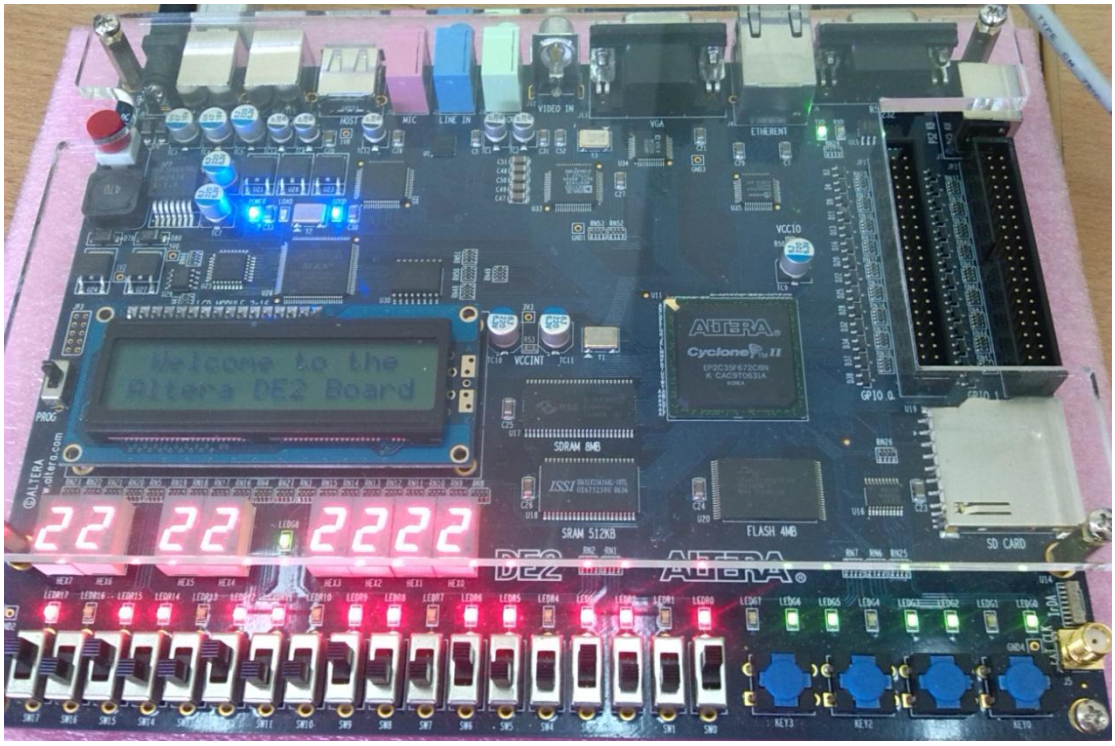
Όπως φαίνεται και στο block διάγραμμα [Σχήμα 3.1] η αναπτυξιακή πλακέτα DE2 εμπεριέχει αναλυτικά τα εξής:

- **Cyclone II 2C35 FPGA:** περιέχει 33.216 λογικά στοιχεία, 105 M4K RAM blocks, συνολικά 483.840 bits RAM, 35 ενσωματωμένους πολλαπλασιαστές, 4 PLLs, 475 I/O pins (ακροδέκτες εισόδου/εξόδου) για τον χρήστη και πακέτο FineLine BGA 672 pins.
- **Σειριακή διάταξη μορφοποίησης και κύκλωμα USB Blaster:** Η πλακέτα υποστηρίζει παράλληλη και σειριακή (JTAG και AS) μέθοδο προγραμματισμού, εμπεριέχει σειριακή διάταξη μορφοποίησης, την EPCS16 της Altera καθώς και USB Blaster, εγκατεστημένο πάνω στην αναπτυξιακή πλακέτα για προγραμματισμό και για έλεγχο χρήστη API.
- **Μνήμη Static RAM (SRAM):** μεγέθους 512-Kbyte, οργανωμένη ως 256K x 16 bits.
- **Synchronous Dynamic RAM (SDRAM):** δυναμική μνήμη RAM των 8-Mbyte, οργανωμένη ως 1M x 16 bits x 4 banks.
- **Μνήμη Flash:** είναι μεγέθους 4-Mbyte (1 Mbyte σε μερικές πλακέτες) με 8-bit δίαυλο δεδομένων.

  Όλες οι μνήμες είναι διαθέσιμες από τον επεξεργαστή Nios II και από το DE2 Control Panel.
- **Υποδοχή κάρτας SD:** παρέχει SPI και 4-bit SD λειτουργία για πρόσβαση σε SD κάρτες και είναι προσβάσιμη ως μνήμη για τον επεξεργαστή Nios II, μέσω της εγκατάστασης του DE2 SD Card Driver.
- **Pushbutton switches:** Στην πλακέτα παρέχονται 4 διακόπτες pushbutton που βρίσκονται κανονικά σε υψηλή στάθμη (λογικό 1). Όταν όμως πιέζονται τα pushbuttons, παράγεται παλμός σε χαμηλή στάθμη, λογικό 0 (active-low).
- **Toggle switches:** Επίσης υπάρχουν 18 διακόπτες εναλλαγής για είσοδο από τον χρήστη. Ένας διακόπτης παράγει λογικό 0 όταν είναι στη θέση “down” (πιο κοντά στην άκρη της πλακέτας) και λογικό 1 όταν είναι στη θέση “up”.
- **Οθόνη LCD:** μεγέθους 16 x 2 για την προβολή του κειμένου.
- **LEDs:** Για τις ανάγκες εξόδου υπάρχουν φωτεινοί ενδείκτες (LEDs). Συγκεκριμένα 9 πράσινα LEDs και 18 κόκκινα.

- **Είσοδοι ρολογιού:** Υπάρχουν δύο κρύσταλλοι χρονισμού, 50-MHz ταλάντωσης και 27-MHz πηγή ρολογιού και μέσω SMA έχει και εξωτερική πηγή ρολογιού.
- **Audio CODEC:** Για τον ήχο υπάρχει ένα audio CODEC (Wolfson WM8731) των 24 bits sigma-delta, με γραμμή εισόδου, γραμμή εξόδου και υποδοχές για μικρόφωνο. Η συχνότητα δειγματοληψίας κυμαίνεται από 8 έως 96 KHz. Εφαρμογές που στηρίζει: συσκευές αναπαραγωγής και εγγραφής MP3, smart phones, συσκευές ηχογράφησης κ.α.
- **Έξοδος VGA:** χρησιμοποιείται ο VGA μετατροπέας, ADV7123, των 140-MHz με triple 10bit υψηλής ταχύτητας video DAC. Με υποδοχή D-sub με 15 ακροδέκτες υψηλής πυκνότητας, υποστηρίζει αναλύσεις έως και 1600 x 1200 με ρυθμό ανανέωσης 100HZ. Μπορεί να χρησιμοποιηθεί με το Cyclone II FPGA για την υλοποίηση ενός υψηλής απόδοσης κωδικοποιητή τηλεόρασης.
- **Αποκωδικοποιητής τηλεόρασης NTSC/PAL:** χρησιμοποιείται ο αποκωδικοποιητής ADV7180 Multi-format και υποστηρίζει παγκοσμίως NTSC/PAL/SECAM έγχρωμη αποδιαμόρφωση. Επίσης υποστηρίζει υποδοχή σύνθετου σήματος Video (CVBS) και ψηφιακές μορφές εξόδου (8-bit/ 16-bit). Εφαρμογές που στηρίζει είναι: οι συσκευές εγγραφής DVD, LCD TV, ψηφιακή τηλεόραση κ.α.
- **10/100 Ethernet ελεγκτής:** υποστηρίζει πλήρη αμφίδρομη επικοινωνία με ταχύτητες 10 Mb/s και 100Mb/s και είναι συμβατό με την προδιαγραφή IEEE 802.3u
- **USB Host/Slave ελεγκτής:** που είναι συμβατό με το Universal Serial Bus Specification Rev. 2.0, υποστηρίζει μεταφορά δεδομένων σε υψηλή αλλά και χαμηλή ταχύτητα και μπορεί να λειτουργήσει και ως USB host και ως συσκευή. Υπάρχουν δύο θύρες USB που παρέχουν υψηλής ταχύτητας παράλληλη διασύνδεση στους περισσότερους επεξεργαστές και υποστηρίζει προγραμματισμένη είσοδο/έξοδο (PIO) και άμεση προσπέλαση μνήμης (DMA).
- **Σειριακές θύρες:** Υπάρχουν δύο σειριακές θύρες. Μία σειριακή θύρα είναι η RS-232, για την οποία υπάρχει ο σειριακός σύνδεσμος DB-9, και η δεύτερη είναι η PS/2, στην οποία συνδέονται σειριακά ένα ποντίκι ή πληκτρολόγιο PS2 πάνω στην αναπτυξιακή πλακέτα DE2.

- **Πομποδέκτης IrDA:** Είναι ένας πομποδέκτης υπερύθρων με ρυθμό μετάδοσης δεδομένων 115,2-kb/s, με μετάδοση ρεύματος στα LED στα 32mA, ολοκληρωμένη ασπίδα EMI, IEC825-1 Class 1 eye safe και είσοδο Edge detection.
- **Δύο κεφαλές επέκτασης των 40-ακροδεκτών (pins):** 72 ακροδέκτες εισόδου εξόδου (I/O) της Cyclone II, καθώς και 8 γραμμές τροφοδοσίας και γείωσης, εξέρχονται σε δύο υποδοχές επέκτασης με 40 pins. Κεφαλή με 40 pins είναι σχεδιασμένη για να δεχτεί μία τυπική καλωδιοταινία των 40-pin που χρησιμοποιείται για IDE σκληρούς δίσκους.



Εικόνα 3.2: Καλωσόρισμα της αναπτυξιακής πλακέτας DE2 της Altera αμέσως μετά την ενεργοποίησή της.

Κεφάλαιο 4 : Εισαγωγή Στα Εργαλεία Σχεδίασης Ψηφιακών Κυκλωμάτων

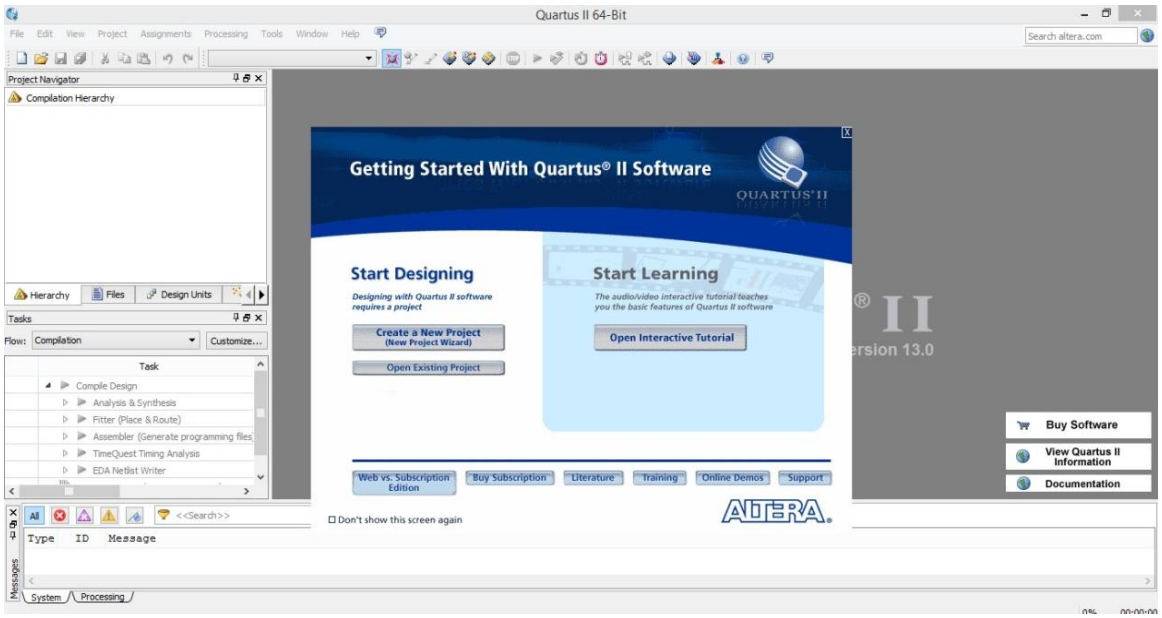
Οι κατασκευαστές των τσιπ FPGA παρέχουν ισχυρά εργαλεία σχεδιασμού (CAD - Computer Aided Design), τα οποία κάνουν τον σχεδιασμό ενσωματωμένων συστημάτων ευκολότερο. Μια ποικιλία από προκαθορισμένες μονάδες (modules) παρέχονται σε παραμετροποιημένη μορφή. Ο σχεδιαστής δημιουργεί ένα σύστημα συμπεριλαμβάνοντας αυτά τα modules και προσδιορίζοντας τις παραμέτρους για να ταιριάζουν με τις απαιτήσεις της εκάστοτε εφαρμογής. Παραδείγματα τέτοιων modules είναι οι πυρήνες του επεξεργαστή, οι μονάδες μνήμης και οι διεπαφές, οι παράλληλες διασυνδέσεις εισόδου/εξόδου (Parallel I/O), οι σειριακές διασυνδέσεις I/O, το χρονόμετρο/μετρητής κυκλωμάτων. Αυτά τα modules μπορεί να είναι επαρκή για να πραγματοποιήσουν όλες τις λειτουργίες που απαιτούνται σε ένα επιθυμητό ενσωματωμένο σύστημα. Αν δεν είναι, τότε επιπλέον εξειδικευμένα κυκλώματα πρέπει να σχεδιαστούν και να συμπεριληφθούν στο σύστημα.

Τυπικά, ένα υποσύστημα που περιλαμβάνει ένα πυρήνα επεξεργαστή και άλλα παραμετροποιημένα modules καθορίζεται πρώτο. Ένα εργαλείο σχεδίασης ψηφιακών κυκλωμάτων (CAD tool) χρησιμοποιείται για να δημιουργήσει ένα module το οποίο θα υλοποιεί το υποσύστημα. Αυτό το module ορίζεται σε γλώσσα περιγραφής υλικού. Στη συνέχεια ενσωματώνεται στο συνολικό σχέδιο, μαζί με τυχόν πρόσθετα κυκλώματα για συγκεκριμένες εφαρμογές που έχουν δημιουργηθεί. Τέλος, ένα διαφορετικό εργαλείο CAD χρησιμοποιείται για τη σύνθεση και την εφαρμογή του συνολικού σχεδίου σε τέτοια μορφή που μπορεί να χρησιμοποιηθεί για την ρύθμιση του FPGA.

Εκτός από το τσιπ του FPGA, είναι απαραίτητο να περιλαμβάνονται τα εξωτερικά εξαρτήματα που απαιτούνται για την ολοκλήρωση του συστήματος, όπως διακόπτες, οθόνες, και επιπρόσθετα τσιπς μνήμης. Τέτοια εξαρτήματα πρέπει να συνδέονται με τις κατάλληλες ακίδες (pins) του FPGA.

Τα εργαλεία CAD της εταιρείας ALTERA που χρησιμοποιούνται για την σχεδίαση ψηφιακών κυκλωμάτων είναι τα εξής:

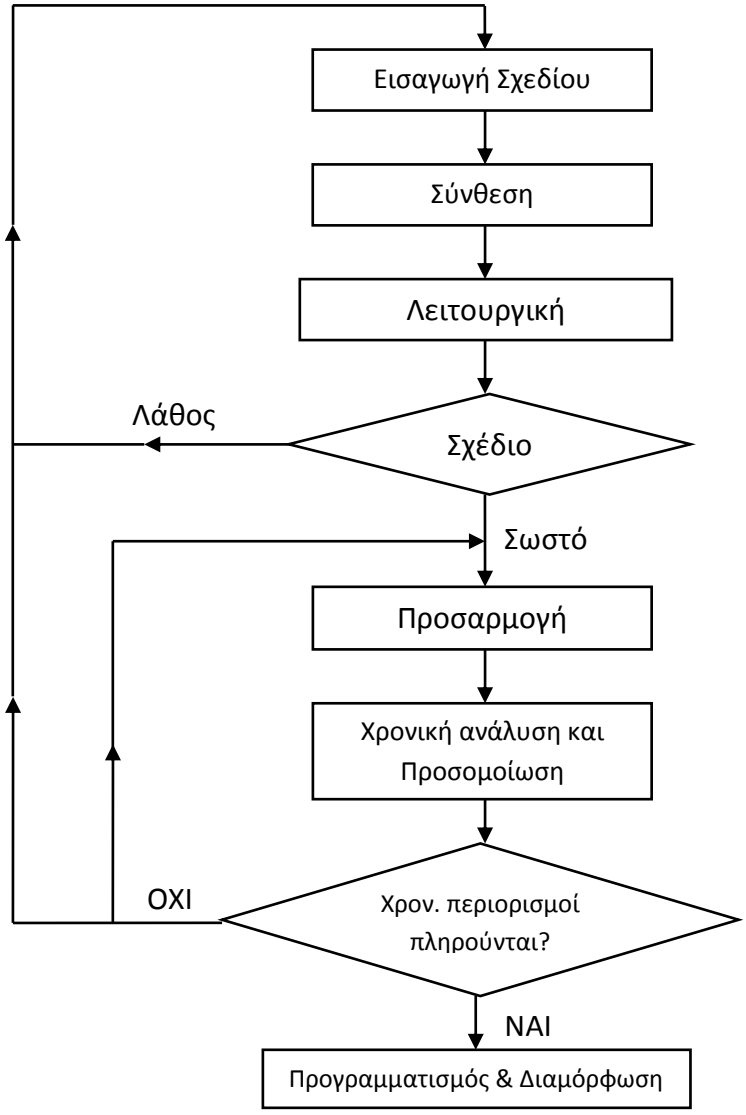
4.1 Quartus II



Εικόνα 4.1: Το παράθυρο του λογισμικού Quartus II (Έκδοση 13)

Το λογισμικό Quartus II είναι ένα από τα γνωστότερα προγράμματα σχεδίασης CAD (Computer-Aided Design). Είναι πνευματική ιδιοκτησία της εταιρείας Altera. Το Quartus II χρησιμοποιείται για την ανάπτυξη και τον προγραμματισμό όλων των αναπτυξιακών κυκλωμάτων της εταιρείας Altera, δηλαδή των διατάξεων CPLDs και FPGAs που κατασκευάζει η εταιρεία.

Στην Εικόνα 4.1 απεικονίζεται το παράθυρο του λογισμικού Quartus II αμέσως μετά το άνοιγμα του. Εγώ χρησιμοποίησα την έκδοση την 13. Όμως στην ιστοσελίδα της Altera υπάρχουν διαθέσιμες και νεότερες εκδόσεις. Η νεότερη έκδοση του Quartus II είναι η version 15. Ωστόσο για την αναπτυξιακή πλακέτα η νεότερη έκδοση που μπορούμε να χρησιμοποιήσουμε είναι η 13, γιατί οι νεότερες από την 13 δεν υποστηρίζουν το Cyclone II FPGA που περιέχει η αναπτυξιακή πλακέτα DE2.



Σχήμα 4.1: Ροη διεργασιών στο Quartus II [17]

Το Σχήμα 4.1, μας δείχνει την ροή διεργασιών στο Quartus II, της οποίας τις βαθμίδες θα αναλύσουμε στις επόμενες παραγράφους.

4.1.1 Εισαγωγή σχεδίου

Κατά την εισαγωγή σχεδίου, ο χρήστης περιγράφει το κύκλωμα που θέλει να υλοποιήσει. Υπάρχουν τρεις τρόποι περιγραφής ενός κυκλώματος: η σχεδίαση με την βοήθεια πινάκων αληθείας, η σχεδίαση με βάση σχηματικά διαγράμματα και η σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής υλικού (HDL).

Στην **σχεδίαση με την βοήθεια πινάκων αληθείας**, ο χρήστης μέσω ενός επεξεργαστή κυματομορφών εισάγει στο πρόγραμμα τις τιμές των εισόδων και τις επιθυμητές τιμές των εξόδων για το κύκλωμά του. Για αυτόν τον τρόπο χρησιμοποιούμε στο Quartus II τον *Waveform Editor*. Όμως προτιμάται μόνο για απλά και μικρά κυκλώματα γιατί δεν είναι εύχρηστος.

Η **σχεδίαση με βάση σχηματικά διαγράμματα** γίνεται με την βοήθεια σχεδιαστικών εργαλείων που παρέχονται από το πρόγραμμα σχεδίασης και με την χρήση εγκατεστημένων βιβλιοθηκών, που περιέχουν απλές πύλες ή και σύνθετα κυκλώματα. Για αυτόν τον τρόπο χρησιμοποιούμε στο Quartus II τον *Block Editor*. Είναι κατάλληλος και για σύνθετα κυκλώματα.

Τέλος, **η σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής υλικού (HDL)**. Η VHDL είναι η πιο ευρέως διαδεδομένη γλώσσα περιγραφής υλικού. Υπάρχει επίσης και η Verilog. Η σχεδίαση ενός κυκλώματος με μία πρότυπη γλώσσα περιγραφής υλικού παρέχει στον χρήστη την δυνατότητα της μεταφοράς σε κάποιο άλλο σύστημα σχεδίασης, χωρίς να χρειάζεται αλλαγή στον κώδικα. Αυτό είναι και το πλεονέκτημα σε αντίθεση με τους άλλους τρόπους. Ένα κοινό σημείο που έχει η σχεδίαση με την VHDL με το σχηματικό διάγραμμα είναι ότι ο κώδικας HDL μπορεί να πάρει ιεραρχική μορφή, δίνοντας μας την δυνατότητα να σχεδιάσουμε σύνθετα κυκλώματα. Για αυτόν τον τρόπο χρησιμοποιούμε στο Quartus II τον *Text Editor* μέσω του οποίου περιγράφουμε το ψηφιακό σύστημα με βάση τη συμπεριφορά του.

4.1.2 Σύνθεση

Το επόμενο βήμα μετά την εισαγωγή σχεδίου είναι η σύνθεση (*Analysis & Synthesis*). Κατά την διάρκεια αυτής της διαδικασίας η είσοδος μετατρέπεται στις κατάλληλες λογικές συναρτήσεις, με τρόπο που να ταιριάζει στην τεχνολογία της συγκεκριμένης διάταξης που τελικά θα διαμορφώσουμε. Δηλαδή η σύνθεση χρησιμοποιείται ως ένα αυτόματο εργαλείο υλοποίησης του αρχικού κυκλώματος που δίνει ο σχεδιαστής, με τρόπο κατάλληλο για την τεχνολογία του στόχου.

Ταυτόχρονα, το κύκλωμα που παίρνουμε στην έξοδο της διαδικασίας αυτής είναι βελτιστοποιημένο σε σχέση με το αρχικό.

4.1.3 Λειτουργική Προσομοίωση (functional Simulation)

Το επόμενο βήμα είναι η λειτουργία της προσομοίωσης (*Simulation*). Κατά την διάρκεια της προσομοίωσης ελέγχουμε κατά πόσο το κύκλωμά μας λειτουργεί σωστά. Ο έλεγχος της σωστής λειτουργίας γίνεται στο τμήμα των διεργασιών που ονομάζεται προσομοίωση. Εδώ συνδυάζονται το αρχικό σχέδιο με τις τιμές που δίνει ο χρήστης στις εισόδους του κυκλώματος ώστε να ελέγξει αν οι τιμές που θα πάρει στην έξοδο του ανταποκρίνονται στις αρχικές προδιαγραφές που είχαμε θέσει για το κύκλωμά μας. Στην λειτουργική προσομοίωση δε λαμβάνονται υπόψη οι καθυστερήσεις των στοιχείων (πυλών και διασυνδέσεων) του κυκλώματος αλλά απλά επαληθεύεται ότι η λογική συνάρτηση που υλοποιεί το κύκλωμα είναι η σωστή. Αν η διαδικασία αυτή είναι επιτυχημένη το πρόγραμμα συνεχίζει στο επόμενο βήμα, την προσαρμογή. Διαφορετικά ξεκινάει τα βήματα από την αρχή.

4.1.4 Προσαρμογή (fitting)

Η διαδικασία της προσαρμογής (fitting) λέγεται αλλιώς και δρομολόγηση (place and route). Δηλαδή, στη φάση αυτή καθορίζεται το πόσα και ποια συγκεκριμένα λογικά στοιχεία (logic elements - LE’s) του ολοκληρωμένου κυκλώματος (chip) θα χρησιμοποιηθούν. Επίσης επιλέγονται συνδέσεις από τον προγραμματιζόμενο πίνακα διασυνδέσεων (programmable interconnect), ώστε να διασυνδεθούν τα απαραίτητα LE’s μεταξύ τους.

4.1.5 Χρονική ανάλυση και Προσομοίωση

Αναλύονται οι καθυστερήσεις μεταδόσεως κατά μήκος των διαφόρων διαδρομών (λόγω του μήκους των καλωδίων και του αριθμού των ενδιάμεσων βαθμίδων) στο κύκλωμα και παρέχεται μία ένδειξη της αναμενόμενης απόδοσης του κυκλώματος.

Με την χρονική προσομοίωση επαληθεύουμε την ορθότητα του κυκλώματος με βάση τους χρονικούς περιορισμούς του. Εμφανίζει τη χειρότερη περίπτωση καθυστέρησης στην έξοδο.

4.1.6 Προγραμματισμός και διαμόρφωση της συσκευής

Η διαδικασία του προγραμματισμού (Compilation) είναι η τελευταία ενέργεια που γίνεται για την ολοκλήρωση της δημιουργίας του κυκλώματος μας. Τα CPLDs ή τα FPGAs πρέπει να προγραμματιστούν για να υλοποιήσουν το κύκλωμα που σχεδιάσαμε. Η Altera επιτρέπει τον προγραμματισμό των συσκευών της με δύο τρόπους. Ο ένας είναι μέσω του κυκλώματος διεπαφής JTAG και ο άλλος ο Active Serial (AS) mode.

Τα διαμορφωμένα αρχεία μεταφέρονται από τον υπολογιστή του χρήστη στο board μέσω ενός καλωδίου το οποίο συνδέεται σε θύρα του υπολογιστή μας (παράλληλη ή USB) και στο board όπου βρίσκεται το CPLD ή το FPGA. Η σύνδεση αυτή γίνεται μέσω του κατάλληλου οδηγού (driver) USB-Blaster ή BYTE-BLASTER. Στη περίπτωση που χρησιμοποιήσουμε την JTAG διεπαφή, τα δεδομένα μας πηγαίνουν κατευθείαν στο ολοκληρωμένο κύκλωμα. Με αυτόν τον τρόπο το ολοκληρωμένο κύκλωμα (αν είναι FPGA) διατηρεί την διαμόρφωση που του έχουμε δώσει για όσο διαρκεί η τροφοδοσία του. Αν σταματήσει να τροφοδοτείται χάνεται και η διαμόρφωση του. Αυτό δεν ισχύει για τα κυκλώματα CPLDs, τα οποία διαμορφώνονται μόνιμα, με δυνατότητα επανεγγραφής, όπως οι μνήμες flash EEPROM.

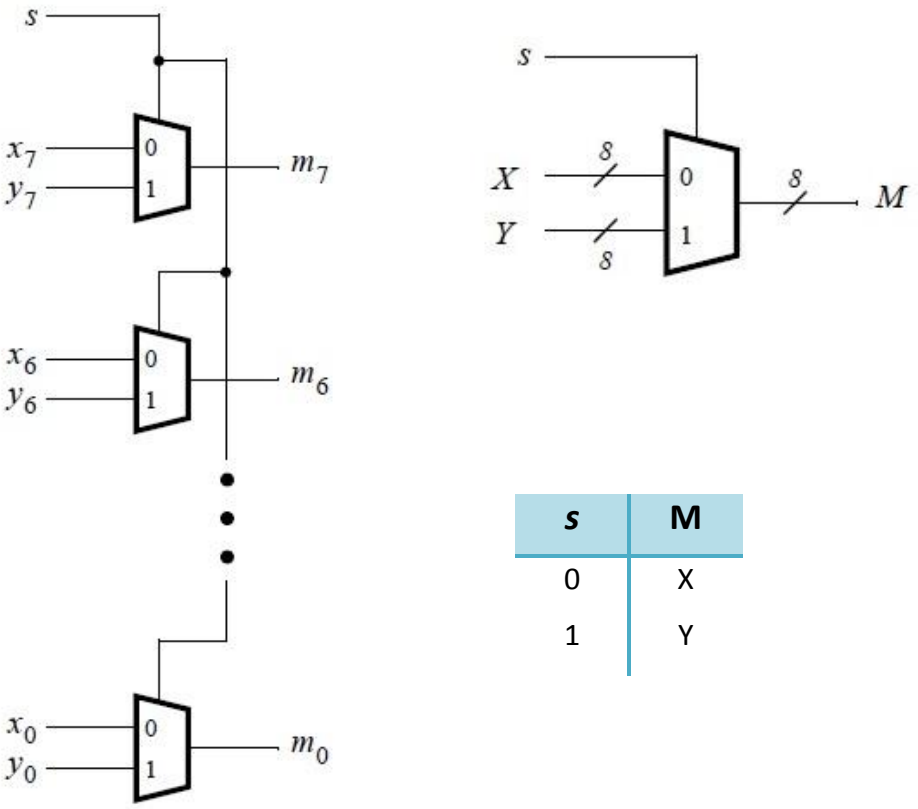
Στην άλλη περίπτωση (AS) μια διάταξη με μνήμες flash, που βρίσκεται πάνω στην ίδια πλακέτα με το FPGA, χρησιμοποιείται για να αποθηκεύει τα αρχεία διαμόρφωσης. Σε αυτήν την περίπτωση το Quartus II στέλνει τα δεδομένα στη μνήμη Flash και αυτή με τη σειρά της στο chip FPGA. Στην περίπτωση αυτή δεν μας ενδιαφέρει αν υπάρχει τροφοδοσία ή όχι. Η επιλογή για το ποιόν από τους δύο τρόπους θα χρησιμοποιήσουμε γίνεται συνήθως μέσω ενός διακόπτη RUN/PROG

που βρίσκεται πάνω στο board. Η αυτόματη (default) επιλογή είναι για διαμόρφωση με JTAG, ενώ η δεύτερη για Active Serial (AS).

Τέλος για να επιβεβαιώσουμε ότι το κύκλωμα μας λειτουργεί σωστά πρέπει να το δοκιμάσουμε δίνοντας κατάλληλες εισόδους 0 ή 1 από τους διακόπτες ή άλλες συσκευές εισόδου που βρίσκονται πάνω στο board. Το αποτέλεσμα των εξόδων εμφανίζεται στους κατάλληλους ακροδέκτες εξόδου, που πρέπει να τους συνδέσουμε με συσκευές απεικόνισης (π.χ. LEDs) πάνω στην αναπτυξιακή πλακέτα.

4.1.7 Πρώτο Παράδειγμα Εργαστηριακής Άσκησης στο Quartus II

Το Σχήμα 4.2 δείχνει ένα κύκλωμα που υλοποιεί έναν *πολυπλέκτη 2 προς 1* με μια είσοδο επιλογής *s*, με δύο εισόδους των *8-bits*, *X* και *Y*, και παράγει την 8-bit έξοδο *M*. Να γραφεί μία VHDL οντότητα που να περιέχει οχτώ καταστάσεις εκχώρησης για να περιγράψει το Σχήμα 4.2.



s	M
0	X
1	Y

Σχήμα 4.2: Κύκλωμα πολυπλέκτη 2 προς 1 - Σύμβολο - Πίνακας αληθείας [20]

Αρχικά κάνουμε ένα νέο project στο Quartus II. Και στην συνέχεια επιλέγουμε **File > New > VHDL File** (Αναλυτικότερη ανάλυση βημάτων δημιουργίας του project στο Quartus II στην Ενότητα 4.6).

Σύμφωνα με τον πίνακα αληθείας του παραπάνω σχήματος, εάν *s* = 0 τότε *M=X*, ενώ εάν *s* = 1 τότε *M = Y,* όπου X, Y και M 8-bit είσοδοι και έξοδοι. Οπότε το top-level VHDL αρχείο διαμορφώνεται όπως παρακάτω:

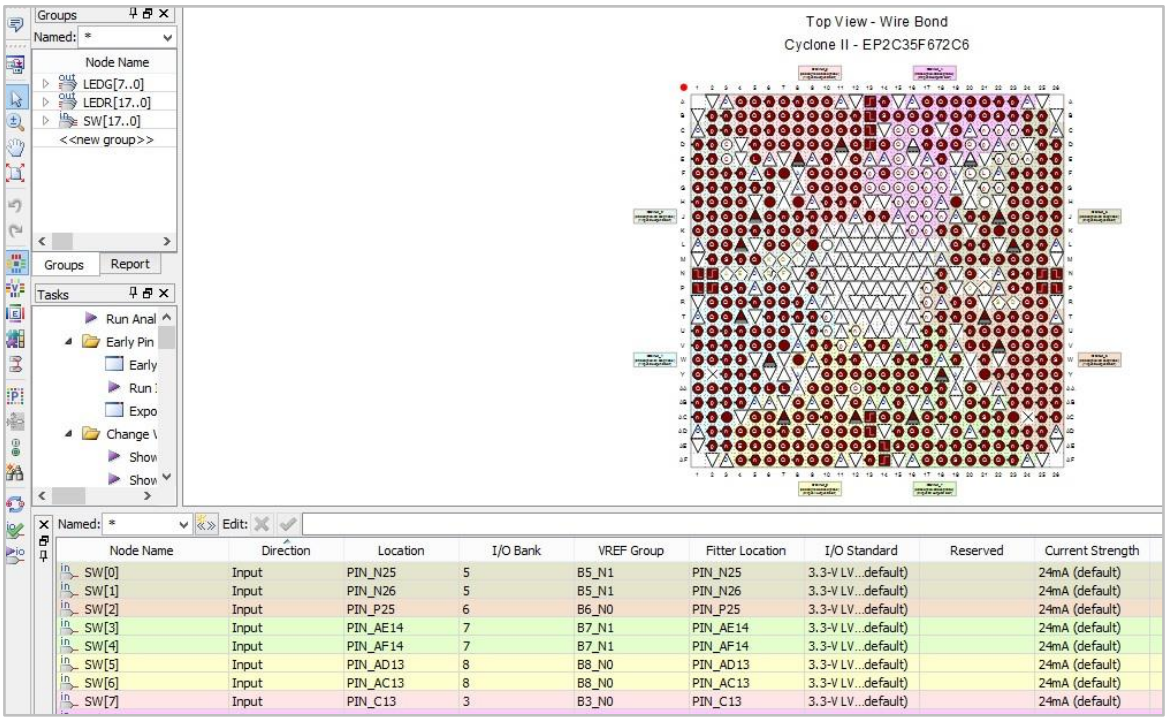
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

--2-σε-1 πολυπλέκτης με εισόδους των 8-bit
--Δήλωση εισόδων (διακόπτες),εξόδων (πράσινα και κόκκινα leds)

ENTITY lab1_p2 IS
      PORT ( SW  : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
             LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
             LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0));
END lab1_p2;

ARCHITECTURE structural OF lab1_p2 IS

  COMPONENT mux_8bit_2to1
      PORT (X, Y : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
             S    : IN STD_LOGIC;
             M    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
  END COMPONENT;


BEGIN
      stage: mux_8bit_2to1 PORT MAP (SW(15 DOWNTO 8),
                  SW(7 DOWNTO 0), SW(17), LEDG(7 DOWNTO 0));

--Το LEDR(17) θα λειτουργεί ανάλογα με τον διακόπτη SW(17),
--δηλαδή την είσοδο επιλογής s
      LEDR(17) <= SW(17);
--Τα 16 κόκκινα leds θα λειτουργούν ανάλογα με τους διακόπτες
--SW0-15
      LEDR(15 DOWNTO 0) <= SW(15 DOWNTO 0);

END structural;
```

Το αρχείο VHDL που περιλαμβάνει τον component mux_8bit_2to1 διαμορφώνεται ως εξής:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_8bit_2to1 IS
      PORT (X,Y  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               S   : IN STD_LOGIC;
               M   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END mux_8bit_2to1;

ARCHITECTURE Behavior OF mux_8bit_2to1 IS
BEGIN
      WITH S SELECT
            M<= X WHEN '0',
                Y WHEN OTHERS;
END Behavior;
```

Όπου *SW* είναι οι διακόπτες, *LEDG* τα πράσινα leds και *LEDR* τα κόκκινα leds. Χρησιμοποιούμε τον διακόπτη SW_{17} ως την είσοδο επιλογής *s,* τους διακόπτες SW_{7-0} ως την είσοδο *Y* και τους SW_{15-8} ως την είσοδο *X*. Ο διακόπτες *SW* συνδέονται με τα αντίστοιχα κόκκινα leds *LEDR* και η έξοδος *M* με τα πράσινα leds $LEDG_{7-0}$.

Το επόμενο βήμα είναι να αντιστοιχίσουμε τις εισόδους και τις εξόδους του κυκλώματός μας με συγκεκριμένους ακροδέκτες της διάταξης που πρόκειται να προγραμματίσουμε. Υπάρχουν δύο τρόποι για τον **ορισμό των ακροδεκτών (pin assignments)**. Στον πρώτο, επιλέγουμε **Assignments > Pin Planer** και ανοίγει το παρακάτω παράθυρο [Εικόνα 4.2]. Στην θέση Location ορίζουμε τα pins που αντιστοιχούν στις εισόδους και τις εξόδους του κυκλώματος.

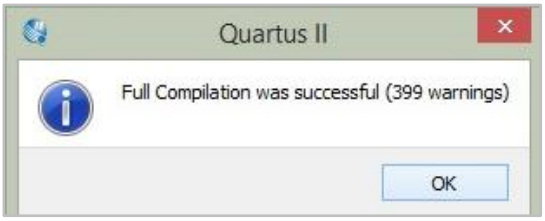


Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
SW[0]	Input	PIN_N25	5	B5_N1	PIN_N25	3.3-V LV...default)		24mA (default)
SW[1]	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)		24mA (default)
SW[2]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)		24mA (default)
SW[3]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)		24mA (default)
SW[4]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)		24mA (default)
SW[5]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)		24mA (default)
SW[6]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)		24mA (default)
SW[7]	Input	PIN_C13	3	B3_N0	PIN_C13	3.3-V LV...default)		24mA (default)

Εικόνα 4.2: Παράθυρο ορισμού ακροδεκτών για το Cyclone II EP2C35F672C6.

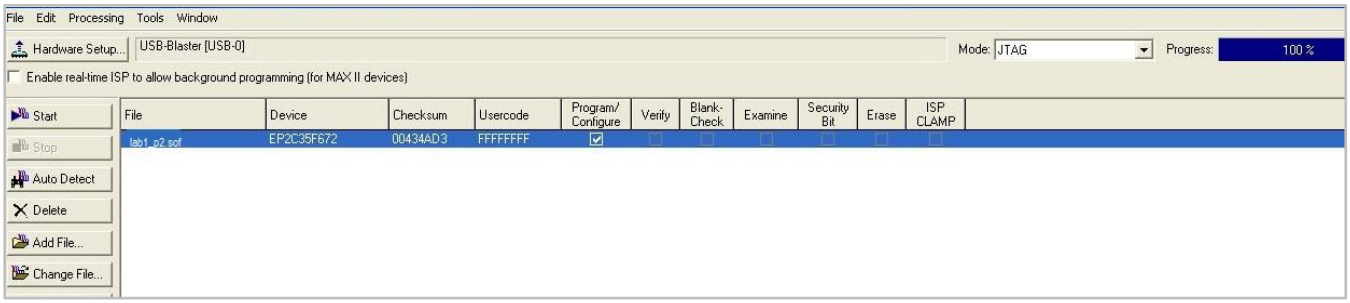
Ο δεύτερος και πιο εύχρηστος τρόπος είναι ο εξής: επιλέγουμε **Assignments > Import Assignments** και επιλέγουμε ένα αρχείο .qsf από την τοποθεσία που βρίσκεται, το οποίο παρέχει η Altera στην ιστοσελίδα της (www.altera.com) και ονομάζεται *de2_pin.qsf* για την αναπτυξιακή πλακέτα DE2. Όταν προσθέτουμε αυτό το αρχείο εκχωρούνται όλοι οι ακροδέκτες του FPGA.

Αφού ολοκληρωθεί ο ορισμός των pins και πριν πραγματοποιηθεί η διαμόρφωση του FPGA θα πρέπει να γίνει **Μετάφραση (Compilation)** ώστε να δημιουργηθεί το αρχείο *.sof το οποίο παρέχει όποιες πληροφορίες χρειάζονται για την διαμόρφωση του FPGA. Επιλέγουμε **Processing > Start Compilation** για να γίνει η μετάφραση του κώδικα. Η μετάφραση περιλαμβάνει την Ανάλυση & Σύνθεση, τη Δρομολόγηση, την Χρονική Ανάλυση και την παραγωγή των αρχείων προγραμματισμού του FPGA. Μόλις ολοκληρωθεί η μετάφραση πρέπει να εμφανιστεί το μήνυμα της Εικόνας 4.3, για να μπορέσουμε να συνεχίσουμε.



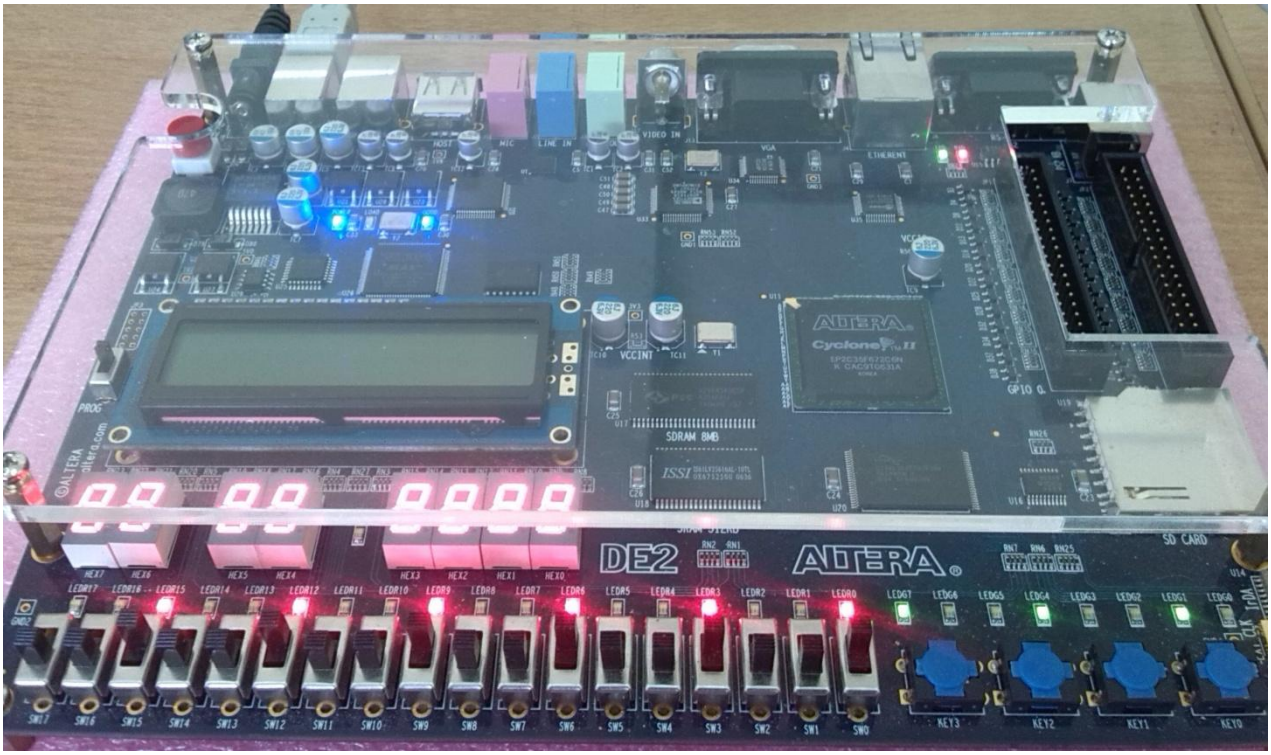
Εικόνα 4.3: Μετάφραση (Compilation) του VHDL αρχείου

Τώρα μπορεί να γίνει η διαμόρφωση του FPGA με τον Programmer, ο οποίος βρίσκεται στο **Tools > Programmer**. Στο παράθυρο του Programmer πρέπει να επιλέξουμε το κατάλληλο υλικό (Hardware Setup). Οι επιλογές του Hardware Setup είναι ο οδηγός **Byte Blaster** για προγραμματισμό μέσω παράλληλης θύρας ή ο **USB Blaster** για προγραμματισμό μέσω της θύρας USB, τον οποίο χρησιμοποιούμε και στο αναπτυξιακό DE2. Αν δεν είναι ήδη επιλεγμένο θα πρέπει να επιλέξουμε το τελικό αρχείο διαμόρφωσης (*lab1_p2.sof*) που δημιουργήθηκε από την μετάφραση. Κατόπιν, επιλέγοντας *Start* αρχίζει η διαδικασία διαμόρφωσης. Η πρόοδος της διαδικασίας φαίνεται στην μπάρα πάνω δεξιά της οθόνης [Εικόνα 4.4].



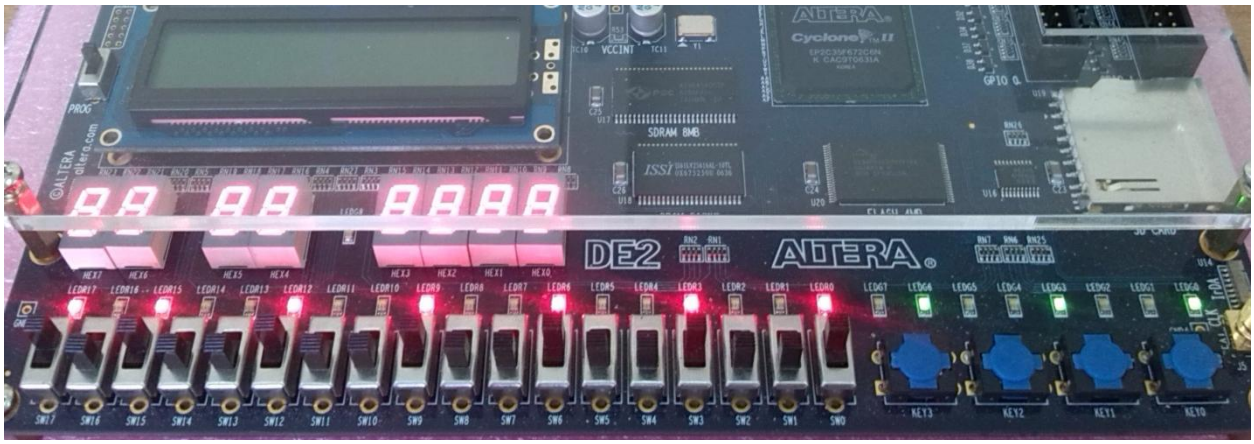
Εικόνα 4.4: Ολοκλήρωση διαμόρφωσης στον Programmer

Αφού έχει γίνει η διαμόρφωση του συστήματος τα δεδομένα μας πηγαίνουν κατευθείαν στο FPGA. Τέλος για να επιβεβαιώσουμε ότι το κύκλωμά μας λειτουργεί σωστά πρέπει να το δοκιμάσουμε δίνοντας κατάλληλες εισόδους 0 ή 1 από τους διακόπτες. Το αποτέλεσμα των εξόδων εμφανίζεται στα κόκκινα και πράσινα leds. Στην Εικόνα 4.5, έχουμε ορίσει την είσοδο επιλογή S = 0 (SW_{17}), $Y_{0,3,6}$= 1 ($SW_{0,3,6}$), $Y_{1,2,4,5,7}$ = 0 ($SW_{1,2,4,5,7}$), $X_{0,2,3,5,6}$ = 0 ($SW_{8,10,11,13,14}$) και $X_{1,4,7}$ = 1 ($SW_{9,12,15}$). Άρα σύμφωνα με την λειτουργία ενός πολυπλέκτη που εξηγήσαμε παραπάνω, όπως διαπιστώνουμε και από την Εικόνα 4.5, θα πρέπει να ανάβουν τα $LEDR_{0,3,6,9,12,15}$ (αφού SW = LEDR) και τα $LEDG_{1,4,7}$, γιατί όταν το S = 0 τότε M = X (M -> LEDG) (βλ. πίνακα αληθείας στο Σχήμα 4.2).



Εικόνα 4.5: Επαλήθευση της λειτουργίας του κυκλώματος μέσω των διακοπτών

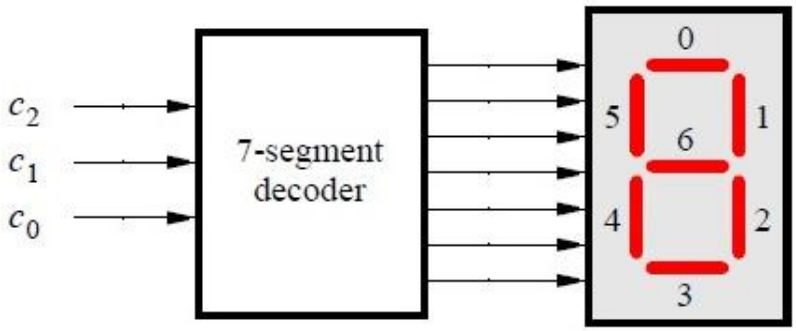
Στην Εικόνα 4.6, έχουμε ορίσει την είσοδο επιλογή S = 1 (SW_{17}) και τους υπόλοιπους διακόπτες όπως πριν: $Y_{0,3,6}$= 1 ($SW_{0,3,6}$), $Y_{1,2,4,5,7}$ = 0 ($SW_{1,2,4,5,7}$), $X_{0,2,3,5,6}$ = 0 ($SW_{8,10,11,13,14}$) και $X_{1,4,7}$ = 1 ($SW_{9,12,15}$). Άρα σύμφωνα με την λειτουργία του πολυπλέκτη, όπως διαπιστώνουμε και από την Εικόνα 4.6, θα πρέπει να ανάβουν τα $LEDR_{0,3,6,9,12,15}$ (αφού SW = LEDR) και τα $LEDG_{0,3,6}$, γιατί όταν το S = 1 τότε M = Y (M -> LEDG).



Εικόνα 4.6: Επαλήθευση της λειτουργίας του κυκλώματος μέσω των διακοπτών

4.1.8 2ο Παράδειγμα Εργαστηριακής Άσκησης στο Quartus II

Σε αυτή την άσκηση θα υλοποιηθεί σε γλώσσα VHDL ένας αποκωδικοποιητής 7 τομέων (*7-segment decoder*), ο οποίος έχει 3-bit είσοδο $c_2c_1c_0$. Αυτός ο αποκωδικοποιητής παράγει επτά εξόδους που χρησιμοποιούνται για την εμφάνιση ενός χαρακτήρα σε μία ένδειξη 7 τομέων (*7-segment*) [Σχήμα 4.3].



Σχήμα 4.3: Ένας αποκωδικοποιητής 7-τομέων [20]

Ο Πίνακας 4.1 παραθέτει τους χαρακτήρες που πρέπει να εμφανίζονται για κάθε συνδυασμό της εισόδου $c_2c_1c_0$. Όπως φαίνεται και στο πίνακα τέσσερις χαρακτήρες θα απεικονίζονται (συν το χαρακτήρα "κενό", ο οποίος επιλέγεται για τους κωδικούς 100 - 111).

$c_2c_1c_0$	Χαρακτήρες
000	H
001	E
010	L
011	O
100	
101	
110	
111	

Πίνακας 4.1: Κώδικες χαρακτήρων

Οι επτά τομείς στην ένδειξη προσδιορίζονται από τους δείκτες 0 έως 6 που φαίνεται στο Σχήμα 4.3. Κάθε τομέας ανάβει οδηγώντας το στην λογική τιμή 0. Η οντότητα VHDL θα υλοποιεί λογικές συναρτήσεις που αντιπροσωπεύουν τα κυκλώματα που χρειάζονται για να ενεργοποιηθεί κάθε ένας από τους επτά τομείς. Αφού δημιουργήσουμε το project της άσκησής μας, επιλέγουμε την δημιουργία νέου vhdl αρχείου. Παρακάτω ακολουθεί ο κώδικας που εισάγουμε μέσα στο αρχείο που δημιουργήσαμε, το οποίο είναι και το top-level αρχείο VHDL του project μας:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY seven_seg IS
      PORT ( SW  : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
             HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6));
END seven_seg;

ARCHITECTURE Behavior OF seven_seg IS
      COMPONENT char_7seg
         PORT (  C          : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
                 Display    : OUT STD_LOGIC_VECTOR(0 TO 6));
      END COMPONENT;
BEGIN
      H0: char_7seg PORT MAP (M, HEX0);
END Behavior;
```

Οι έξοδοι Display ($HEX0_{0\text{-}6}$) είναι *active low,* δηλαδή ενεργοποιούνται όταν οι έξοδοι έχουν λογικό 0. Επομένως ο κώδικας του component char_7seg διαμορφώνεται ως εξής:

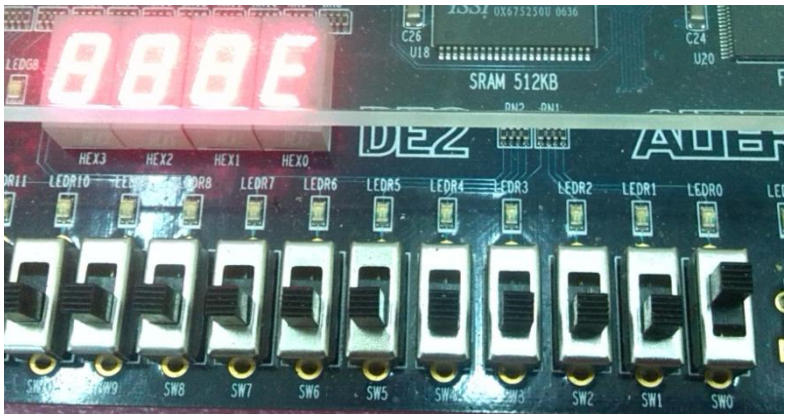
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
      PORT ( C         : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
             Display   : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;
--
--        0
--       ---
--      |   |
--    5|   |1
--      | 6 |
--       ---
--      |   |
--    4|   |2
--      |   |
--       ---
--        3
--
ARCHITECTURE Behavior OF char_7seg IS
BEGIN
      --active low
      WITH C SELECT
      Display <= "1001000" WHEN "000",  -- H
                 "0110000" WHEN "001",  -- E
                 "1110001" WHEN "010",  -- L
                 "0000001" WHEN "011",  -- O
                 "1111111" WHEN OTHERS; --' '
END Behavior;
```

Στον παραπάνω κώδικα, η 7-bit έξοδος HEX0 είναι μία από τις οκτώ απεικονίσεις 7 τομέων της αναπτυξιακής πλακέτας DE2 την όποια αντιστοιχίζουμε με την 7-bit έξοδο Display του Component *char_7seg*. Επίσης χρησιμοποιούμε ως είσοδο τους διακόπτες SW_{0-2} για την εκχώρηση τιμών στο κύκλωμά μας και αντιστοιχούνται με τις εισόδους C_{0-2} των 3 bits του Component *char_7seg*.

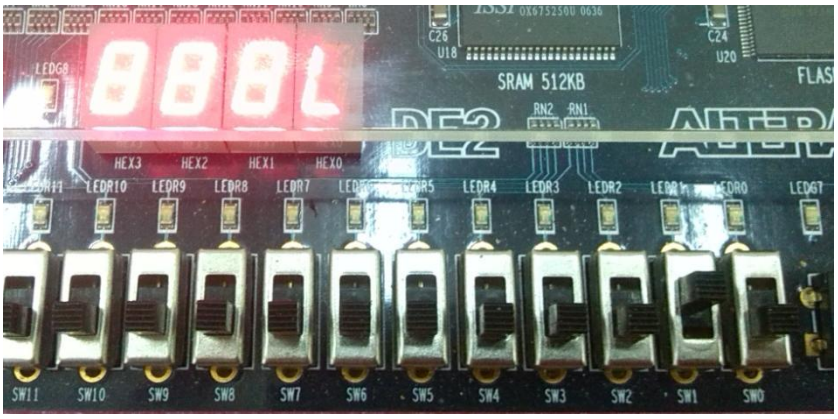
Η λογική συνάρτηση δημιουργήθηκε σύμφωνα με το Σχήμα 4.3 και τον Πίνακα 4.1 γνωρίζοντας ότι ισχύει το active low για τις εξόδους του seven segment. Έτσι για να δημιουργήσουμε τη συνάρτηση θέτουμε 0 όπου ανάβει ο τομέας D_{0-7} για τον χαρακτήρα που επιθυμούμε. Για παράδειγμα, για τον χαρακτήρα Η επιθυμούμε να ανάβουν τα Display $_{1,2,4,5,6}$ [Σχήμα 4.3]. Άρα, τα συγκεκριμένα Display τα θέτουμε 0 και τα υπόλοιπα 1, *1001000.* Ομοίως λειτουργούμε και για τους άλλους χαρακτήρες.

Αφού γράψουμε αυτά τα δύο παραπάνω αρχεία vhdl, ακολουθούν τα βήματα που αναφέραμε στο προηγούμενο παράδειγμα 4.1.7, όπως η εισαγωγή ακροδεκτών, το compilation του project και αφού ολοκληρωθεί με επιτυχία πρέπει να μεταφορτώσουμε το πρόγραμμά μας στην πλακέτα DE2 μέσω του εργαλείου Programmer. Μόλις ολοκληρώσουμε τα παραπάνω βήματα μπορούμε να πάρουμε τα παρακάτω αποτελέσματα, όπως φαίνονται στις Εικόνες 4.7 έως 4.11.

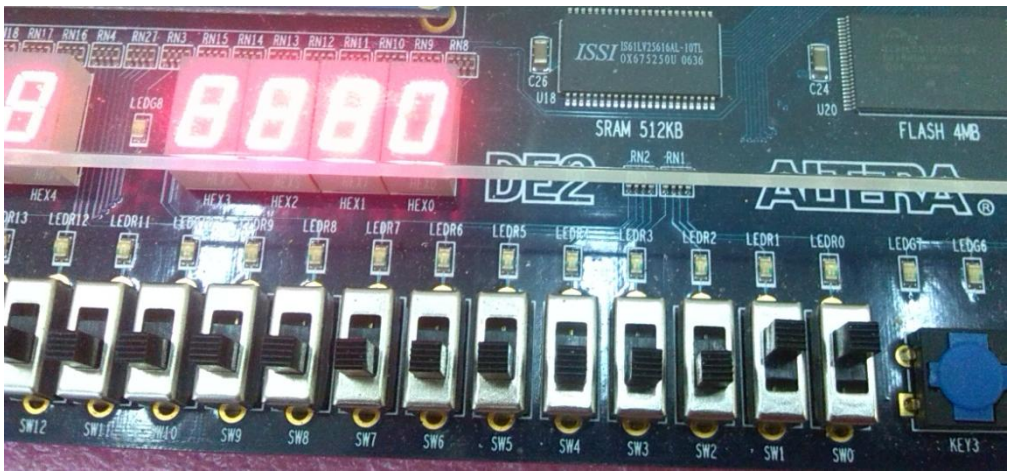
Εικόνα 4.7: Εμφάνιση του χαρακτήρα H, όταν έχουμε ως είσοδο SW0=SW1=SW2=0.



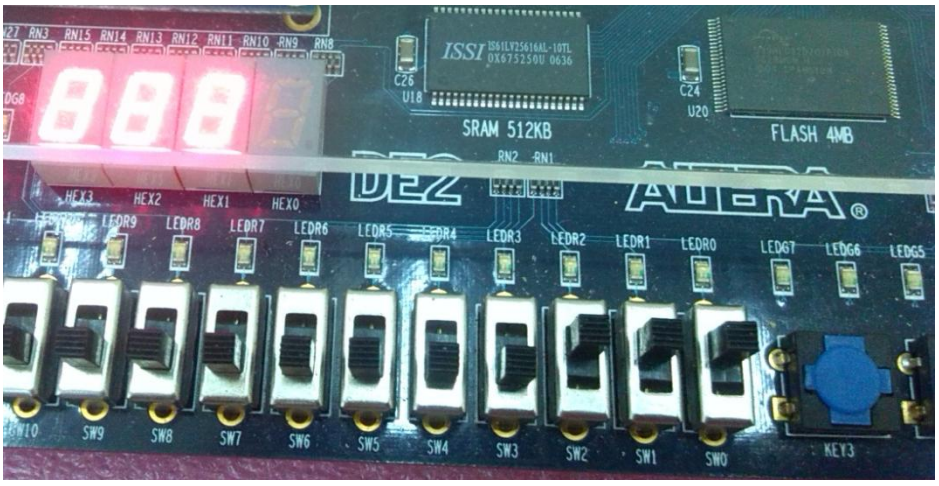
Εικόνα 4.8: Εμφάνιση του χαρακτήρα E, όταν έχουμε ως είσοδο SW0=1, SW1=SW2=0.



Εικόνα 4.9: Εμφάνιση του χαρακτήρα L, όταν έχουμε ως είσοδο SW0=SW2=0, SW1=1.



Εικόνα 4.10: Εμφάνιση του χαρακτήρα Ο, όταν έχουμε ως είσοδο SW0= SW1=1, SW2=0.



Εικόνα 4.11 Εμφάνιση του κενού, όταν έχουμε οτιδήποτε άλλο από τα προηγούμενα.

4.2 SOPC Builder

Ο SOPC Builder είναι ένα ισχυρό εργαλείο για την ανάπτυξη ενός συστήματος. Επιτρέπει να καθοριστεί και να δημιουργηθεί ένα ολοκληρωμένο σύστημα σε προγραμματιζόμενο τσιπ (system-on-a-chip, SOPC) σε πολύ λιγότερο χρόνο από ό, τι με τις παραδοσιακές μεθόδους ολοκλήρωσης. Περιλαμβάνεται ως μέρος του λογισμικού Quartus II.

Ο SOPC Builder είναι ένα εργαλείο γενικής χρήσης για τη δημιουργία συστημάτων που μπορεί ή δεν μπορεί να περιέχουν ένα επεξεργαστή και μπορεί να περιλαμβάνει ένα soft processor και εκτός του επεξεργαστή Nios II. Επίσης αυτοματοποιεί το έργο της ενσωμάτωσης των υποσυστημάτων υλικού. Χρησιμοποιώντας παραδοσιακές μεθόδους σχεδιασμού, θα πρέπει να γράψουμε με

το χέρι HDL ενότητες για να συνδεθούν μεταξύ τους τα κομμάτια του συστήματος. Χρησιμοποιώντας τον SOPC Builder, μπορούμε να καθορίσουμε τα στοιχεία του συστήματος σε ένα γραφικό περιβάλλον (Graphical User Interface – GUI) και ο SOPC Builder δημιουργεί τη λογική διασύνδεση αυτόματα. Δημιουργεί αρχεία HDL που καθορίζουν όλα τα εξαρτήματα του συστήματος, και ένα αρχείο HDL top-level που συνδέει όλα τα υποσυστήματα μεταξύ τους. Ο SOPC Builder παράγει αρχεία περιγραφής υλικού στη γλώσσα που επιθυμούμε, είτε Verilog HDL ή VHDL εξίσου.

Εκτός από το ρόλο του ως εργαλείο παραγωγής συστήματος, ο SOPC Builder παρέχει δυνατότητες για τη διευκόλυνση εγγραφής λογισμικού και την επιτάχυνση της προσομοίωσης του συστήματος.

4.2.1 Αρχιτεκτονική του συστήματος SOPC Builder

Τα **modules** είναι τα δομικά στοιχεία για τη δημιουργία ενός συστήματος SOPC Builder. Χρησιμοποιούν διεπαφές Avalon για τη φυσική διασύνδεση των υποσυστημάτων. Μπορούμε να χρησιμοποιήσουμε τον SOPC Builder για να συνδέσουμε οποιαδήποτε λογική συσκευή (είτε on-chip ή off-chip) που έχει μια διεπαφή Avalon.

4.2.2 Components του SOPC Builder

Η Altera παρέχει έτοιμα προς χρήση Components στο SOPC Builder, όπως οι εξής:

- Μικροεπεξεργαστές, όπως ο επεξεργαστής Nios II
- Περιφερειακά μικροελεγκτή, όπως ο ελεγκτής Scatter-Gather DMA και χρονόμετρο
- Διασυνδέσεις σειριακής επικοινωνίας, όπως ένα UART και μια διασύνδεση σειριακού περιφερειακού (SPI)
- Είσοδοι/έξοδοι (I/O) γενικής χρήσης

- Περιφερειακά επικοινωνιών, όπως το 10/100/1000 Ethernet MAC
- Διεπαφές για off-chip συσκευές

Παρά τα εγκατεστημένα components όπως του εκπαιδευτικού πακέτου της Altera “University Program” αλλά και IPs τρίτων. Επίσης έχουμε την δυνατότητα της συγγραφής των δικών μας components, ώστε να ενσωματώνουν τη λειτουργία και τις διασυνδέσεις που θέλουμε.

Πλέον ο SOPC Builder δεν χρησιμοποιείται για νεότερες αναβαθμίσεις του Quartus II (από την έκδοση 13 και σε νεότερες) και αντί αυτού χρησιμοποιείται το Qsys, το οποίο θα αναλύσουμε παρακάτω.

4.3 Qsys

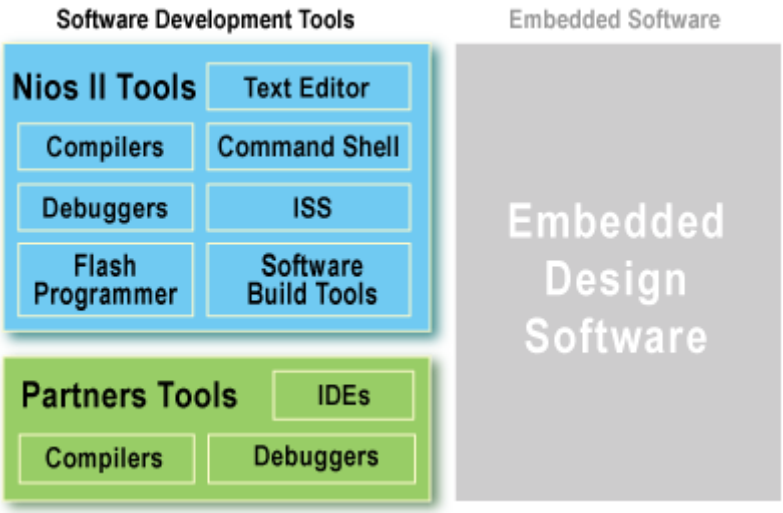
Το εργαλείο Qsys χρησιμοποιείται σε συνδυασμό με το σχεδιαστικό λογισμικό Quartus II. Επιτρέπει στο χρήστη να δημιουργήσει εύκολα ένα σύστημα που βασίζεται στον επεξεργαστή Nios II, με την απλή επιλογή των επιθυμητών λειτουργικών μονάδων και να προσδιορίσει τις παραμέτρους τους.

Το εργαλείο Qsys για την ολοκλήρωση του συστήματος εξοικονομεί σημαντικό χρόνο και προσπάθεια στην διαδικασία του σχεδιασμού του FPGA με την λογική αυτόματης δημιουργίας διασύνδεσης.

Το εργαλείο Qsys είναι η επόμενη γενιά του SOPC Builder, το οποίο τροφοδοτείται από ένα FPGA νέας τεχνολογίας, FPGA-optimized network-on-a-chip (NoC), παρέχοντας υψηλότερες επιδόσεις, βελτιωμένη επαναχρησιμοποίηση σχεδίου, και γρηγορότερη επαλήθευση σε σχέση με το εργαλείο SOPC Builder. Αναλυτικότερα τα πλεονεκτήματα του Qsys σε σχέση με το SOPC Builder είναι τα εξής:

Οφέλη του Qsys	Πλεονεκτήματα του Qsys
Ταχύτερη Ανάπτυξη	• Εύκολο γραφικό περιβάλλον για τον χρήστη επιτρέπει τη γρήγορη ολοκλήρωση μεταξύ των λειτουργιών IP και υποσυστημάτων • Αυτόματη δημιουργία λογικών διασυνδέσεων (συνδέσεις διαύλων διευθύνσεων/δεδομένων, λογική αποκωδικοποίηση διευθύνσεων κ.α. • Διαθεσιμότητα του plug-and-play Qsys-compliant IP από την Altera και τους συνέταιρους IP. • Υποστηρίζει την μίξη των διαφορετικών σε βιομηχανία-πρότυπο διασυνδέσεων, συμπεριλαμβανομένων τις διεπαφές των Avalon, ARM AMBA AXITM, AMBA APBTM, και AMBA AHBTM • Αυτόματη δημιουργία γλώσσας περιγραφής υλικού (HDL) του συστήματος • Ιεραρχική ροή σχεδιασμού επιτρέπει κλιμακούμενα σχέδια, σχεδιασμό με βάση την ομαδοποίηση και μεγιστοποιεί την επαναχρησιμοποίηση του σχεδίου • Ροή αλλαγής προς το Qsys για τα σχέδια του SOPC Builder
Ταχύτερος Χρονισμός	• Διασύνδεση Qsys υψηλής απόδοσης με βάση την αρχιτεκτονική NoC και αυτόματη διοχέτευση που προσφέρει μέχρι και διπλάσια υψηλότερη απόδοση σε σύγκριση με το SOPC Builder
Ταχύτερη Επαλήθευση	• Δυνατότητα να ξεκινήσει την προσομοίωση γρηγορότερα με την αυτόματη δημιουργία δοκιμών και χρησιμοποιώντας τη σουίτα επαλήθευσης IP • Ταχύτερη αναφορά πίνακα με System Console στέλνοντας τις διενέργειες να διαβαστούν και να γραφτούν σε ένα ζωντανό σύστημα

4.4 Nios II EDS



Εικόνα 4.12: Nios II Embedded Design Suite [18]

Μετά από την διαμόρφωση του συστήματος από το Quartus II, για να προγραμματιστεί ο Nios II επεξεργαστής είναι απαραίτητη η συγγραφή κώδικα στο Nios II Embedded Design Suite (Nios II EDS) σε C ή C++ που θα εκτελεστεί από τον επεξεργαστεί.

Όταν ο χρήστης σχεδιάζει με τον ενσωματωμένο επεξεργαστή Nios II, έχει πρόσβαση σε μία σειρά από εξελιγμένα εργαλεία ανάπτυξης λογισμικού που διατίθενται από την εταιρεία Altera και τους συνεργάτες αυτής. Το Nios II EDS είναι ένα ολοκληρωμένο αναπτυξιακό πακέτο για το σχεδιασμό του λογισμικού Nios II. Περιλαμβάνει όχι μόνο τα αναπτυξιακά εργαλεία, αλλά και λογισμικό, drivers συσκευών, βιβλιοθήκη Hardware Abstraction Layer (HAL), ένα εμπορικό λογισμικό ποιότητας του δικτύου και έκδοση αξιολόγησης του λειτουργικού συστήματος σε πραγματικό χρόνο.

Το Nios II EDS περιλαμβάνει:

- Nios II Software Build Tools for Eclipse
- Nios II Software Build Tools
- Ενσωματωμένο Λογισμικό
- Οδηγούς συσκευών για Altera IP και HAL API

4.4.1 Nios II Software Build Tools (SBT) για το Eclipse

Το Nios II Software Build Tools είναι ένα πλήρως ολοκληρωμένο περιβάλλον ανάπτυξης που χτίστηκε από το μηδέν. Χρησιμοποιείται ως βάση και σχεδιάστηκε ως plug-in για τη βιομηχανία-πρότυπο Eclipse.

Το Nios II Software Build Tools για το Eclipse επικεντρώνεται στη βελτίωση της παραγωγικότητας του λογισμικού για μεγάλες εφαρμογές λογισμικού.

Το Nios II Software Build Tools για το Eclipse περιέχει:

- Eclipse IDE
- Source navigator and editor
- Source debugger and profiler
- Compiler, linker, and assembler for C and C++
- Nios II plug-ins for Eclipse
- Nios II Project Manager
- Nios II Software Templates
- Nios II Flash Programmer
- Nios II BSP Editor
- Quartus II Programmer
- Nios II Command Shell

4.4.2 Nios II Software Build Tools

Το Nios II Software Build Tools είναι ένα σύνολο από ισχυρές εντολές, βοηθητικά προγράμματα, καθώς και scripts για την διαχείριση των επιλογών κατασκευής για εφαρμογές, BSPs και βιβλιοθήκες λογισμικού.

4.4.3 Ενσωματωμένο Λογισμικό

Με τον ενσωματωμένο επεξεργαστή Nios II, ο χρήστης έχει πρόσβαση σε ένα ευρύ φάσμα των ενσωματωμένων εξαρτημάτων λογισμικού. Τα παρακάτω στοιχεία περιλαμβάνονται με την Νιος II EDS:

- Λειτουργικό σύστημα πραγματικού χρόνου MicroC/OS-II (η άδεια παραγωγής πωλείται ξεχωριστά)
- NicheStack TCP/IP Network Stack - Nios II Edition (παρέχεται δωρεάν, ως μέρος του Nios II EDS)
- Πρότυπη βιβλιοθήκη Newlib ANSI-C
- Απλό σύστημα αρχείων
- Παραδείγματα σχεδιασμού υλικού και εφαρμογών λογισμικού

Ένας πλήρης κατάλογος από παραδείγματα εφαρμογών, λειτουργικά συστήματα, ενδιάμεσο λογισμικό, και πυρήνες πνευματικής ιδιοκτησίας (IP) είναι διαθέσιμα από την Altera.

4.4.4 Οδηγοί συσκευών για Altera IP και HAL API

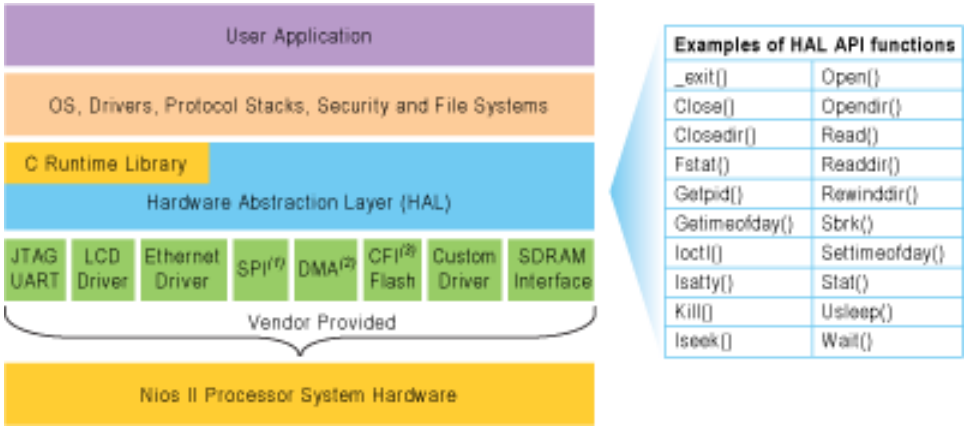
Η εταιρεία Altera παρέχει ένα πλήρες σετ των οδηγών περιφερειακών συσκευών, και ακόμα δημιουργεί αυτόματα ένα πακέτο υποστήριξης προσαρμοσμένου πίνακα για να επισπεύσουν τις προσπάθειες του χρήστη για την ανάπτυξη του λογισμικού του.

- *Επίπεδο αφαίρεσης υλικού (Hardware abstraction layer - HAL)*: Αυξάνει την παραγωγικότητα του χρήστη με την δημιουργία εφαρμογών στην κορυφή του Nios II HAL. Το HAL είναι ένα καλά καθορισμένο επίπεδο λογισμικού που σχηματίζει μία σαφή διάκριση μεταξύ εφαρμογής και λογισμικού (Εικόνα 4.13). Επίσης παρέχει υπηρεσίες όπως descriptors αρχείων (αφηρημένοι δείκτες για την πρόσβαση σε αρχεία), έλεγχος εισόδου/εξόδου (I/O) και προσωρινή μνήμη,

τα οποία απαιτούνται για την λειτουργία της βιβλιοθήκης ANSI C, ώστε ο οδηγός HAL να μην χρειάζεται να παρέχει αυτές τις λειτουργίες.

- *BSP*: Το Nios II Software Build Tools για Eclipse δημιουργεί αυτόματα ένα πλήρες BSP, συμπεριλαμβανομένων των οδηγών για τα περιφερειακά στο σύστημά μας.
- *Custom Drivers*: Ο χρήστης έχει την δυνατότητα να δημιουργεί τους δικούς του custom drivers στην κορυφή του HAL με την αξιοποίηση του HAL API χρησιμοποιώντας μία προκαθορισμένη και καλά τεκμηριωμένη διαδικασία ανάπτυξης του οδηγού. Αυτή η προσέγγιση προάγει συνεκτικό, και επαναχρησιμοποιήσιμο κώδικα.

Εικόνα 4.13: Nios II HAL [18]



(1) SPI: serial peripheral interface (σειριακή περιφεριακή διεπαφή)
(2) DMA: direct memory access (άμεση προσπέλαση μνήμης)
(3) CFI: common flash interface (κοινή διεπαφή flash)

Μετά την συγγραφή και εκσφαλμάτωση του κώδικα γίνεται παραγωγή του εκτελέσιμου το οποίο αφού μεταμορφωθεί στον Nios II αρχίζει η εκτέλεση του.

4.5 Altera Monitor Program

Το Altera Monitor program επιτρέπει στο χρήστη να μεταφράσει (compile), να προγραμματίσει το fpga μεταφορτώνοντας στην πλακέτα το σύστημα που δημιουργήθηκε και την εφαρμογή και να κάνει την εκσφαλμάτωση εύκολα σε προγράμματα γλώσσας assembly αλλά και σε γλώσσας C. Συγκεκριμένα το Monitor Program επιτρέπει ένα χρήστη:

- Να δημιουργήσει ένα project του Nios II, που καθορίζει ένα επιθυμητό σύστημα υλικού και πρόγραμμα λογισμικού.
- Να μεταφορτώσει το σύστημα του Nios II στην πλακέτα με το FPGA.
- Να κάνει Compile προγράμματα λογισμικού, που ορίζονται σε assembly ή C, και να κατεβάζει το προκύπτον κώδικα μηχανής στο σύστημα υλικού του Nios II.
- Να εκτελεί τον Nios II, είτε συνεχόμενα ή βήμα-βήμα.
- Να εξετάζει και να τροποποιεί τα περιεχόμενα των καταχωρητών του Nios II.
- Να εξετάζει και να τροποποιεί τα περιεχόμενα της μνήμης, καθώς και των καταχωρητών που αντιστοιχίζονται με μνήμη σε συσκευές I/O.
- Να ορίζει τα breakpoints που σταματούν την εκτέλεση του προγράμματος σε μία καθορισμένη διεύθυνση.
- Να εκτελεί τερματικό εισόδου/εξόδου μέσω ενός component, το JTAG UART, στο σύστημα του Nios II.
- Να αναπτύσσει προγράμματα του Nios II, που κάνουν χρήση των λειτουργιών του προγράμματος οδήγησης συσκευής που παρέχεται μέσω του Altera's Hardware Abstraction Layer (HAL).

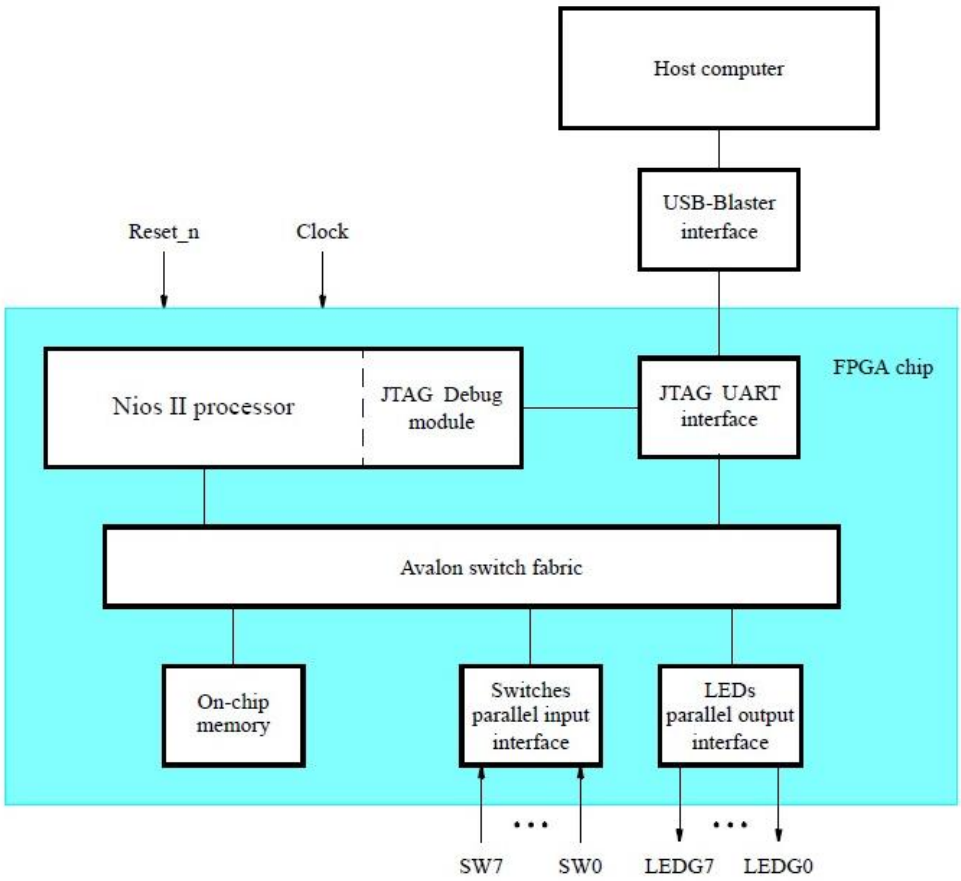
  Η διαδικασία της λήψης και του εντοπισμού σφαλμάτων ενός προγράμματος Nios II απαιτεί την παρουσία μιας πλακέτας με FPGA για την υλοποίηση του συστήματος υλικού Nios II.

Το Altera Monitor Program αποτελεί κυρίως εφαρμογή εκπαιδευτικής φύσεως και είναι διαθέσιμο ως μέρος του **University Program Installer.**

4.6 Παράδειγμα Δημιουργίας Ψηφιακού Συστήματος Nios II

Θα χρησιμοποιήσουμε ένα απλό σύστημα υλικού που φαίνεται στο Σχήμα 4.4. Περιλαμβάνει τον ενσωματωμένο επεξεργαστή Nios II της Altera. Η μονάδα του Nios II μπορεί να συμπεριληφθεί ως μέρος ενός μεγαλύτερου συστήματος, και στη συνέχεια, το σύστημα μπορεί να εφαρμοστεί σε ένα τσιπ Altera FPGA χρησιμοποιώντας το λογισμικό Quartus II.

Όπως φαίνεται στο Σχήμα 4.4, ο επεξεργαστής Nios II είναι συνδεδεμένος με την μνήμη και τις διασυνδέσεις I/O μέσω ενός δικτύου διασύνδεσης που ονομάζεται *Avalon switch fabric*. Το δίκτυο διασύνδεσης δημιουργείται αυτόματα από το εργαλείο Qsys.



Σχήμα 4.4: Ένα απλό παράδειγμα συστήματος του Nios II [19]

Το παραπάνω σύστημα είναι ένα απλό παράδειγμα. Περιέχει 8 ρυθμιστικούς διακόπτες (*switches*) στην πλακέτα DE2, *SW7-0*, που χρησιμοποιούνται για την ενεργοποίηση ή την απενεργοποίηση των οκτώ πράσινων LEDs, LEDG7-0. Οι διακόπτες είναι συνδεδεμένοι με τον επεξεργαστή Nios II μέσω παράλληλης διασύνδεσης I/O, διαμορφωμένη σαν περιφερειακή θύρα εισόδου (PIO). Τα LEDs οδηγούνται από τα σήματα άλλων παράλληλων διασυνδέσεων I/O τα οποία είναι διαμορφωμένα για να λειτουργούν ως θύρα εξόδου. Για να επιτευχθεί η επιθυμητή λειτουργία, το μοτίβο των οκτώ-bit που αντιστοιχεί στην κατάσταση των διακοπτών πρέπει να αποσταλεί στη θύρα εξόδου για την ενεργοποίηση των LEDs. Αυτό θα γίνει έχοντας ο επεξεργαστής Nios II εκτελέσει ένα πρόγραμμα αποθηκευμένο στην

on-chip memory. Η συνεχής λειτουργία απαιτείται, ώστε καθώς οι διακόπτες ανοιγοκλείνουν, τα LEDs να αλλάζουν αντίστοιχα.

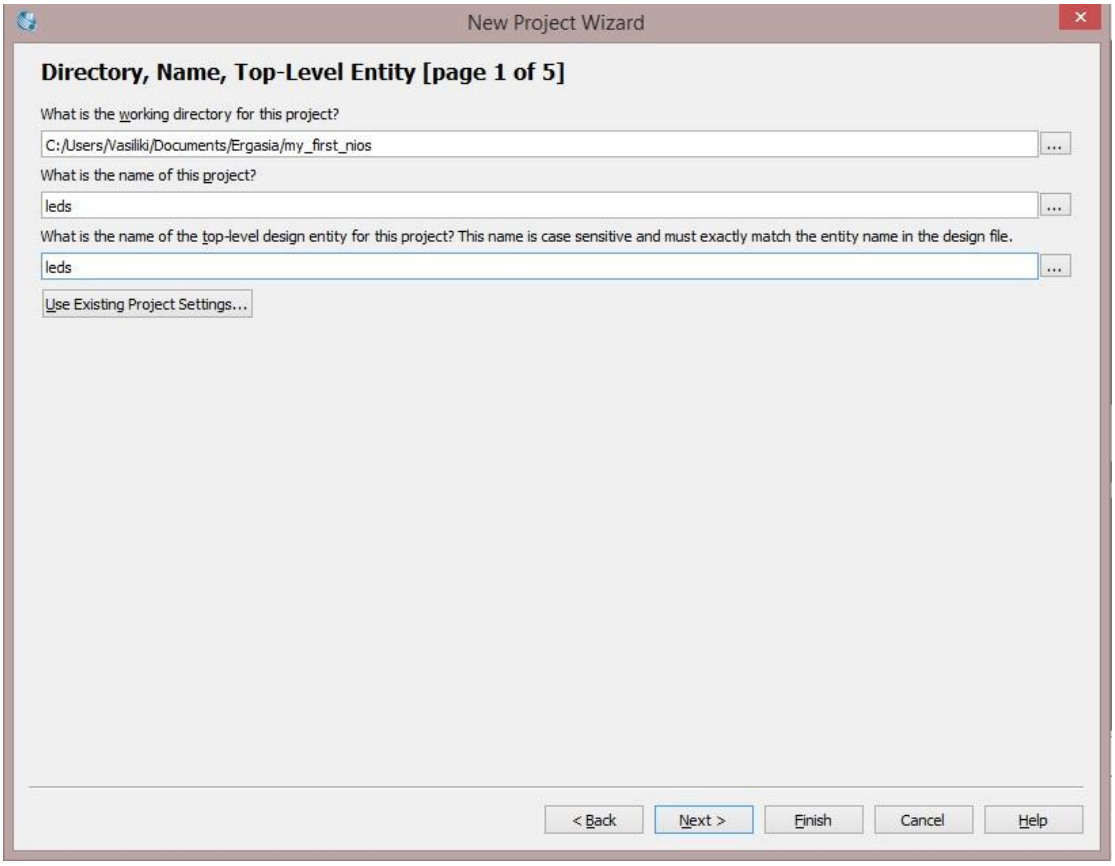
4.6.1 Δημιουργία του συστήματος Nios II μέσω του εργαλείου Qsys

Το εργαλείο Qsys χρησιμοποιείται σε συνδυασμό με το λογισμικό Quartus II. Επιτρέπει στο χρήστη να δημιουργήσει εύκολα ένα σύστημα που βασίζεται στον επεξεργαστή Nios II, με την επιλογή των επιθυμητών λειτουργικών μονάδων και τον καθορισμό των παραμέτρων τους. Για την εφαρμογή του συστήματος στο Σχήμα 4.4, απαιτούνται οι ακόλουθες λειτουργικές μονάδες:

- Επεξεργαστής Nios II
- On-chip memory, η οποία αποτελείται από μπλοκ μνήμης στο chip FPGA. Θα ορίσουμε μία μνήμη 4Kbyte τοποθετημένη σε 32-bit λέξεις (words).
- Δύο παράλληλες διασυνδέσεις I/O
- Διασύνδεση JTAG UART για επικοινωνία με τον κεντρικό υπολογιστή (host computer).

Για να ορίσουμε το επιθυμητό σύστημα, ανοίγουμε το λογισμικό Quartus II και ακολουθούμε τα εξής βήματα:

1. Δημιουργία ενός νέου project στο Quartus II για το σύστημα με την επιλογή ***File>New Project Wizard***, αποθηκεύουμε το project σ' έναν φάκελο που ονομάζεται *my_first_nios*, και γράφουμε το όνομα leds για το project (αυτόματα γράφεται και για το top-level design entity) [Εικόνα 4.14]. Επόμενο βήμα είναι η προσθήκη αρχείων σχεδίασης, όμως στην περίπτωση αυτή θα προστεθούν αργότερα. Στη συνέχεια επιλέγουμε την συσκευή FPGA που χρησιμοποιούμε. Μία λίστα από συσκευές FPGA της σειράς DE δίνεται στον Πίνακα 4.2. συγκεκριμένα στο σύστημα μας χρησιμοποιούμε την πλακέτα DE2 με FPGA το Cyclone II EP2C35F672C6.



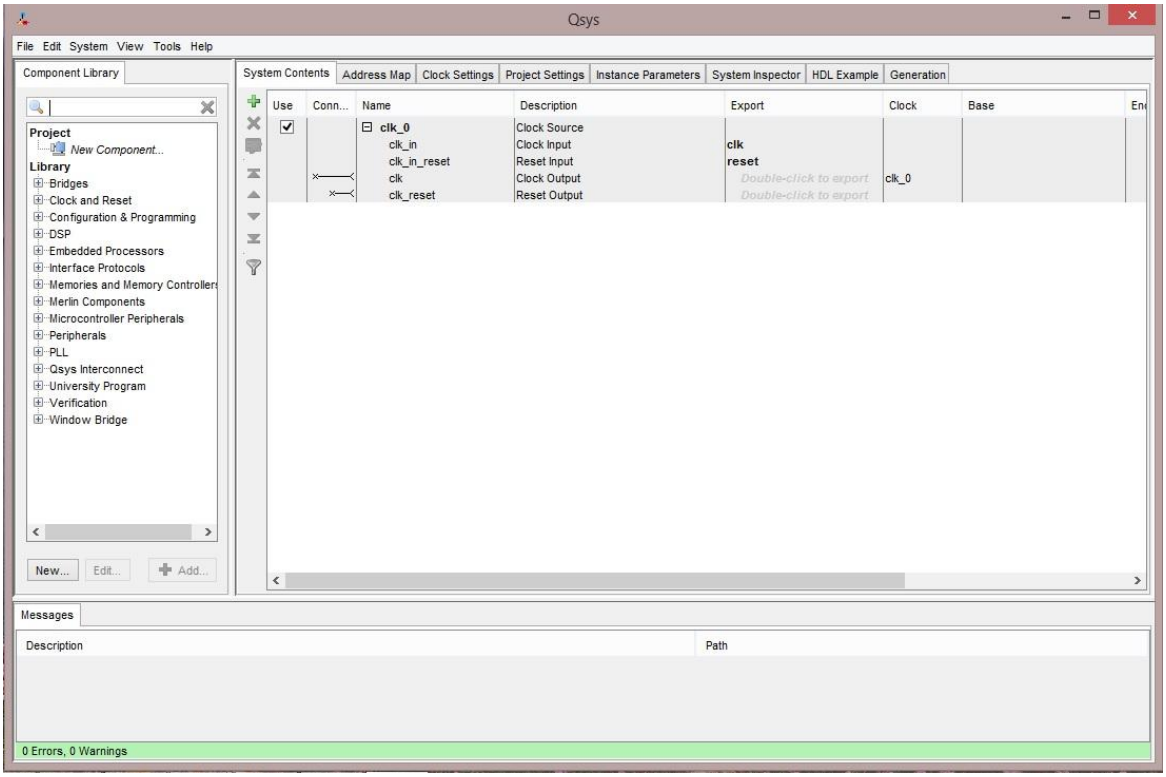
Εικόνα 4.14: Δημιουργία νέου project στο Quartus II

Πλακέτα	Όνομα FPGA
DE0	Cyclone III EP3C16F484C6
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

Πίνακας 4.2: Ονόματα συσκευών FPGA της DE-σειράς [19]

2. Μετά την ολοκλήρωση του New Project Wizard, στο κυρίως παράθυρο του Quartus II επιλέγουμε ***Tools > Qsys*** , το οποίο οδηγεί στην Εικόνα 4.15. Αυτή είναι η καρτέλα περιεχομένων του συστήματος (System Contents) του εργαλείου Qsys, η οποία χρησιμοποιείται για την προσθήκη εξαρτημάτων στο σύστημα και τη ρύθμιση των επιλεγμένων εξαρτημάτων για να πληρούν τις

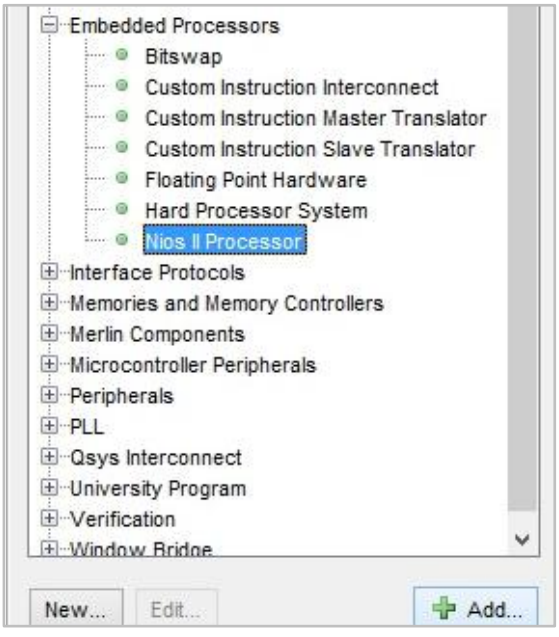
απαιτήσεις του σχεδιασμού. Τα διαθέσιμα εξαρτήματα δίνονται σε λίστα στην αριστερή πλευρά του παραθύρου.



Εικόνα 4.15: Δημιουργία νέου συστήματος Nios II με το εργαλείο Qsys

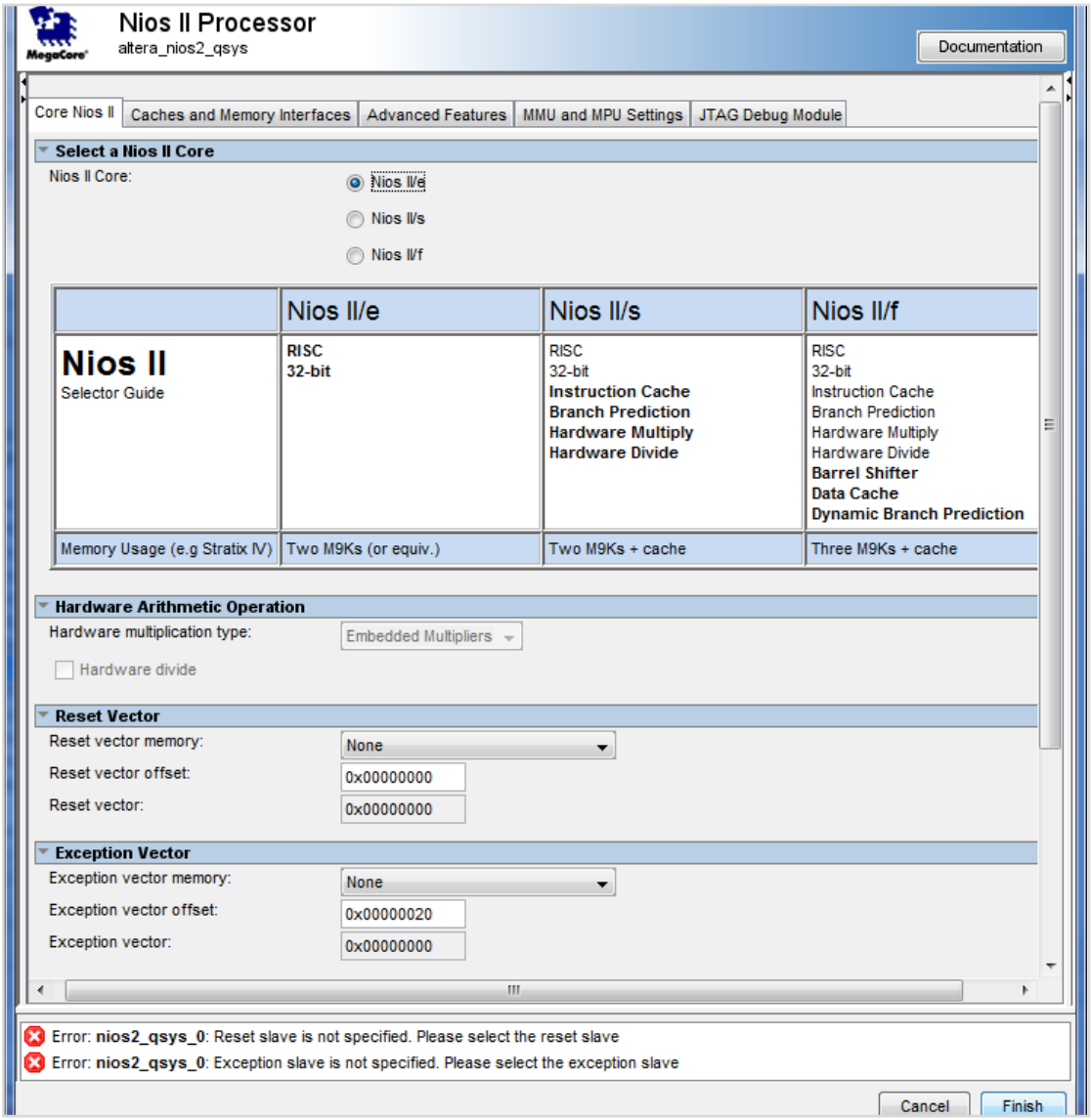
3. Το σύστημα υλικού που θα δημιουργηθεί με τη χρήση του Qsys εκτελείται με τον έλεγχο ενός ρολογιού. Για αυτό το project θα χρησιμοποιήσουμε ρολόι των 50-MHz που παρέχεται στον πίνακα DE2. Επιλέγουμε την καρτέλα **Clock Settings** (Ρυθμίσεις ρολογιού). Εδώ, είναι δυνατόν ο καθορισμός των ονομάτων και της συχνότητας των σημάτων χρονισμού που χρησιμοποιούνται στο project. Επιστροφή στην καρτέλα System Contents.

4. Καθορισμός του επεξεργαστή Nios II:
   - Στην αριστερή πλευρά του παραθύρου Qsys επεκτείνουμε την κατηγορία **Embedded Processors**, επιλέγουμε τον **Nios II Processor** και πατώντας το Add [Εικόνα 4.16], εμφανίζεται το παράθυρο στην Εικόνα 4.17.

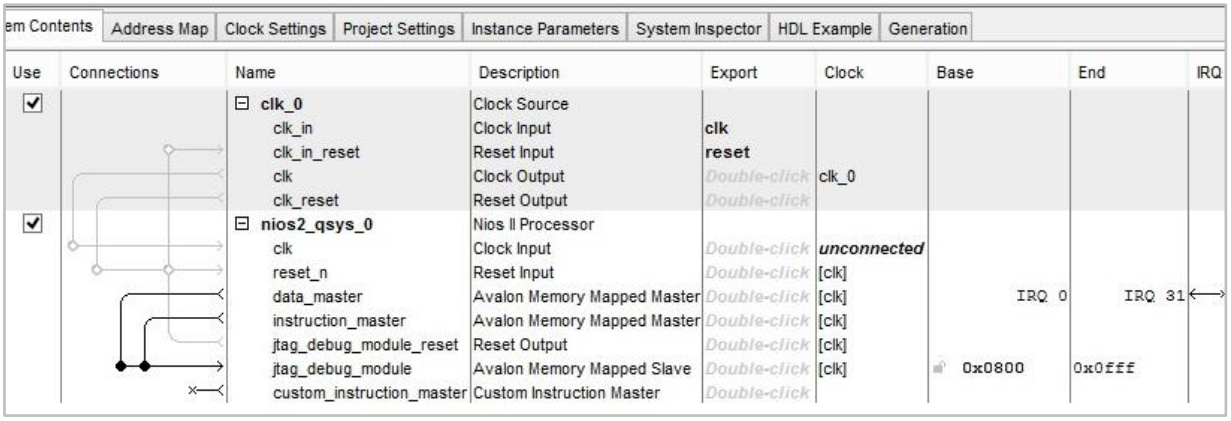


Εικόνα 4.16: Προσθήκη του επεξεργαστή Nios II από την καρτέλα Component Library.

- Επιλέγουμε τον **Nios II/e,** που είναι η οικονομική έκδοση του επεξεργαστή [Εικόνα 4.17]. Αυτή η έκδοση είναι διαθέσιμη για χρήση με δωρεάν άδεια. Ο Nios II έχει εισόδους *reset* και *interrupt*. Όταν μία από αυτές είναι ενεργοποιημένη, ο επεξεργαστής ξεκινά την εκτέλεση των εντολών που αποθηκεύονται στις διευθύνσεις μνήμης, γνωστή ως *reset vector* και *interrupt vector,* αντίστοιχα. Από τη στιγμή που δεν έχουν ακόμη περιληφθεί κανένα από τα εξαρτήματα της μνήμης στο σχεδιασμό μας, το Qsys θα εμφανίσει μηνύματα λάθους. Γι' αυτό αγνοούμε αυτά τα μηνύματα και πατώντας το Finish εμφανίζεται το κύριο παράθυρο του Qsys, το οποίο δείχνει τώρα και τον επεξεργαστή Nios II [Εικόνα 4.18].

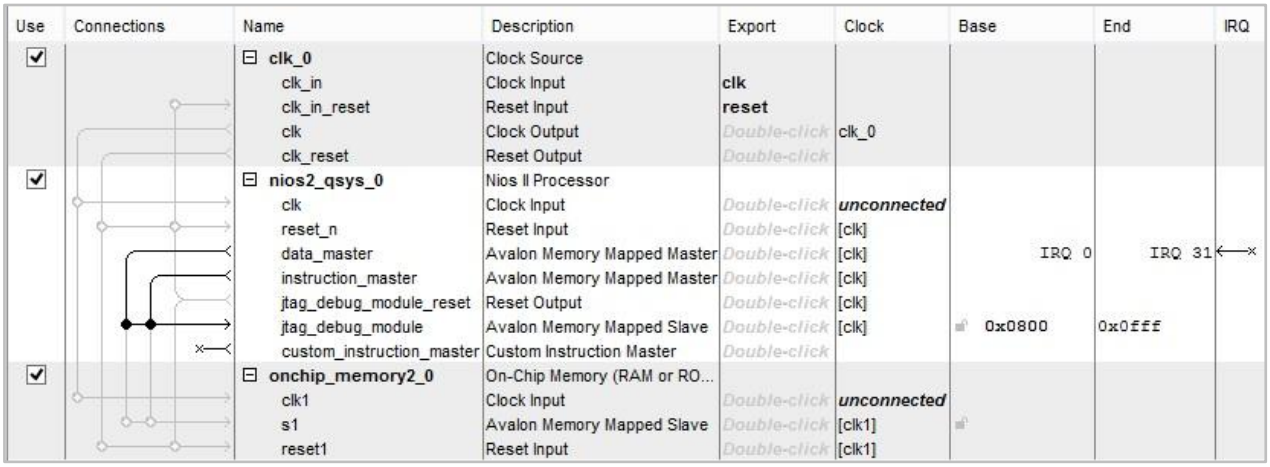


Εικόνα 4.17: Δημιουργία του επεξεργαστή Nios II



Εικόνα 4.18: Ένταξη του επεξεργαστή Nios II στο σχέδιο

5. Για τον καθορισμό της on-chip memory εκτελούνται τα παρακάτω:
   - Επεκτείνουμε στα αριστερά την κατηγορία **Memories and Memory Controllers,** επεκτείνουμε **On-Chip** και επιλέγουμε **On-Chip Memory (RAM or ROM)**, και πατάμε Add.
   - Στο παράθυρο διαμόρφωσης της On-Chip Memory που εμφανίζεται, το **Data Width ορίζεται στα 32 bits** και το **Total memory στα 4Kbytes** (4096 bytes).
   - Δεν χρειάζεται καμία άλλη αλλαγή στις προεπιλεγμένες ρυθμίσεις. Πατώντας το Finish εμφανίζεται το κύριο παράθυρο του Qsys όπως στην Εικόνα 4.19.

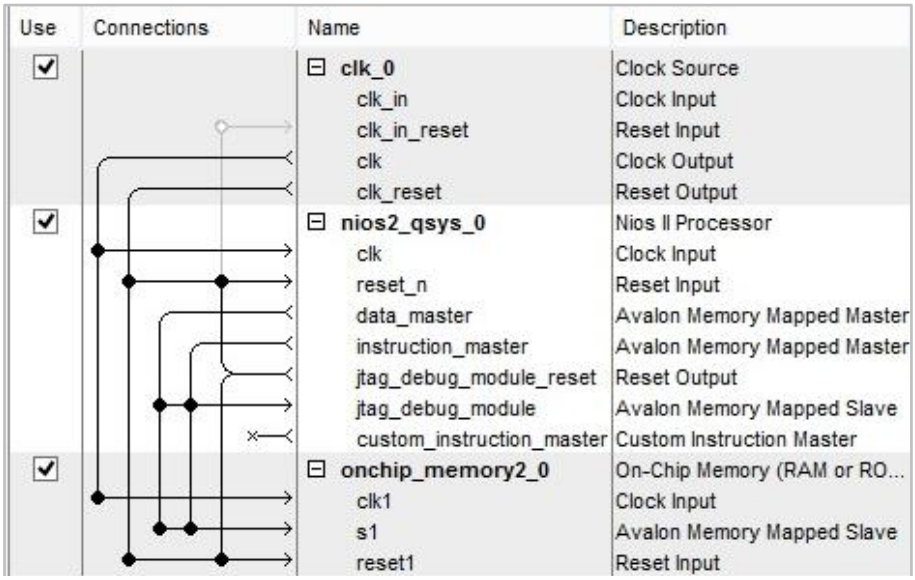


Εικόνα 4.19: Η On-chip memory συμπεριλαμβάνεται στην πλακέτα DE2.

6. Ενώ ο επεξεργαστής Nios II και η on-chip memory έχουν συμπεριληφθεί στο σχέδιο, δεν έχει γίνει καμία σύνδεση μεταξύ των εξαρτημάτων. Στην Εικόνα 4.19, οι συνδέσεις που ήδη έχουν γίνει σημειώνονται με μαύρους γεμάτους κύκλους και οι άλλες πιθανές συνδέσεις με άδειους κύκλους. Κάνοντας κλικ πάνω σε έναν άδειο κύκλο δημιουργείται μία σύνδεση, ενώ κάνοντας κλικ σε ένα γεμάτο κύκλο καταργείται η αντίστοιχη σύνδεση. Δημιουργούμε τις παρακάτω συνδέσεις:

- Οι είσοδοι ρολογιού του επεξεργαστή και της μνήμης στην έξοδο του ρολογιού του εξαρτήματος Clock.
- Οι είσοδοι *reset* του επεξεργαστή και της μνήμης στην έξοδο *reset* του εξαρτήματος του ρολογιού και την έξοδο *jtag_debug_module_reset.*
- Η είσοδος *s1* της μνήμης στις εξόδους *data_master* και *instruction_master* του επεξεργαστή.

Η παραπάνω συνδέσεις φαίνονται στην Εικόνα 4.20.

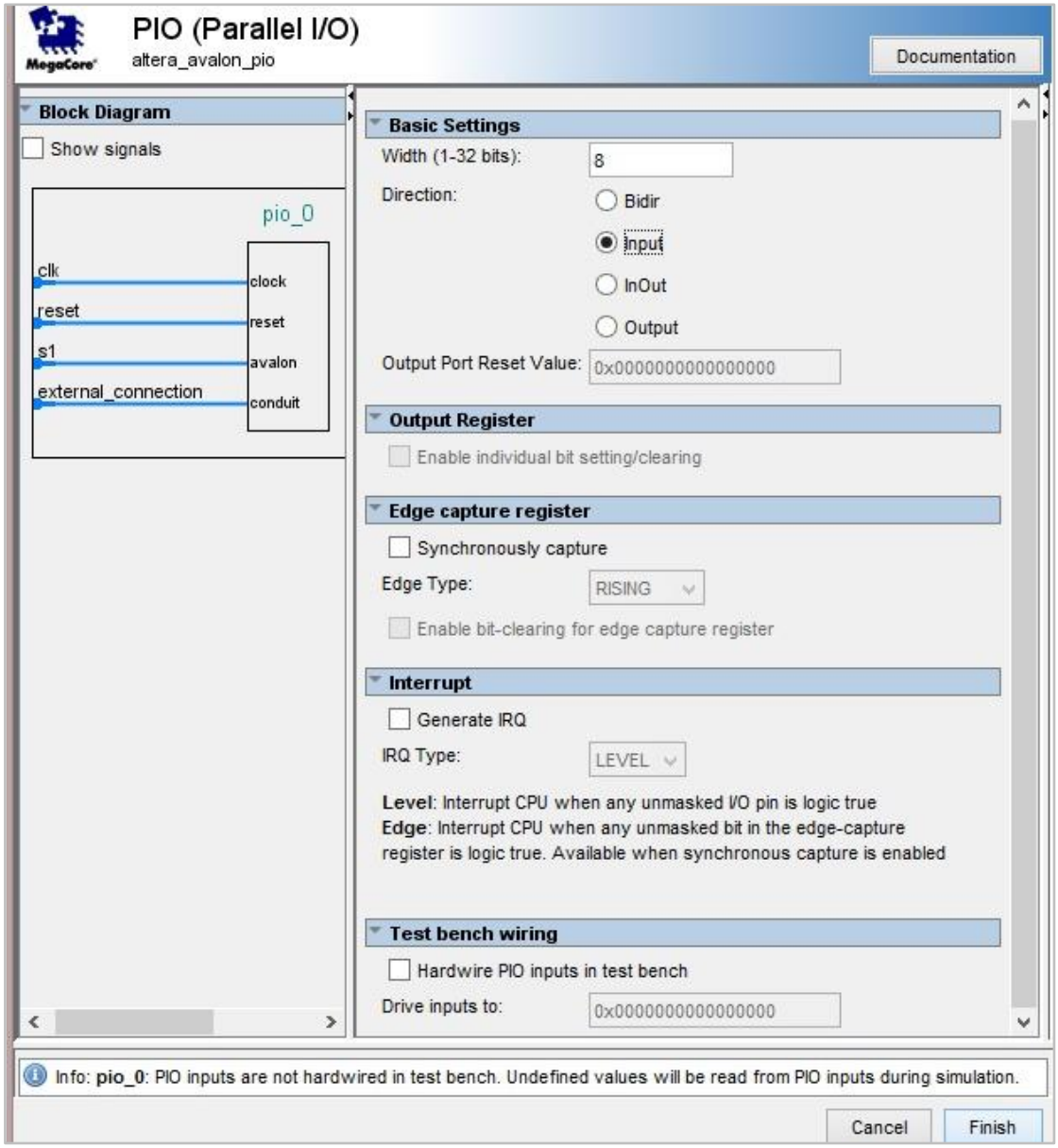


Εικόνα 4.20: Οι συνδέσεις που ισχύουν πλέον στο σύστημά μας

7. Καθορισμός της εισόδου παράλληλης διασύνδεσης:

- Επιλέγουμε **Peripherals > Microcontroller Peripherals > PIO (Parallel I/O)** και κάνοντας κλικ στο Add εμφανίζεται το παράθυρο διαμόρφωσης του PIO [Εικόνα 4.21].
- Το πλάτος της θύρας (**Width**) ορίζεται στα **8bits** και επιλέγουμε την κατεύθυνση (**Direction**) της θύρας ως είσοδος **(Input).**

- Κάνουμε κλικ στο Finish.



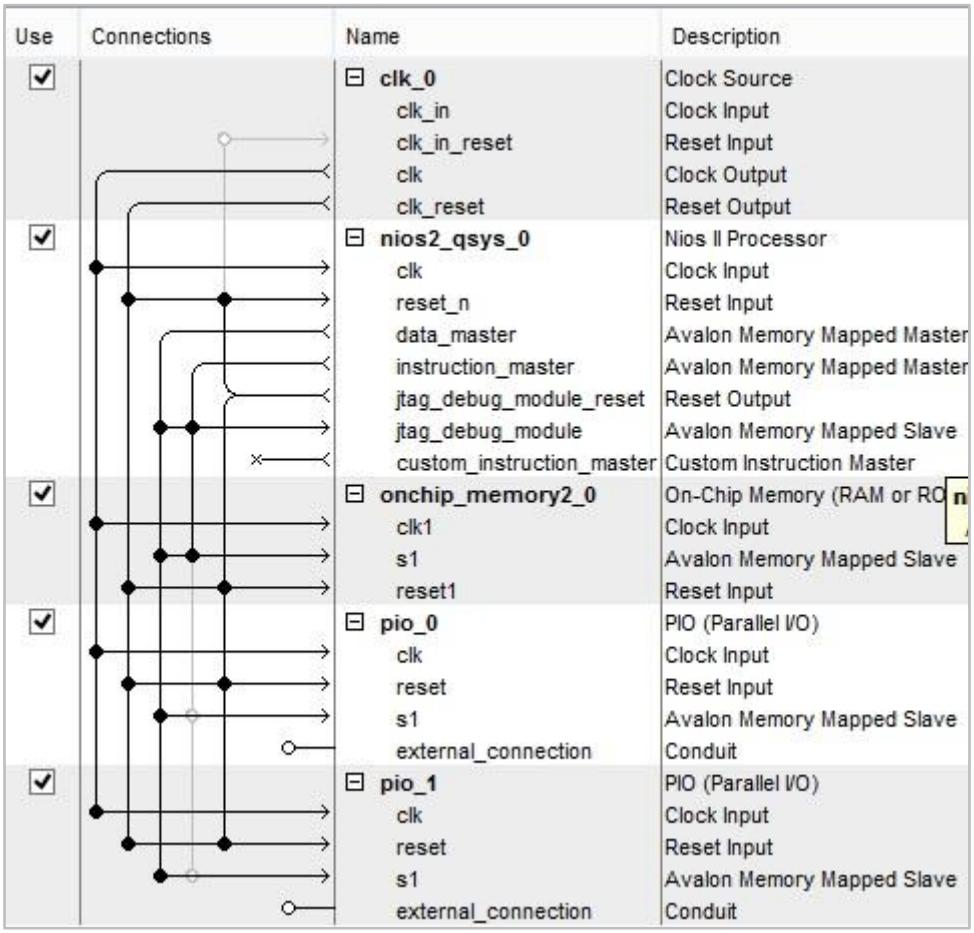
Εικόνα 4.21: Ορισμός μιας παράλληλης διασύνδεσης εισόδου.

8. Με τον ίδιο τρόπο ορίζεται και η έξοδος παράλληλης διασύνδεσης μόνο που στην κατηγορία Direction επιλέγουμε **Output**.

9. Καθορισμός των απαραίτητων συνδέσεων για τις δύο PIOs:

- Οι είσοδοι ρολογιού των PIOs στην έξοδο ρολογιού του ρολογιού.
- Οι είσοδοι *reset* των PIOs στην έξοδο reset του ρολογιού και στην έξοδο *jtag_debug_module_reset*.
- Οι είσοδοι *s1* των PIOs στην έξοδο *data_master* του επεξεργαστή.

Τα αποτελέσματα των συνδέσεων φαίνονται στην Εικόνα 4.22.

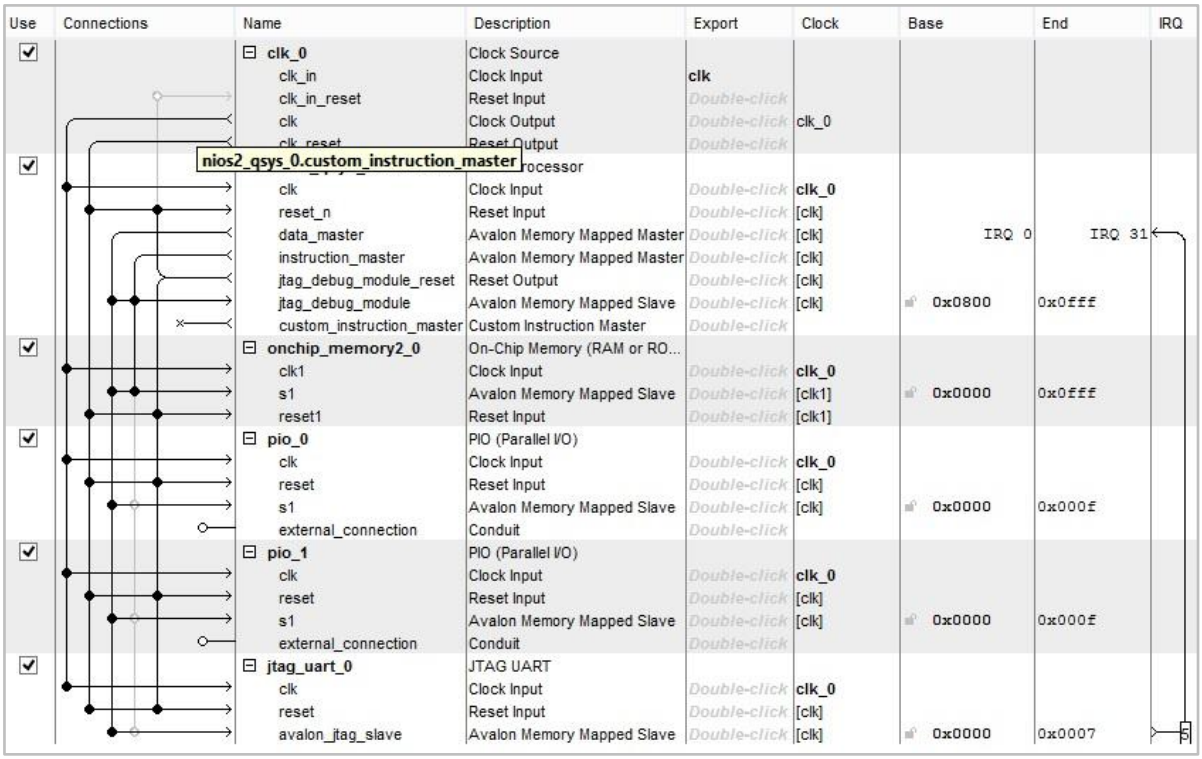


Εικόνα 4.22: Το σύστημα με όλα τα εξαρτήματα και τις συνδέσεις.

10. Θέλουμε την σύνδεση σε έναν κεντρικό υπολογιστή (host computer) και την παροχή ενός μέσου για επικοινωνία μεταξύ του συστήματος Nios II και του host computer. Αυτό μπορεί να επιτευχθεί με διασύνδεση *JTAG UART* ορίζοντάς την ως εξής:

- Επιλέγουμε Interface **Interface Protocols > Serial > JTAG UART** και πατώντας το Add φθάνουμε στο παράθυρο διαμόρφωσης του JTAG UART.
- Δεν χρειάζεται καμία αλλαγή στις προεπιλεγμένες ρυθμίσεις, επομένως πατώντας το Finish επιστρέφουμε στο κυρίως παράθυρο του Qsys.

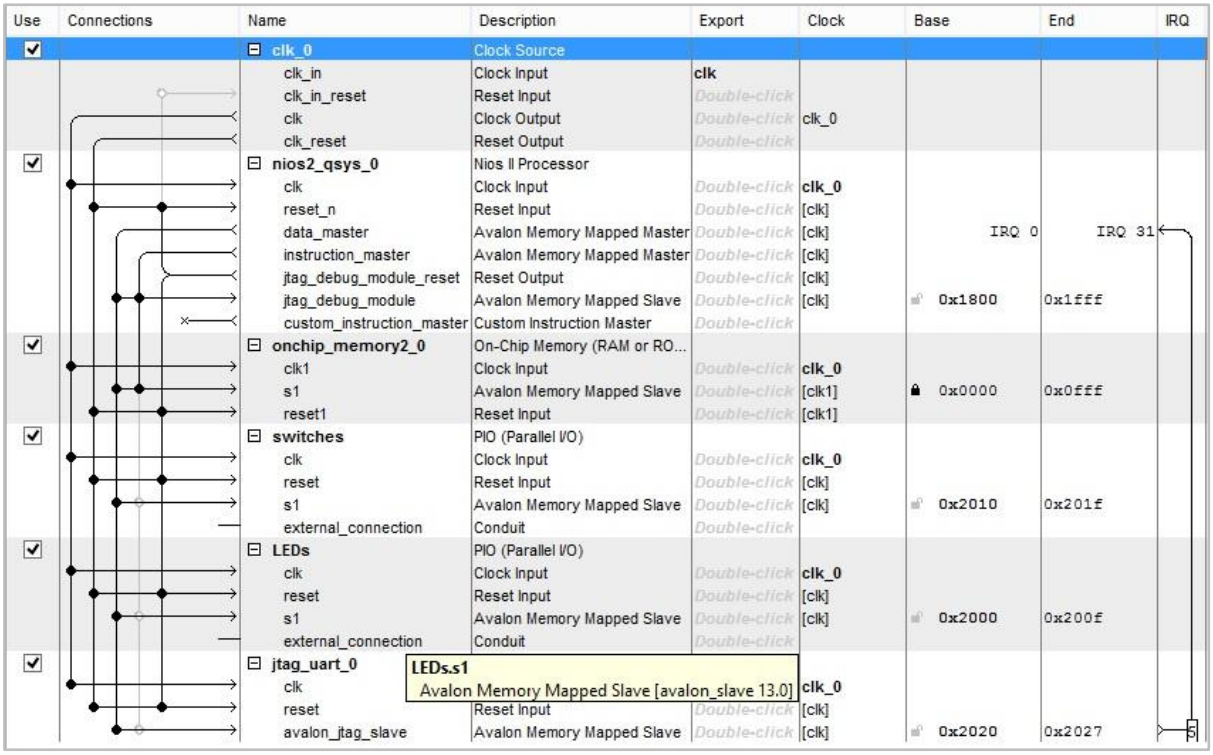
Συνδέουμε το JTAG UART με το ρολόι, τις θύρες reset και data-master, όπως και με τις PIOs. Συνδέουμε την γραμμή αίτησης διακοπής (IRQ) από το JTAG UART στον επεξεργαστή Nios II επιλέγοντας την σύνδεση κάτω από την στήλη IRQ [Εικόνα 4.23]. Μόλις γίνει η σύνδεση, ένα πλαίσιο με τον αριθμό 0 εμφανίζεται στην σύνδεση. Ο επεξεργαστής Nios II έχει 32 θύρες διακοπής που κυμαίνονται από το 0 έως το 31, και ο αριθμός σε αυτό το πλαίσιο επιλέγει σε ποια θύρα θα χρησιμοποιηθεί για αυτό το IRQ. Κάνοντας κλικ στο πλαίσιο, το αλλάζουμε για την χρήση της θύρας 5.



Εικόνα 4.23: Σύνδεση της γραμμής IRQ από το JTAG UART στον επεξεργαστή Nios II.

11. Το εργαλείο Qsys επιλέγει αυτόματα ονόματα για τα διάφορα εξαρτήματα. Τα ονόματα δεν είναι απαραίτητα αρκετά περιγραφικά, αλλά μπορούν να αλλάξουν. Στο Σχήμα 4.4, χρησιμοποιούμε τα ονόματα Switches και LEDs για τις παράλληλες διασυνδέσεις I/O, αντίστοιχα. Αυτά τα ονόματα μπορούν να χρησιμοποιηθούν στο σύστημα. Κάνουμε δεξί κλικ στο όνομα pio_0 και, στη συνέχεια, επιλέγουμε Rename (Μετονομασία). Αλλάζουμε το όνομα σε switches. Ομοίως, αλλάξτε pio_1 για LEDs. Η Εικόνα 4.24 δείχνει το σύστημα με τις αλλαγές στα ονόματα που κάναμε για όλα τα εξαρτήματα.

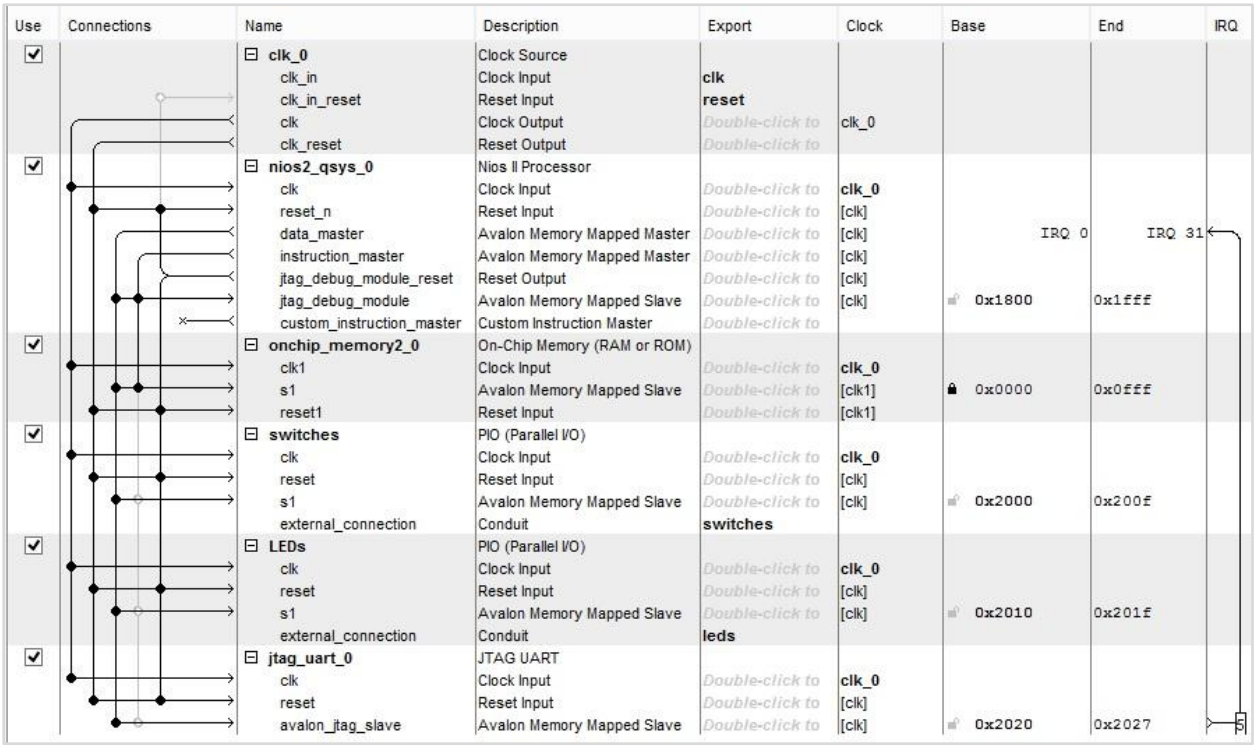
12. Παρατηρούμε ότι οι διευθύνσεις βάσης (Base) και τέλους (End) των διάφορων εξαρτημάτων στο σύστημα δεν έχουν εκχωρηθεί σωστά. Αυτές οι διευθύνσεις μπορούν να εκχωρηθούν από τον χρήστη ή αυτόματα από το εργαλείο Qsys. Εδώ θα χρησιμοποιήσουμε την τελευταία πιθανότητα, ωστόσο βεβαιωνόμαστε ότι η μνήμη on-chip έχει διεύθυνση βάσης το μηδέν. Κάνουμε διπλό κλικ στη διεύθυνση βάσης για την on-chip memory στο παράθυρο του Qsys και εισάγουμε την διεύθυνση 0x00000000. Μετά κλειδώνουμε την διεύθυνση κάνοντας κλικ στο παρακείμενο λουκέτο [Εικόνα 4.24]. Το Qsys θα εκχωρήσει τις υπόλοιπες διευθύνσεις επιλέγοντας **System > Assign Base Addresses**.



Εικόνα 4.24: Το σύστημα με κατάλληλα ονόματα σε όλα τα εξαρτήματα και με τις εκχωρημένες διευθύνσεις.

13. Η συμπεριφορά του επεξεργαστή Nios II όταν γίνεται επανεκκίνηση καθορίζεται από το *reset vector*. Είναι η θέση της μνήμης από την οποία ο επεξεργαστής παίρνει την επόμενη εντολή όταν κάνει επανεκκίνηση. Ομοίως, ο *exception vector* είναι η διεύθυνση μνήμης της εντολής που ο επεξεργαστής εκτελεί όταν συμβαίνει μία διακοπή. Για τον καθορισμό των δυο αυτών παραμέτρων εκτελούνται τα παρακάτω:

- Δεξί κλικ στο *nios2_qsys_0* και επιλέγοντας το Edit (επεξεργασία) φθάνουμε στο παράθυρο επεξεργασίας του επεξεργαστή Nios II. [Εικόνα 4.17]
- Επιλέγουμε *onchip_memory* να είναι η συσκευή μνήμης για τους *reset* και *exception vectors*.
- Δεν κάνουμε άλλη αλλαγή στις προεπιλεγμένες ρυθμίσεις.
- Παρατηρούμε ότι τα λάθη που εμφανίζονταν μέχρι τώρα δεν υπάρχουν τώρα, και πατάμε το Finish για να επιστρέψουμε στο κύριο παράθυρο του Qsys.

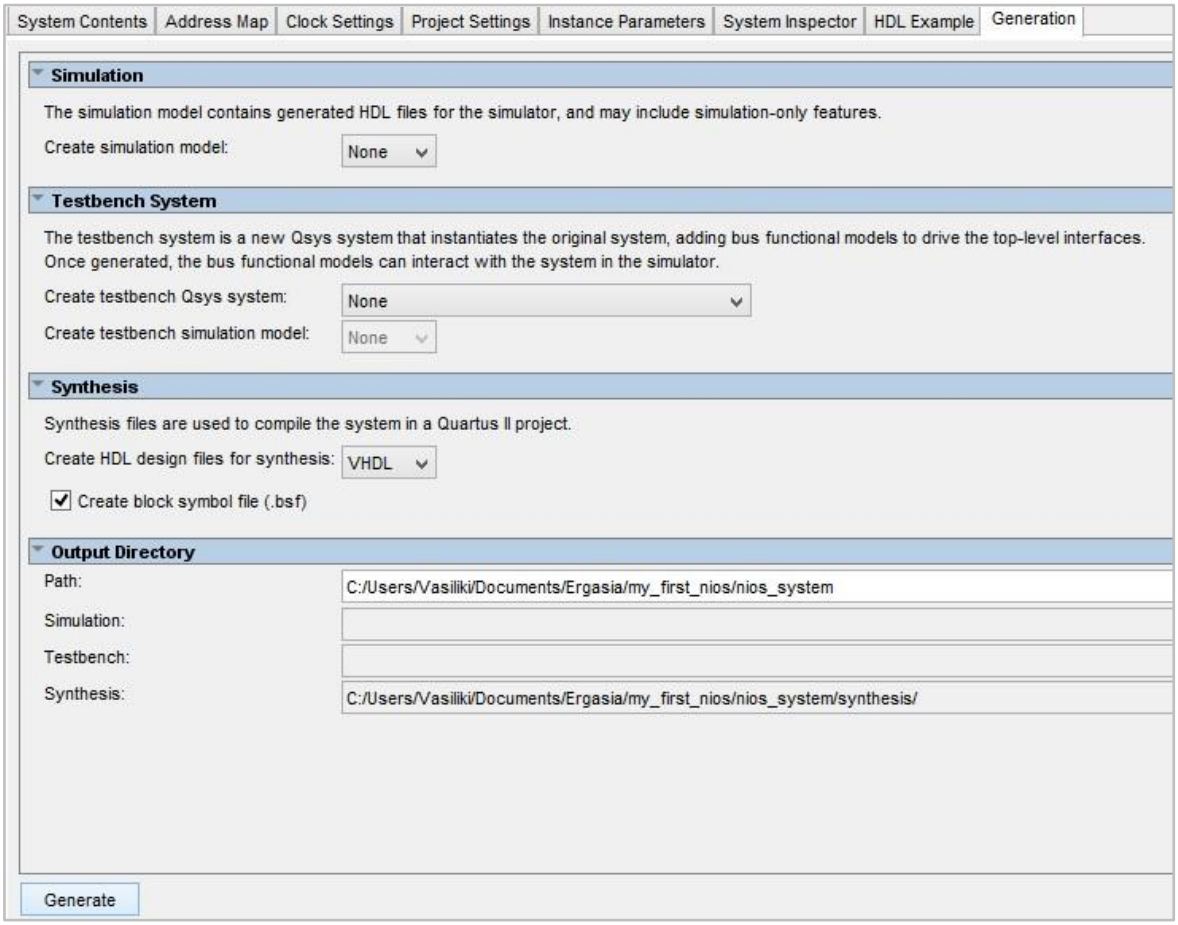


Εικόνα 4.25: Το ολοκληρωμένο σύστημα

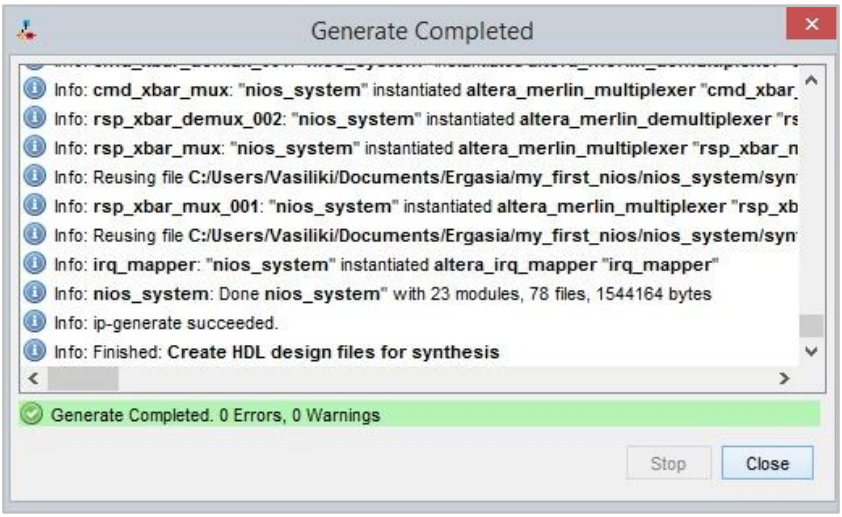
14. Μέχρι τώρα έχουμε καθορίσει όλες τις συνδέσεις μέσα στο κύκλωμα του *nios_system*. Επίσης είναι απαραίτητο τις συνδέσεις για τα εξωτερικά εξαρτήματα (components), τα οποία είναι οι διακόπτες (switches) και τα LEDs στην περίπτωσή μας. Για να επιτευχθεί το παραπάνω, κάνουμε διπλό κλικ στο Double-click (στη στήλη Export της καρτέλας System Contents) στο external_connection του switches, και το ονομάζουμε *switches*. Ομοίως, κάνουμε την εξωτερική σύνδεση για τα LEDs και την ονομάζουμε *leds*. Επίσης, αν δεν έχει ήδη δημιουργηθεί το *reset* στην στήλη Export,κάνουμε διπλό κλικ

στο Double-click του *clk_in_reset* και το ονομάζουμε *reset*. Με αυτές τις συνδέσεις τελειώνει ο καθορισμός του συστήματός μας. [Εικόνα 4.25]

15. Πριν περάσουμε στην δημιουργία του συστήματος (Generation) ελέγχουμε τις καταχωρημένες διευθύνσεις από το Qsys [Εικόνα 4.25]. Βλέπουμε ότι οι διευθύνσεις των παράλληλων διασυνδέσεων switches και leds είναι 0x00002000 και 0x00002010 αντίστοιχα, τις οποίες και θα χρειαστούμε στο πρόγραμμα της εφαρμογής του Nios II επεξεργαστή. Έπειτα έχοντας καθορίσει όλα τα εξαρτήματα που χρειάζονται στο επιθυμητό σύστημα, τώρα μπορεί να δημιουργηθεί. Σώζουμε το καθορισμένο σύστημα με το όνομα ***nios_system***. Στη συνέχεια, επιλέγουμε την καρτέλα **Generation** [Εικόνα 4.26]. Επιλέγουμε το *None* στα πλαίσια *Create simulation model* και *Create testbench Qsys system*, επειδή σε αυτό το παράδειγμα δεν θα κάνουμε προσομοίωση υλικού. Πατάμε το Generate στο κάτω μέρος του παράθυρου. Όταν ολοκληρωθεί με επιτυχία, η διαδικασία παραγωγής δίνει το μήνυμα "Generate Completed". [Εικόνα 4.27]



Εικόνα 4.26: Δημιουργία του συστήματος (Generation)



Εικόνα 4.27: Ολοκλήρωση της διαδικασίας παραγωγής συστήματος.

4.6.2 Ένταξη του Συστήματος Nios II σε ένα Project του Quartus II

Για να ολοκληρωθεί το σχέδιο υλικού (hardware), θα πρέπει να γίνουν τα ακόλουθα:

- Ενσωμάτωση της μονάδας (module) που δημιουργήθηκε από το Qsys στο project του Quartus II.
- Εκχώρηση των ακίδων (*pins*) του FPGA
- Μεταγλώττιση (Compilation) του σχεδιασμένου κυκλώματος.
- Προγραμματισμός και διαμόρφωση του FPGA στην πλακέτα DE2.

Το εργαλείο Qsys μετά την διαδικασία της παραγωγής, δημιουργεί ένα Verilog ή VHDL module (ανάλογα τι επιλέξαμε στην καρτέλα Generation) το οποίο καθορίζει το επιθυμητό σύστημα Nios II. Στην περίπτωσή μας, αυτό το module δημιουργείται στο αρχείο *nios_system.vhd*, επειδή επιλέξαμε VHDL, το οποίο μπορεί να βρεθεί στον φάκελο του project *my_first_nios/nios_system/synthesis.*

Κανονικά, η μονάδα Nios II που παράγεται από το εργαλείο Qsys είναι πιθανό να είναι ένα μέρος ενός μεγαλύτερου σχεδίου. Ωστόσο, στο απλό παράδειγμά μας δεν χρειάζεται άλλο κύκλωμα. Το μόνο που χρειάζεται να κάνουμε είναι η ενσωμάτωση του συστήματος Nios II στο **top-level Verilog ή VHDL module**, και η

σύνδεση των εισόδων και εξόδων των παράλληλων θυρών I/O, καθώς και τη σύνδεση του ρολογιού και των reset εισόδων, στα κατάλληλα pins του FPGA.

Ο κώδικας γλώσσας VHDL στο αρχείο *nios_system.vhd* είναι αρκετά μεγάλος. Ο παρακάτω κώδικας είναι μέρος αυτού του αρχείου που καθορίζει τις θύρες εισόδου και εξόδου του συστήματος. Η 8-bit είσοδος της παράλληλης θύρας switches ονομάζεται *switches_export*. Η 8-bit έξοδος ονομάζεται *leds_export*. Το ρολόι και τα σήματα reset ονομάζονται *clk_clk* και *reset_reset_n*, αντίστοιχα. Παρατηρούμε ότι το σήμα reset προστέθηκε αυτόματα από το Qsys.

```
entity nios_system is

      port (
        clk_clk           : in   std_logic    := '0';              --clk.clk
        leds_export       : out  std_logic_vector(7 downto 0);     --leds.export
        switches_export   : in   std_logic_vector(7 downto 0)
                                              := (others => '0'); --switches.export
        reset_reset_n     : in   std_logic    := '0'               -- reset.reset_n
      );
end entity nios_system;
```

Το module nios_system πρέπει να ενσωματωθεί σε ένα top-level module που πρέπει να ονομαστεί *leds* [**Εικόνα 4.14**], γιατί αυτό είναι το όνομα που ορίσαμε για την top-level οντότητα υψηλού επιπέδου στο project του Quartus II. Για τις θύρες εισόδου και εξόδου του module leds έχουμε χρησιμοποιήσει τα ονόματα για τα pins που καθορίζονται στο Εγχειρίδιο Χρήσης του DE2: CLOCK_50 για το ρολόι των 50-MHz, KEY για τους διακόπτες pushbutton, SW για τους διακόπτες slider, και LEDG για τα πράσινα LEDs. Χρησιμοποιώντας αυτά τα ονόματα απλοποιείται η δημιουργία των αναγκαίων αντιστοιχιών των pins.

Ενσωμάτωση για ένα VHDL Module

Ο παρακάτω κώδικας είναι η οντότητα top-level VHDL η οποία ενσωματώνει το σύστημα του Nios II. Τον γράφουμε σε ένα αρχείο που ονομάζεται *leds.vhd.*

```
-- Υλοποιεί ένα απλό σύστημα Nios II για την πλακέτα DE2.
-- Είσοδοι: οι διακόπτες SW7-0 είναι είσοδοι παράλληλης θύρας του
-- συστήματος Nios II.
-- Το CLOCK_50 είναι το ρολόι του συστήματος.
```

```
-- Το KEY0 είναι η active-low επαναφορά του συστήματος.
-- Έξοδοι: Τα LEDG7-0 είναι έξοδοι της παράλληλης θύρας από το
-- σύστημα του Nios II.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.all;


ENTITY leds IS
   PORT(
      CLOCK_50    : IN STD_LOGIC;
      KEY         : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
      SW          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      LEDG        : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
      );
END leds;

ARCHITECTURE leds_rtl OF leds IS

//δήλωση των δομικών στοιχείων του nios_system
COMPONENT nios_system
   PORT(
      SIGNAL clk_clk         : IN STD_LOGIC;
      SIGNAL reset_reset_n   : IN STD_LOGIC;
      SIGNAL switches_export : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      SIGNAL leds_export     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

BEGIN

//ενεργοποίηση του nios_system
NiosII : nios_system
   PORT MAP(
      clk_clk => CLOCK_50,
      reset_reset_n => KEY(0),
      switches_export => SW(7 DOWNTO 0),
      leds_export => LEDG(7 DOWNTO 0));

END leds_rtl;
```

4.6.3 Μεταγλώττιση (Compilation) του Project του Quartus II

Προσθέτουμε το αρχείο που μόλις δημιουργήσαμε στο project στο Quartus II επιλέγοντας **Project > Add Current File to Project**. Επίσης, πρέπει να προσθέσουμε στο project μας το απαραίτητο αρχείο με τις εκχωρήσεις των pins για την πλακέτα DE2 στην περίπτωσή μας, το οποίο μπορούμε να το βρούμε στην ιστοσελίδα της Altera (www.altera.com) και ονομάζεται *de2_pin.qsf.* Για να εισάγουμε το αρχείο με

τα pins στο project επιλέγουμε **Assignments > Import Assignments** και επιλέγουμε το παραπάνω αρχείο από την τοποθεσία που βρίσκεται.

Από τη στιγμή που το σύστημα που σχεδιάζουμε χρειάζεται για να λειτουργήσει ένα ρολόι συχνότητας 50MHz, μπορούμε να προσθέσουμε την απαραίτητη εκχώρηση χρονισμού στο project του Quartus, επιλέγοντας **Assignments > TimeQuest Timing Analyzer Wizard** και κάνουμε τις κατάλληλες διαμορφώσεις. Για το απλό σχέδιό μας, θα χρησιμοποιήσουμε την προεπιλεγμένη εκχώρηση χρονισμού, την οποία ο μεταγλωττιστής του Quartus II λαμβάνει χωρίς συγκεκριμένες προδιαγραφές. Ο μεταγλωττιστής (compiler) υποθέτει ότι το κύκλωμα πρέπει να είναι σε θέση να λειτουργήσει με συχνότητα 1GHz, και ότι θα παράγει μια υλοποίηση που είτε πληροί αυτή την απαίτηση ή έρχεται όσο πιο κοντά σε αυτό όσο το δυνατόν.

Τέλος, πριν την μεταγλώττιση του project, είναι απαραίτητο να προσθέσουμε το αρχείο *nios_system.qip* (IP Variation file) που βρίσκεται στον φάκελο *my_first_nios/nios_system/synthesis* στο project μας. Αυτό γίνεται επιλέγοντας **Project > Add/Remove Files in Project** και στην συνέχεια κάνοντας Add το αντίστοιχο αρχείο. Στη συνέχεια, κάνουμε τη μεταγλώττιση του project (**Processing>Start Compilation**). Τα μηνύματα προειδοποίησης που σχετίζονται με το σύστημα Nios II, όπως για παράδειγμα ορισμένα σήματα που δεν έχουν χρησιμοποιηθεί ή έχουν λάθος μήκη bit. Αυτές οι προειδοποιήσεις μπορούν να αγνοηθούν. [Εικόνα 4.28]

Εικόνα 4.28: Επιτυχημένη μεταγλώττιση με προειδοποιήσεις

4.6.4 Χρήση του Altera Monitor Program για την Ανάκτηση του Σχεδιασμένου Κυκλώματος και την Εκτέλεση ενός Προγράμματος Εφαρμογής

Το κύκλωμα μας πρέπει να μεταφορτωθεί στο FPGA της πλακέτας DE2. Αυτό μπορεί να γίνει με τη χρήση του εργαλείου Programmer (προγραμματιστής) του Quartus II. Ωστόσο, μπορούμε να χρησιμοποιήσουμε και το Altera Monitor Program [Ενότητα 4.5], το οποίο είναι ένα απλό μέσο για την μεταφόρτωση του κυκλώματος στο FPGA και την εκτέλεση των προγραμμάτων της εφαρμογής.

Μια παράλληλη διασύνδεση που παράγεται από το Qsys είναι προσβάσιμη μέσω των καταχωρητών στην διασύνδεση. Ανάλογα με το πώς έχει ρυθμιστεί η PIO μπορεί να υπάρχουν μέχρι τέσσερις καταχωρητές. Ένας από τους καταχωρητές ονομάζεται *Data register* (καταχωρητής δεδομένων). Σε μία PIO που ρυθμίζεται ως διασύνδεση εισόδου, τα δεδομένα που διαβάζονται από τον Data register είναι τα δεδομένα που υπάρχουν επί του παρόντος στις γραμμές εισόδου της PIO.). Σε μία PIO που ρυθμίζεται ως έξοδος, τα δεδομένα που γράφονται (από τον επεξεργαστή Nios II) στον καταχωρητή Data οδηγούν τις γραμμές εξόδου της PIO. Εάν μία PIO έχει ρυθμιστεί ως διπλής κατεύθυνσης διασύνδεση, τότε οι είσοδοι και έξοδοι της PIO χρησιμοποιούν τις ίδιες φυσικές γραμμές. Σε αυτή την περίπτωση υπάρχει ένας καταχωρητής Data Direction (κατεύθυνσης δεδομένων), που καθορίζει την κατεύθυνση της μεταφοράς εισόδου/εξόδου. Στη δική μας περίπτωση έχουμε PIO μίας κατεύθυνσης, άρα απαραίτητος είναι μόνο ο καταχωρητής δεδομένων. Οι διευθύνσεις που εκχωρήθηκαν από το Qsys είναι *0x00002000* για τον καταχωρητή Data στην PIO που ονομάζεται *switches* και *0x00002010* για τον καταχωρητή Data στην PIO που ονομάζεται *LEDs* [Εικόνα 4.24].

Στην εφαρμογή μας, ένα σχέδιο που επιλέγεται από την τρέχουσα ρύθμιση των διακοπτών πρέπει να εμφανίζεται μέσω των LED. Θα δείξουμε πώς αυτό μπορεί να γίνει με την γλώσσα assembly του Nios II και με την γλώσσα προγραμματισμού C.

a. Ένα πρόγραμμα σε γλώσσα Assembly του Nios II

Το παρακάτω πρόγραμμα του Nios II σε assembly φορτώνει τις διευθύνσεις των καταχωρητών δεδομένων των δυο PIOs στους καταχωρητές *r2* και *r3*. Στη συνέχεια έχει ένα άπειρο βρόχο (loop) που μεταφέρει τα δεδομένα από την είσοδο PIO, *switches*, στην έξοδο PIO, *leds*.

```
            .equ        switches, 0x00002000
            .equ        leds, 0x00002010
            .global     _start
_start:     movia       r2, switches
            movia       r3, leds
LOOP:       ldbio       r4, 0(r2)
            stbio       r4, 0(r3)
            br          LOOP
            .end
```

Η εντολή *.global _start* δείχνει ότι στον Assembler η ετικέτα *_start* είναι προσβάσιμη έξω από το assembled object file. Η ετικέτα αυτή είναι η προεπιλεγμένη ετικέτα που χρησιμοποιούμε για να δείξει στο πρόγραμμα Linker την έναρξη του προγράμματος της εφαρμογής.

Εισάγουμε τον κώδικα αυτό στο αρχείο *leds.s*, και τοποθετούμε το αρχείο σε ένα φάκελο που εργαζόμαστε, *my_first_nios/app_software*.

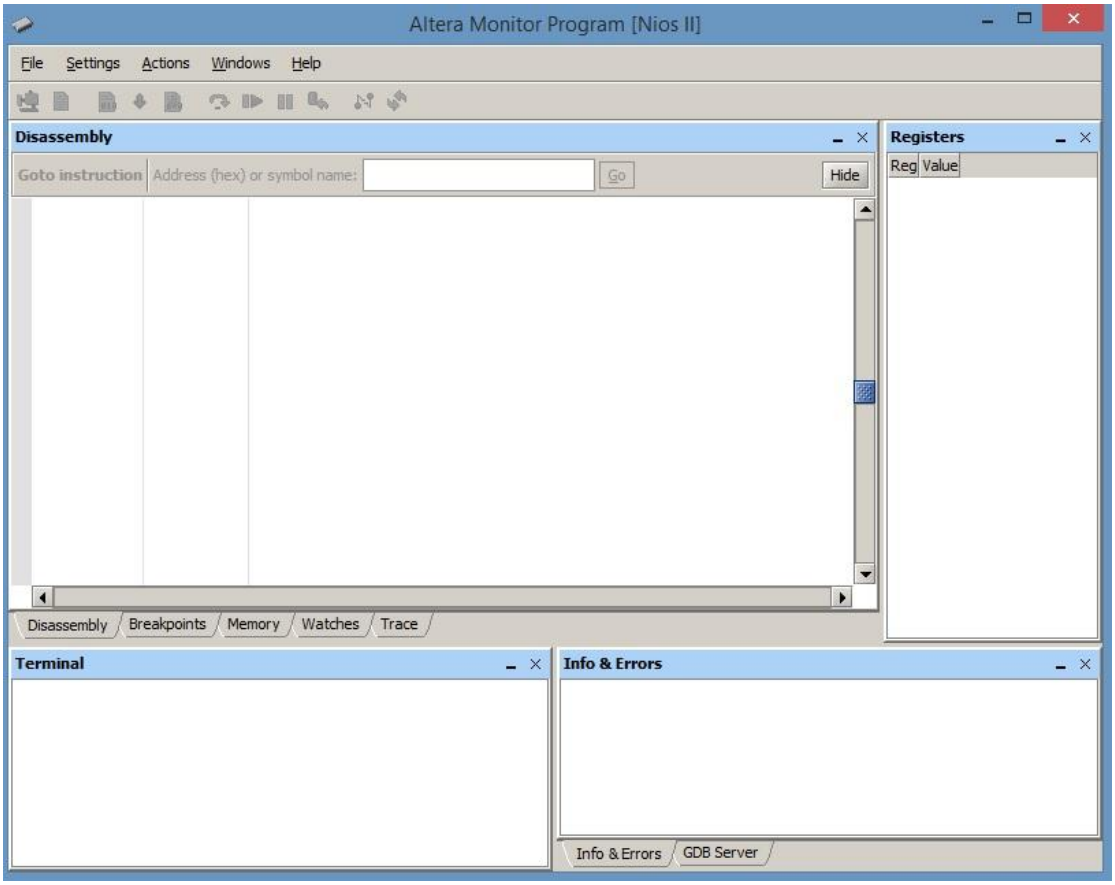
b. Ένα πρόγραμμα γλώσσας C

Ένα πρόγραμμα γραμμένο σε γλώσσα C μπορεί να γίνει με τον ίδιο τρόπο όπως με την γλώσσα assembly. Ένα πρόγραμμα C είναι και το παρακάτω που υλοποιεί το παράδειγμά μας. Εισάγουμε τον κώδικα στο αρχείο *leds.c*, και το τοποθετούμε στο φάκελο του project μας.

```
#define switches (volatile char *) 0x0002000
#define leds (char *) 0x0002010

void main()
{
      while (1)
      *leds = *switches;
}
```

Χρήση του Altera Monitor Program

Το Altera University Program παρέχει το λογισμικό *monitor*, που ονομάζεται *Altera Monitor Program*, για χρήση με τις αναπτυξιακές πλακέτες DE-σειράς [Ενότητα 4.5].



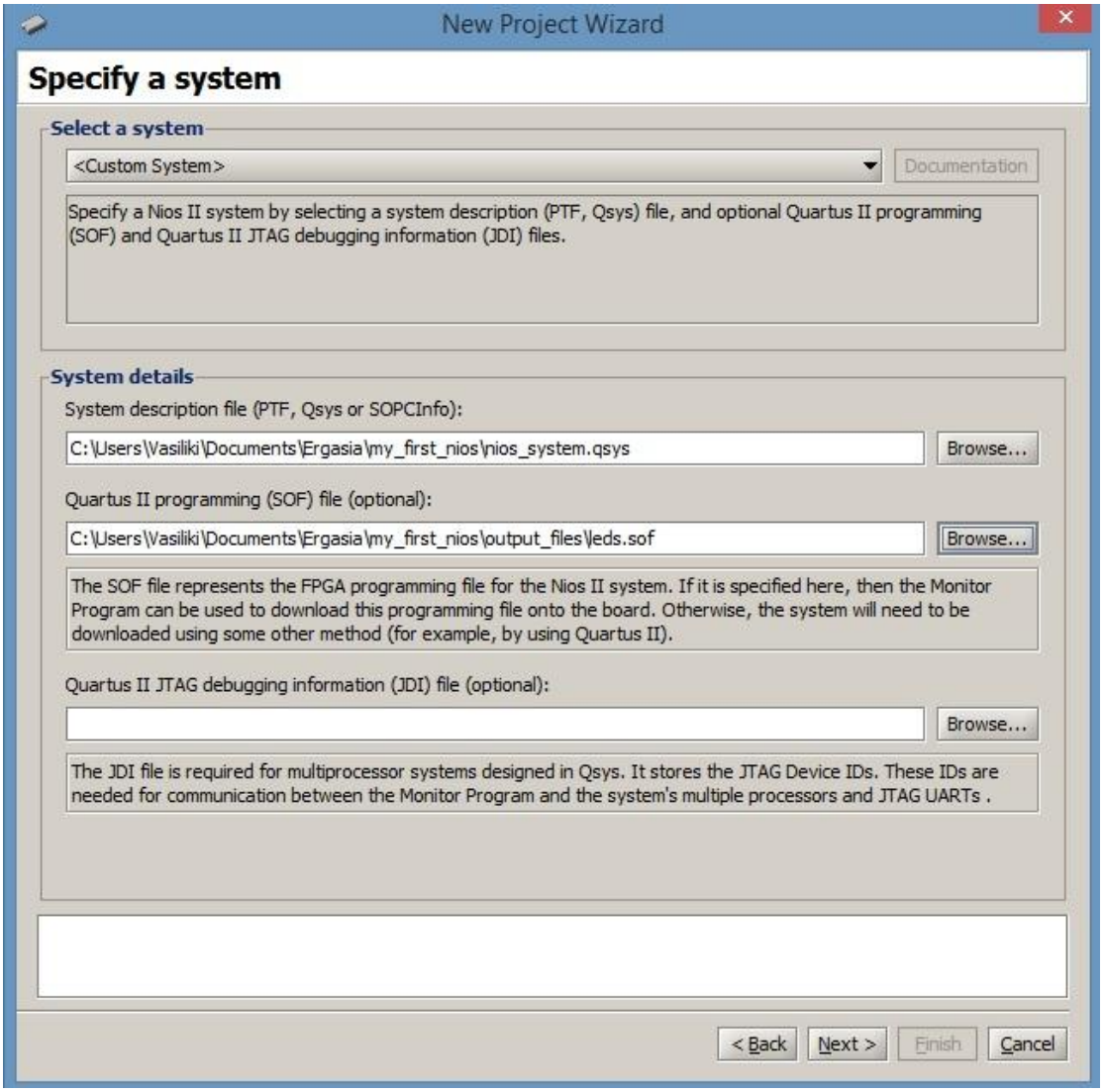
Εικόνα 4.29: Το κύριο παράθυρο του Altera Monitor Program

Το monitor program χρειάζεται να ξέρει τα χαρακτηριστικά του σχεδιασμένου συστήματος Nios II, το οποίο δίνεται στο αρχείο *nios_system.qsys*. Επιλέγοντας **File > New Project** εμφανίζεται το παράθυρο New Project Wizard. Στο πλαίσιο του Project directory ορίζουμε το *my_first_nios/app_software* και στο πλαίσιο Project name ορίζουμε το leds_example [Εικόνα 4.30] και πατάμε το Next.

Στο επόμενο παράθυρο [Εικόνα 4.31], από το αναδυόμενο πλαίσιο Select a system επιλέγουμε το *Custom System*, το οποίο καθορίζει ότι επιθυμούμε για χρήση του hardware, που έχουμε σχεδιάσει. Στο πλαίσιο System description επιλέγουμε το αρχείο *nios_system.qsys*, το οποίο βρίσκεται στο φάκελο my_first_nios. Στο πεδίο Quartus II programming (SOF) file ορίζουμε το αρχείο *leds.sof* (βρίσκεται στον

φάκελο output_files), το οποίο παρέχει τις πληροφορίες που χρειάζονται για την μεταφόρτωση του σχεδιασμένου συστήματος στο FPGA. Στην συνέχεια πατάμε το Next για να μεταβούμε στο επόμενο παράθυρο.

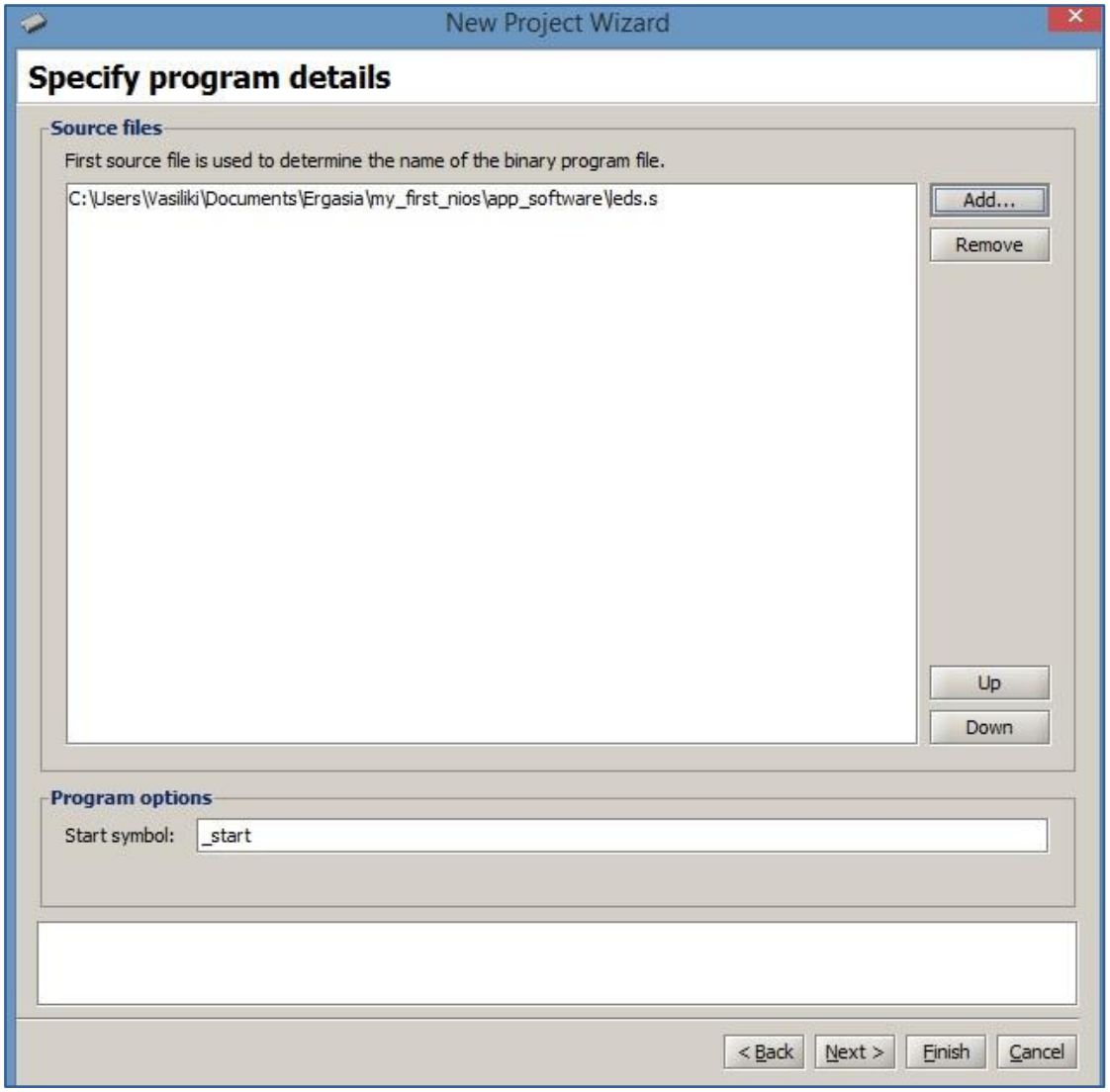
Εικόνα 4.31: Καθορισμός του φακέλου και του ονόματος του Project



Εικόνα 4.30: Παράθυρο καθορισμού συστήματος

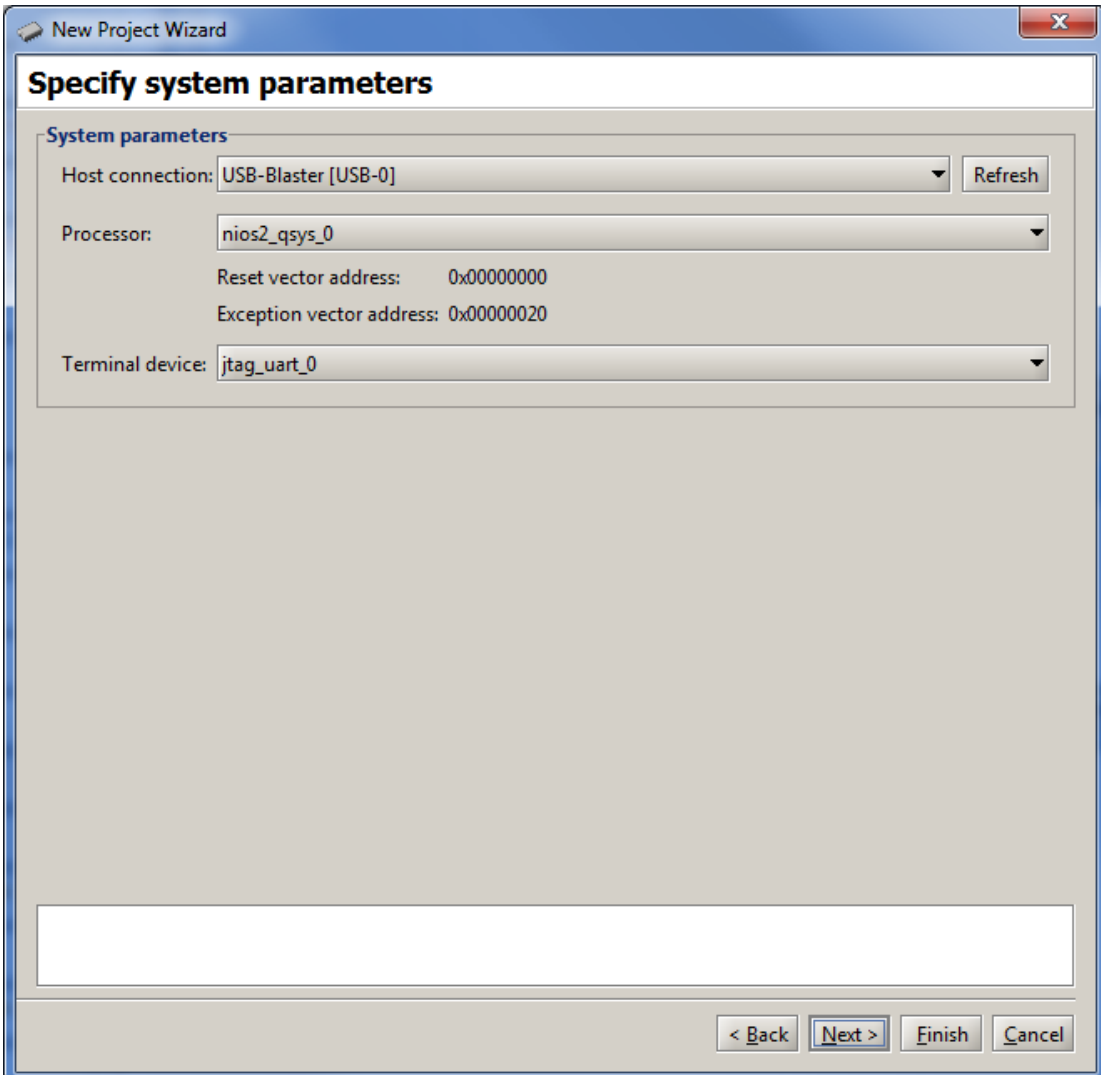
Στο αναδυόμενο μενού του επόμενου παράθυρου επιλέγουμε τον τύπο του προγράμματος που θέλουμε να χρησιμοποιήσουμε. Δηλαδή αν θέλουμε να χρησιμοποιήσουμε πρόγραμμα γλώσσας assembly του Nios II, επιλέγουμε *Assembly Program*. Ενώ, αν επιθυμούμε την χρήση προγράμματος γλώσσας προγραμματισμού C, επιλέγουμε *C Program*. Επιλέγουμε να συνεχίσουμε με Assembly Program. Πατάμε Next για να μεταβούμε στο επόμενο παράθυρο.

Στο επόμενο παράθυρο κάνουμε κλικ στο Add και επιλέγουμε το αρχείο leds.s. Μόλις επιστρέψουμε στην Εικόνα 4.32 πατάμε και πάλι το Next.



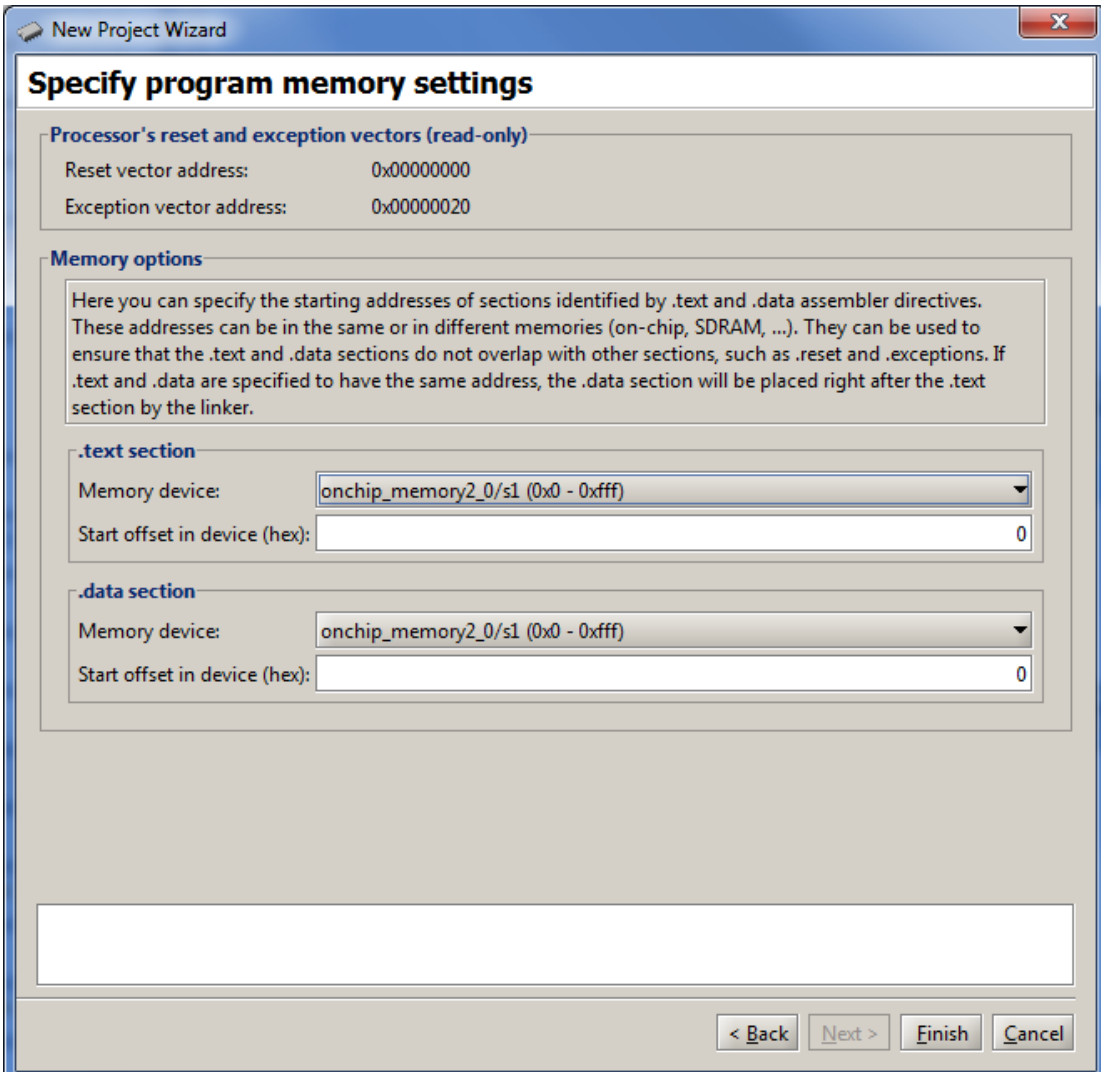
Εικόνα 4.32: Επιλογή του προγράμματος της εφαρμογής που θα χρησιμοποιήσουμε.

Στο επόμενο παράθυρο, ορίζουμε το αναδυόμενο μενού Host Connection στην επιλογή USB-Blaster, στο πλαίσιο Processor επιλέγουμε το nios2_qsys_0 και στο πλαίσιο Terminal Device ορίζουμε το jtag_uart [Εικόνα 4.33] και πατάμε το Next.



Εικόνα 4.33: Καθορισμός των παραμέτρων του συστήματος

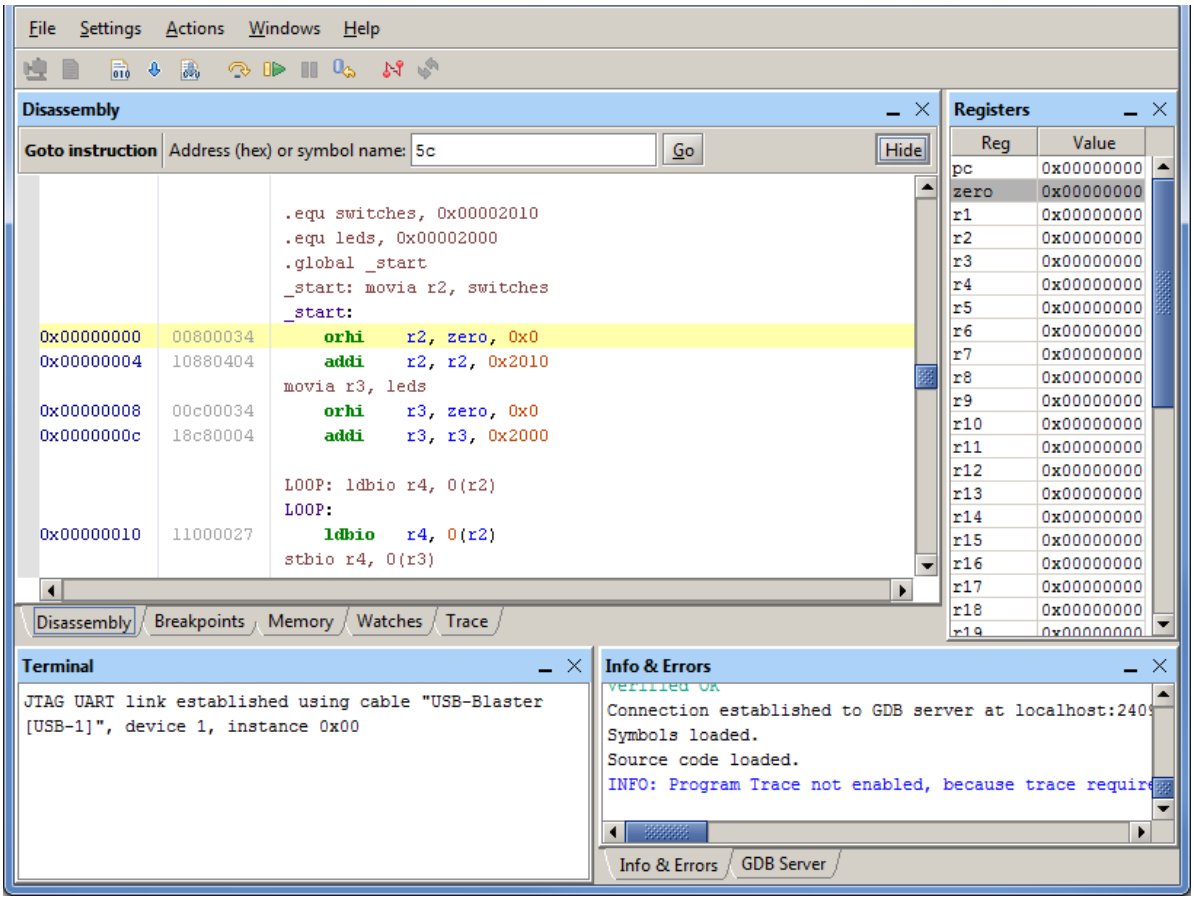
Το Monitor Program επίσης χρειάζεται να γνωρίζει που να φορτώσει το πρόγραμμα της εφαρμογής. Στην περίπτωσή μας, είναι το μπλοκ μνήμης στο FPGA. Το όνομα που καταχωρήθηκε σε αυτή τη μνήμη είναι *onchip_memory*. Αφού δεν υπάρχουν άλλες μνήμες στο σχέδιό μας, το Monitor Program θα επιλέξει αυτή τη μνήμη ως προεπιλογή [Εικόνα 4.34]. Έχοντας δώσει όλες τις απαραίτητες πληροφορίες, πατάμε το Finish για να επιβεβαιώσουμε την διαμόρφωση του συστήματος. Όταν ένα αναδυόμενο πλαίσιο ρωτήσει αν θέλουμε να μεταφορτώσουμε το σύστημά μας στην πλακέτα DE2 επιλέγουμε Yes.



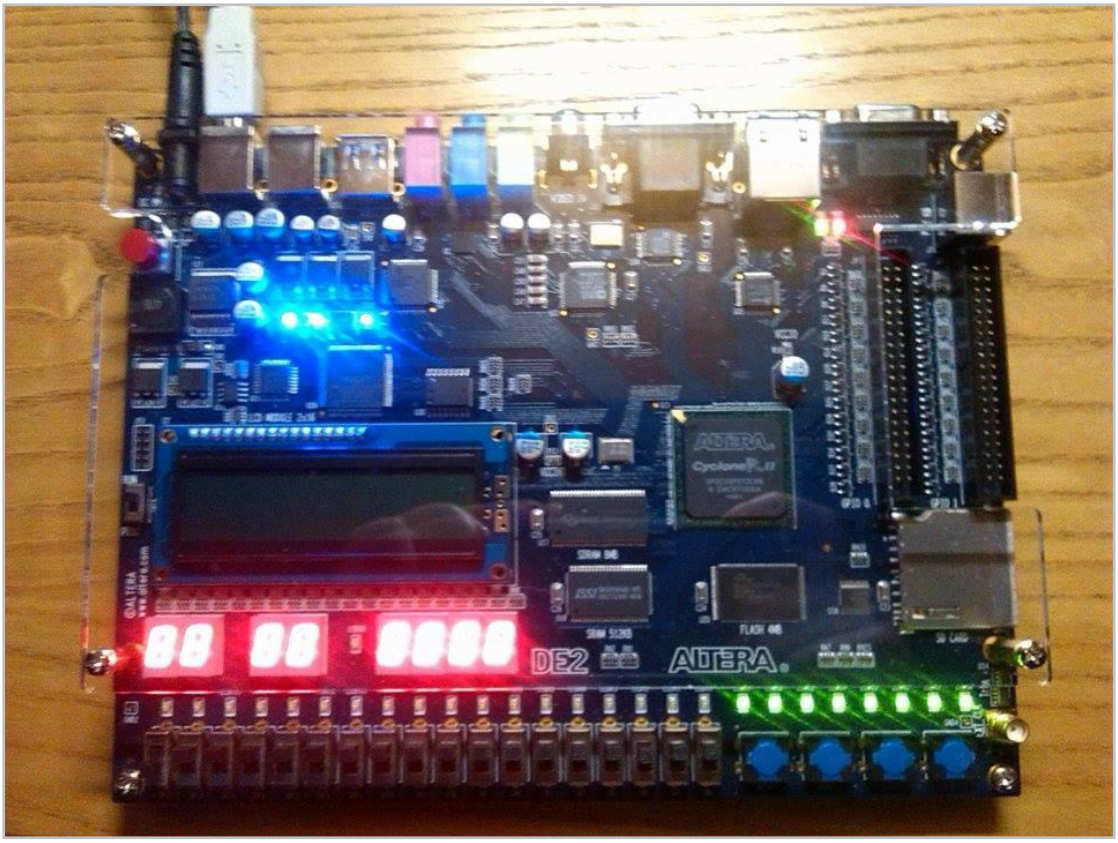
Εικόνα 4.34: Καθορισμός του μέρους που θα μεταφορτωθεί το πρόγραμμα στη μνήμη

Στη συνέχεια, στο κύριο παράθυρο του Altera Monitor Program επιλέγουμε Actions > Compile & Load για να μεταφραστεί και να μεταφορτωθεί το πρόγραμμα.

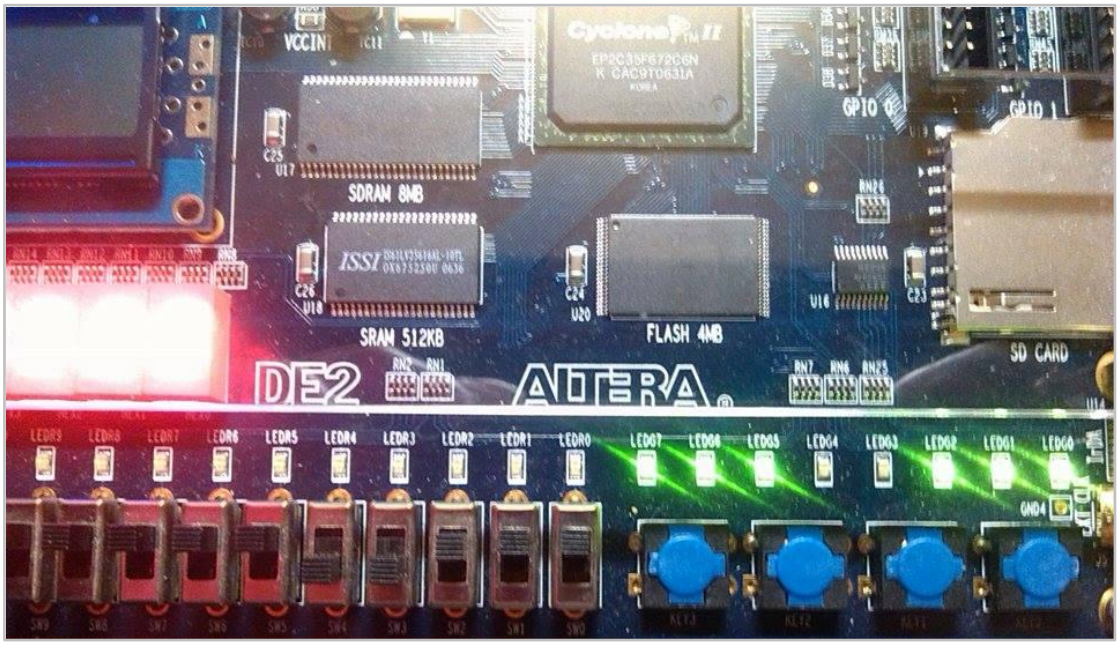
Το μεταφορτωμένο πρόγραμμα φαίνεται στην παρακάτω εικόνα [Εικόνα 4.35]. Εκτελούμε το πρόγραμμα και επαληθεύουμε το σχεδιασμένο σύστημα θέτοντας τους διακόπτες σε διάφορα μοτίβα, περιμένουμε να ανάψουν και τα αντίστοιχα Leds. [Εικόνες 4.36, 4.37]



Εικόνα 4.35: Απεικόνιση του μεταφορτωμένου προγράμματος



Εικόνα 4.37: Απεικόνιση του προγράμματος στην αναπτυξιακή πλακέτα DE2, με τους 8 διακόπτες σηκωμένους (θέση 1), άρα και τα leds (πράσινα) είναι όλα αναμμένα.



Εικόνα 4.36: Απεικόνιση του προγράμματος στην αναπτυξιακή πλακέτα DE2, με τους διακόπτες SW3=SW4=0 και τους υπόλοιπους να ισούνται με 1. Επομένως, τα πράσινα leds που είναι σβηστά είναι τα LEDG3=LEDG4=0, τα υπόλοιπα είναι αναμμένα.

Κεφάλαιο 5 : Εργαστηριακές Εφαρμογές

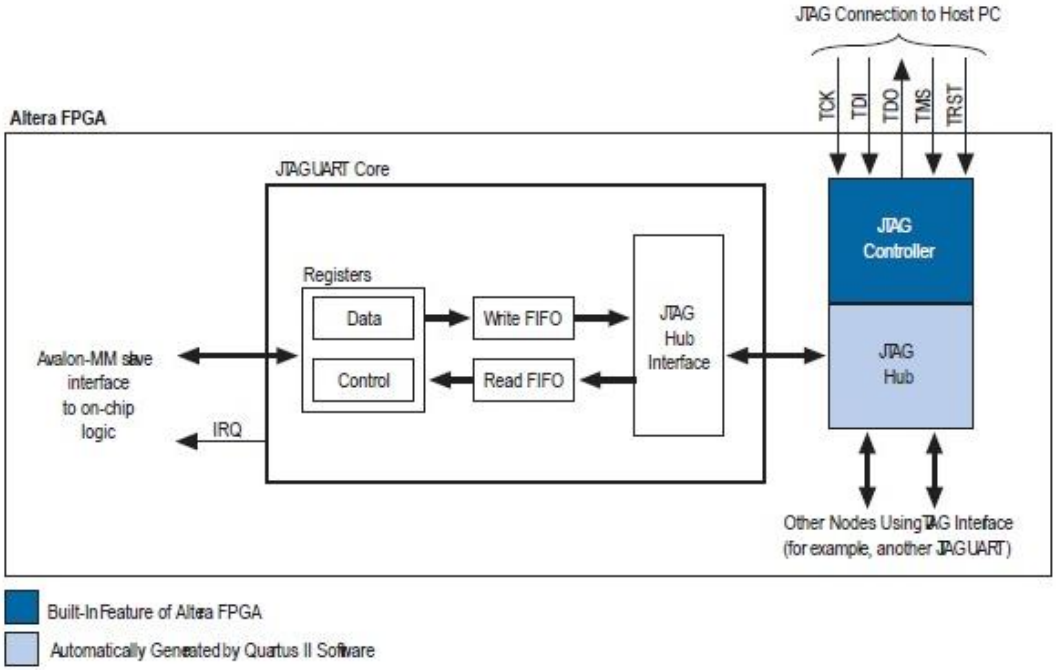
5.1 Πρώτη Εργαστηριακή Εφαρμογή - Αποστολή Ενός Χαρακτήρα στον Nios II

Στην εφαρμογή αυτή θα δείξουμε ένα παράδειγμα αποστολής δεδομένων, και συγκεκριμένα ενός χαρακτήρα, στον επεξεργαστή Nios II, με σειριακή επικοινωνία, μέσω της διασύνδεσης *JTAG UART*, η οποία είναι ένα από τα περιφερειακά του Nios II και είναι συνηθισμένο κύκλωμα για μεταφορά δεδομένων μεταξύ επεξεργαστή και συσκευών I/O.

Ας δούμε πρώτα όμως αναλυτικά της λειτουργία της θύρας JTAG γιατί πάνω σε αυτή βασίζονται και οι τρεις εργαστηριακές εφαρμογές.

Η **θύρα JTAG** υλοποιεί έναν σύνδεσμο επικοινωνίας μεταξύ της αναπτυξιακής πλακέτας DE2 και του υπολογιστή. Αυτός ο σύνδεσμος χρησιμοποιείται αυτόματα από το λογισμικό Quartus II για την μεταφορά αρχείων προγραμματισμού του FPGA στην πλακέτα DE2, και από το Altera Monitor Program. Η θύρα JTAG περιλαμβάνει επίσης ένα **UART**, το οποίο μπορεί να χρησιμοποιηθεί για να μεταφέρει σειριακά δεδομένα χαρακτήρων μεταξύ του υπολογιστή και των προγραμμάτων που εκτελούνται στον επεξεργαστή Nios II. Η διασύνδεση JTAG UART , όπως έχουμε δει σε προηγούμενα κεφάλαια [Σχήμα 4.4], είναι τοποθετημένη μεταξύ του επεξεργαστή και των συσκευών εισόδου/εξόδου. Αποθηκεύει έναν χαρακτήρα και τον εκπέμπει σειριακά. Η μεταφορά των δεδομένων μεταξύ των JTAG UART και επεξεργαστή γίνεται παράλληλα, όπου όλα τα bits ενός χαρακτήρα μεταφέρονται την ίδια στιγμή από τον επεξεργαστή προς τη JTAG UART, μέσω του διαύλου δεδομένων. Η μεταφορά δεδομένων μεταξύ JTAG UART και συσκευών εισόδου/εξόδου γίνεται σειριακά μεταφέροντας ένα bit κάθε φορά. Το Qsys tool μπορεί να υλοποιήσει μια διασύνδεση JTAG UART για χρήση στο σύστημα Nios II, για την μεταφορά χαρακτήρων μεταξύ του επεξεργαστή Nios II και του host υπολογιστή που συνδέεται στην αναπτυξιακή πλακέτα DE2. Η JTAG UART συνδέεται από τη μια μεριά με το Avalon Switch Fabric (δίκτυο διασυνδέσεων που δημιουργεί

το Qsys για διασύνδεση του Nios II με περιφερειακά) και από την άλλη με την διασύνδεση JTAG της πλακέτας DE2, που επικοινωνεί με τον host υπολογιστή μέσω του USB Blaster [Σχήμα 5.1]. Όταν στον κεντρικό υπολογιστή χρησιμοποιείται το Altera Monitor Program, τότε αυτά τα δεδομένα χαρακτήρων στέλνονται και λαμβάνονται μέσω του παράθυρου του τερματικού (Terminal).



Σχήμα 5.1: Block διάγραμμα του πυρήνα της JTAG UART. [21]

Η διασύνδεση JTAG UART περιέχει δύο ειδών καταχωρητές των 32-bits, τον καταχωρητή δεδομένων (Data register) και τον καταχωρητή ελέγχου (Control register), οι οποίοι προσπελαύνονται από τον επεξεργαστή σαν θέσεις μνήμης. Η διεύθυνση του καταχωρητή ελέγχου είναι 4 bytes υψηλότερα από την διεύθυνση του καταχωρητή δεδομένων.

Ο πυρήνας του JTAG UART περιέχει δύο FIFOs που προσφέρουν buffers αποθήκευσης δύο ειδών. Ένα για αποθήκευση των δεδομένων που λήφθηκαν από τον host υπολογιστή και ένα για αποθήκευση δεδομένων που θα σταλούν στον host υπολογιστή. Όταν τα δεδομένα χαρακτήρων λαμβάνονται από τον κεντρικό υπολογιστή από την JTAG UART αποθηκεύονται σε ένα FIFO 64-χαρακτήρων. Ο αριθμός των χαρακτήρων που είναι αποθηκευμένοι σε αυτό το FIFO αναγράφεται

στο πεδίο RAVAIL, που είναι τα bits 31-16 του καταχωρητή δεδομένων. Αν το FIFO λήψης υπερχειλίσει, τότε τα επιπλέον δεδομένα θα χαθούν. Όταν τα δεδομένα βρίσκονται στο FIFO λήψης, τότε η τιμή του RAVAIL θα είναι μεγαλύτερη του 0 και η τιμή του bit 15, RVALID, θα είναι 1. Διαβάζοντας τον επικεφαλή χαρακτήρα του FIFO, ο οποίος δίνεται στα bits 7-0, μειώνεται αντίστοιχα η τιμή του RAVAIL κατά 1 και επιστρέφει την μειωμένη αυτή τιμή ως μέρος της συνάρτησης read. Εάν δεν υπάρχει κανένα δεδομένο στο FIFO λήψης, τότε το RVALID θα γίνει 0 και τα δεδομένα στα bits 7-0 θα είναι απροσδιόριστα.

Offset	Καταχωρητές	R/W	Περιγραφή των Bits														
			31	..	16	15	14	..	11	10	9	8	7	..	2	1	0
0	Data	RW	RAVAIL			RV	Unused						DATA				
1	Control	RW	WSPACE			Unused				AC	WI	RI	Unused			WE	RE

Πίνακας 5.1: Οι Καταχωρητές της JTAG UART - Στα πεδία Unused οι τιμές ανάγνωσης είναι απροσδιόριστες γι' αυτό γράφουμε 0. [21]

Το άλλο FIFO της JTAG UART είναι επίσης 64-χαρακτήρων και αποθηκεύει τα δεδομένα που είναι στην αναμονή για την μεταφορά τους στον κεντρικό υπολογιστή. Τα δεδομένα χαρακτήρων φορτώνονται σε αυτό το FIFO εκτελώντας μία εγγραφή στα bits 7-0 του καταχωρητή δεδομένων [Πίνακας 5.1]. Παρατηρούμε ότι η εγγραφή στον καταχωρητή δεν έχει καμία επίδραση στα δεδομένα που λήφθηκαν. Το μέγεθος του χώρου, WSPACE, που διατίθεται στο FIFO μεταφοράς δίνεται στα bits 31-16 του καταχωρητή ελέγχου. Εάν το FIFO μεταφοράς είναι πλήρης, τότε οποιοιδήποτε χαρακτήρες που είναι γραμμένοι στον καταχωρητή δεδομένων θα χαθούν.

Το bit 10 στον καταχωρητή ελέγχου, AC, έχει την τιμή 1 αν η JTAG UART έχει προσπελασθεί από τον κεντρικό υπολογιστή. Αυτό το bit μπορεί να χρησιμοποιηθεί για να ελέγχει αν έχει πραγματοποιηθεί σύνδεση με τον κεντρικό υπολογιστή. Το bit AC μπορεί να γίνει 0 γράφοντας ένα 1 σε αυτό. Τα bits RE και WE χρησιμοποιούνται για την ενεργοποίηση των διακοπών του επεξεργαστή που έχουν σχέση με τα FIFOs λήψης και μετάδοσης. Όταν ενεργοποιούνται, οι διακοπές δημιουργούνται όταν το RAVAIL για το FIFO λήψης , ή το WSPACE για το FIFOμετάδοσης, υπερβαίνει το 7. Οι διακοπές που εκκρεμούν αναφέρονται στα bits RI και WI του καταχωρητή ελέγχου, και μπορούν να διαγραφούν γράφοντας ή διαβάζοντας δεδομένα σε/από την JTAG UART.

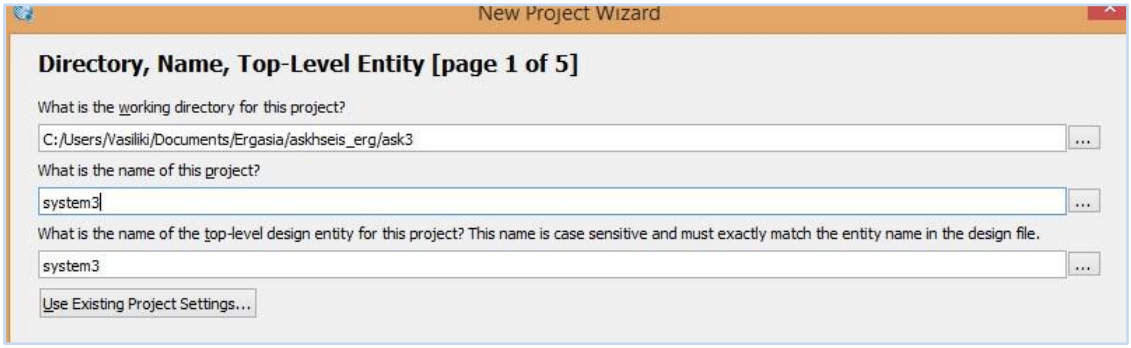
Για την αποστολή και λήψη δεδομένων ελέγχουμε περιοδικά τους καταχωρητές αυτούς, ώστε να διαπιστώσουμε πότε έχει φθάσει έγκυρος χαρακτήρας στη JTAG UART ή πότε υπάρχει χώρος στον FIFO εγγραφής (μετάδοσης) για αποστολή νέου χαρακτήρα. Η μέθοδος αυτή του περιοδικού ελέγχου του περιφερειακού αναφέρεται με τον όρο *polling*.

5.1.1 Διαδικασία σχεδιασμού και υλοποίησης του υλικού της 1ης εργαστηριακής εφαρμογής

Στις παρακάτω ενότητες θα παρουσιαστεί αναλυτικά η μεθοδολογία σχεδιασμού του υλικού του συστήματος της 1ης εργαστηριακής εφαρμογής, με τα εργαλεία που μας παρέχει η Altera για αυτό το σκοπό. Θα δημιουργήσουμε το σύστημα της εφαρμογής με την βοήθεια του εργαλείου Qsys (νεότερο εργαλείο σχεδιασμού από το SoPC Builder). Το σύστημα θα περιέχει τον επεξεργαστή Nios II, on-chip μνήμη RAM και μία JTAG UART.

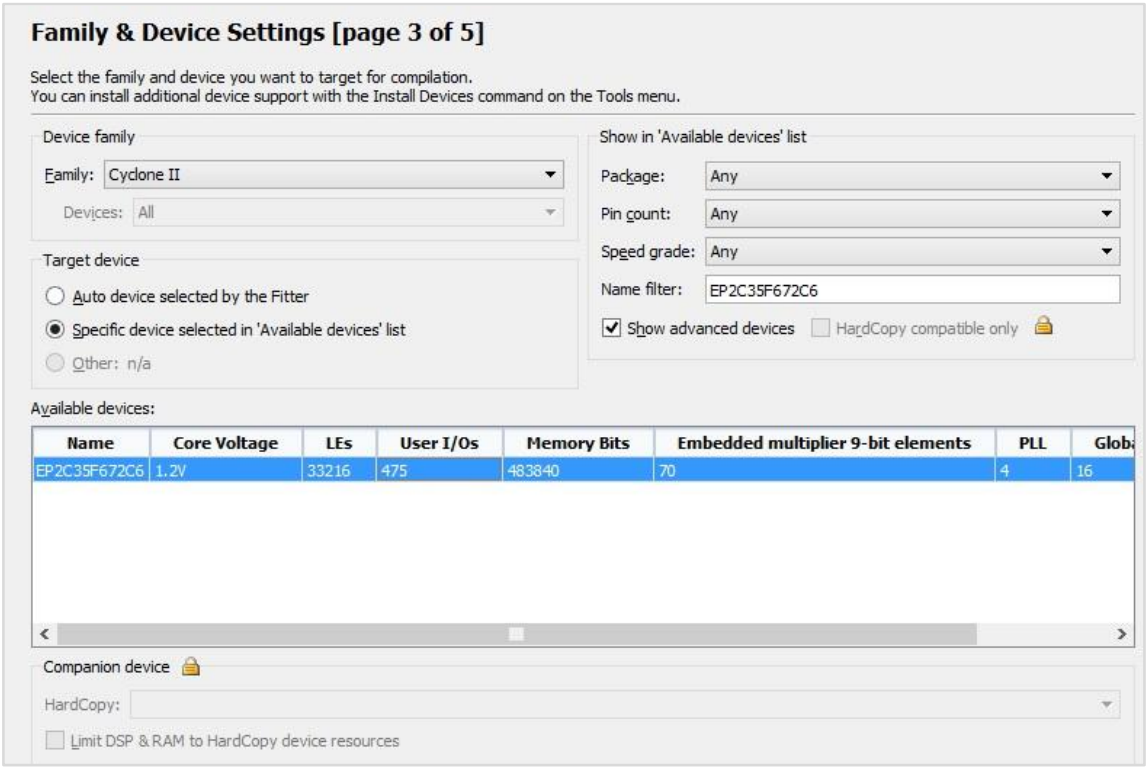
5.1.2 Δημιουργία του project στο Quartus II

Για την δημιουργία ενός project πρέπει να επιλέξουμε *File > New Project Wizard* και εκεί θα ορίσουμε το φάκελο που θα εργαστούμε και το όνομα του project, το οποίο είναι ίδιο με το top-level αρχείο.



Εικόνα 5.1: Δημιουργία project στο Quartus II.

Στην επόμενη καρτέλα δεν αλλάζουμε κάτι. Στην σελίδα 3 από τις 5 ορίζουμε την οικογένεια του FPGA και συγκεκριμένα το μοντέλο που διαθέτουμε. Συγκεκριμένα για την πτυχιακή αυτή εργασία χρησιμοποιείται το DE2 EP2C35F672C6 της οικογένειας Cyclone II της Altera [Εικόνα 5.2]. Στα επόμενα βήματα δεν κάνουμε κάποιες αλλαγές και έτσι πατάμε finish.



Εικόνα 5.2: Quartus II: Δημιουργία project 3/5.

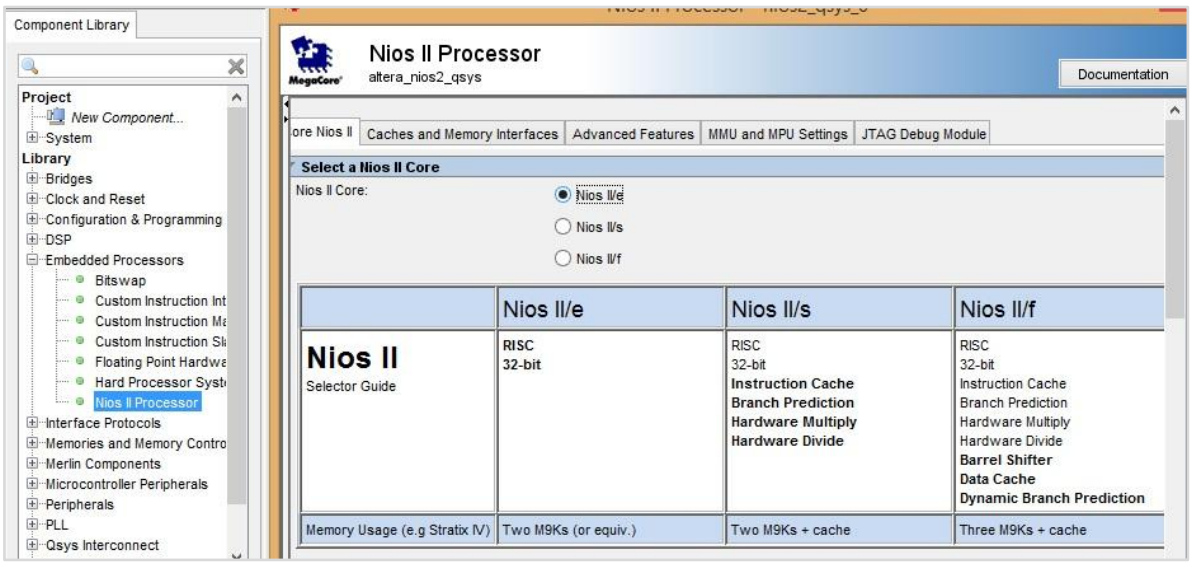
5.1.3 Το σύστημα στο Qsys tool

Για τον σχεδιασμό του συστήματος θα χρησιμοποιήσουμε το εργαλείο Qsys της Altera. Το ανοίγουμε επιλέγοντας *Tools > Qsys*.

Σύμφωνα με τη λογική και τις λειτουργίες που θέλουμε να υλοποιεί το σύστημά μας, θα επιλέξουμε και θα παραμετροποιήσουμε τα components που χρειαζόμαστε. Όπως είπαμε και παραπάνω, θα δημιουργήσουμε ένα σύστημα που θα περιέχει τον επεξεργαστή Nios II, on-chip μνήμη RAM και μία JTAG UART καθώς και το ρολόι για τον χρονισμό του συστήματος.

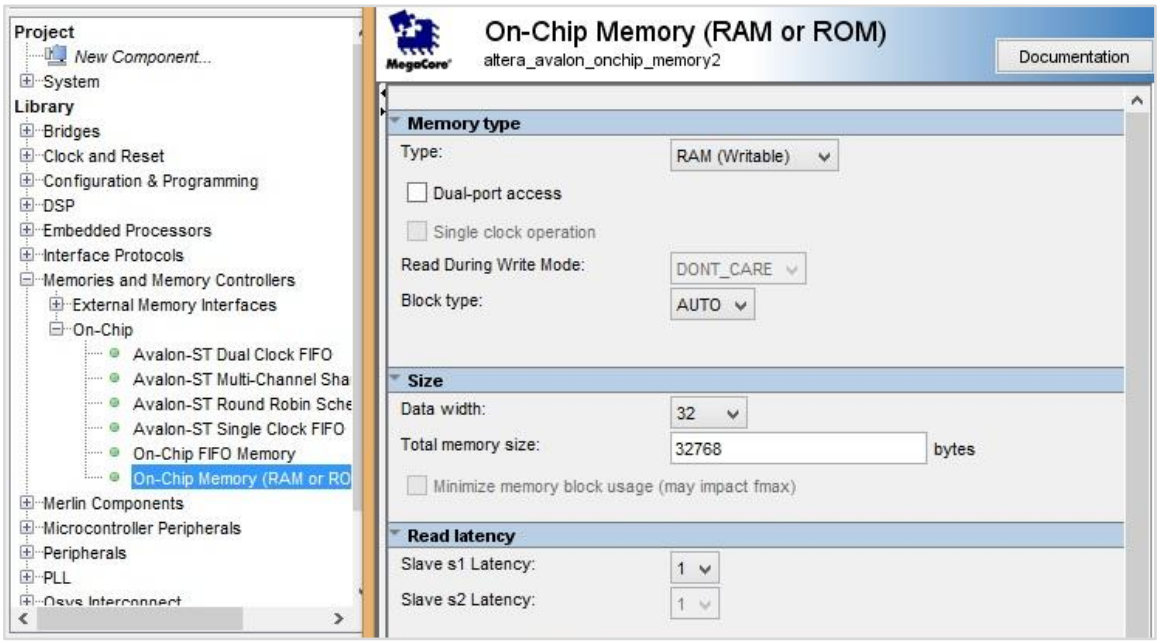
Ήδη αυτόματα έχει δημιουργηθεί ένα clock των 50HZ.

Αρχικά θα εισάγουμε τον επεξεργαστή Nios II. Το κύριο τμήμα του συστήματος είναι ο επεξεργαστής Nios II και αφού τον εισάγουμε (*Embedded Processors > Nios II Processors*) τον παραμετροποιούμε για χρήση στο σύστημα. Διαλέγουμε την έκδοση /e (economy) και το JTAG Debug Module level 1. Η έκδοση economy είναι η μόνη που δίνεται προς χρήση χωρίς χρονικούς περιορισμούς με την δωρεάν έκδοση του επεξεργαστή. Δεν περιέχει cache μνήμες δεδομένων ή εντολών και έχει χαμηλότερη απόδοση σε σχέση με τις άλλες εκδόσεις. [Εικόνα 5.3]

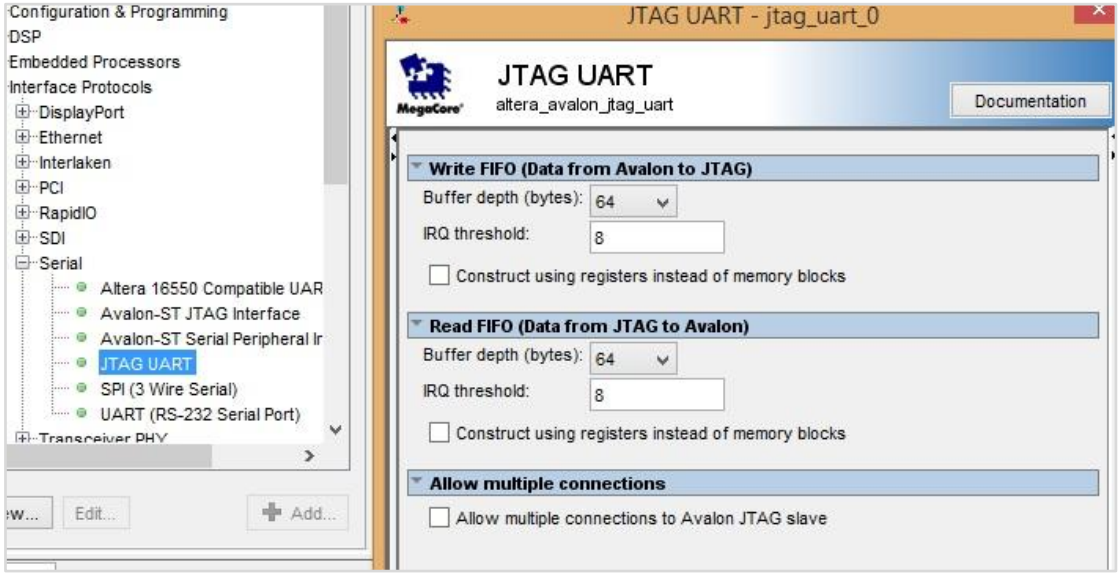


Εικόνα 5.3: Εισαγωγή του επεξεργαστή Nios II στο σύστημα.

Στην συνέχεια εισάγουμε την on-chip μνήμη (*Memories and Memory Controllers > On-chip > On-chip Memory (RAM or ROM)*) RAM μεγέθους 32 Kbytes (32768 bytes) με μήκος 32 bits. Η βασική της διεύθυνση (base address) είναι 0x0000 και η τελική διεύθυνση (end address) είναι 0x7fff. [Εικόνα 5.4]



Εικόνα 5.4: Εισαγωγή της on chip μνήμης RAM στο σύστημα.

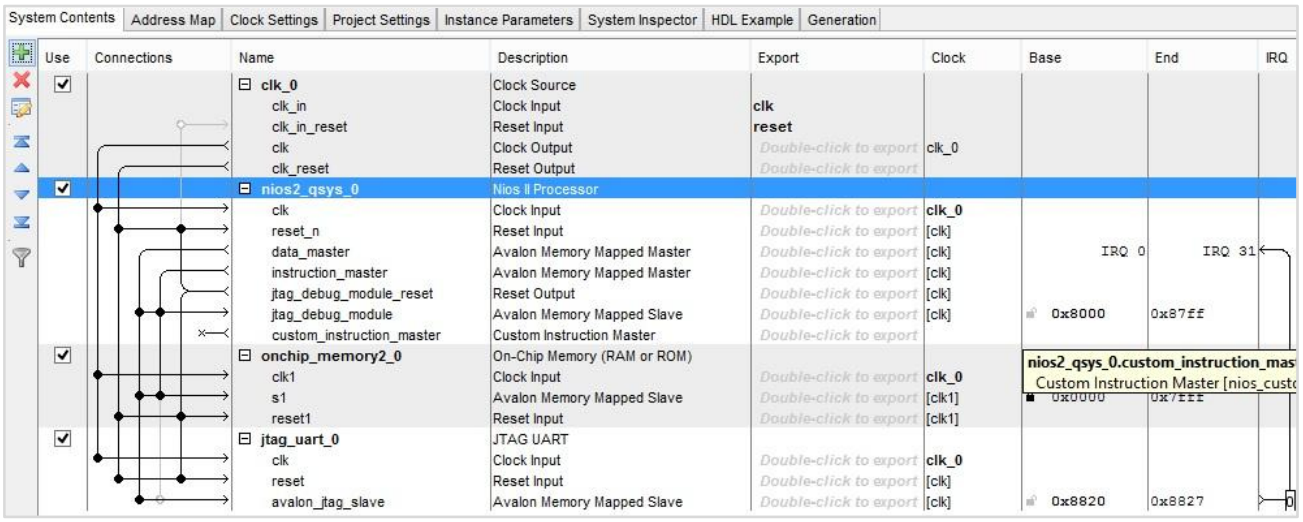


Εικόνα 5.5: Εισαγωγή της διασύνδεσης JTAG UART στο σύστημα.

Τέλος, εισάγουμε την διασύνδεση JTAG UART (*Interface Protocols > Serial > JTAG UART*) με τις προκαθορισμένες ρυθμίσεις, για την επικοινωνία του επεξεργαστή με τον host υπολογιστή. Βασική διεύθυνση της JTAG UART (Base address) είναι 0x8820. [Εικόνα 5.5]

Αφού έχουμε βάλει όλα τα components που χρειάζονται στο σύστημα, πρέπει να γίνουν οι συνδέσεις μεταξύ των components. [Εικόνα 5.6]

Έχοντας ολοκληρώσει τον σχεδιασμό πρέπει να ορίσουμε τις διευθύνσεις με αυτόματο ή μη τρόπο. Ήδη έχουμε ορίσει τις διευθύνσεις της JTAG UART, και της on-chip memory. Θα ορίσουμε αυτόματα τις υπόλοιπες διευθύνσεις επιλέγοντας *System > Assign Base Addresses*. Προσέχω όμως να μην έχουν αλλάξει οι δικές μου ρυθμίσεις στις διευθύνσεις. Τώρα πλέον έχουν φύγει και τα διάφορα errors που εμφανιζόταν κάτω στην οθόνη.



Εικόνα 5.6: Το σύστημα του Nios II της εφαρμογής στο εργαλείο Qsys.

Αφού έχουμε σώσει το σύστημα μας πηγαίνουμε στην καρτέλα Generation, επιλέγω το synthesis file να είναι .vhdl και πατάμε το Generate, ώστε το Qsys να δημιουργήσει όλα τα απαραίτητα αρχεία.

Αφού ολοκληρωθεί με επιτυχία η διαδικασία μπορούμε να συνεχίσουμε.

5.1.4 Ενσωμάτωση συστήματος Nios II στο project του Quartus II

Για την ενσωμάτωση του παραγμένου σχεδίου από το Qsys θα χρησιμοποιήσουμε την VHDL. Μπορούσε να γίνει επίσης με Verilog ή σχηματικά.

Δημιουργούμε το top-level VHDL αρχείο που θα ορίζει και θα ενεργοποιεί τα components. Για να είναι top-level πρέπει να έχει το ίδιο όνομα με το project του Quartus II. Ο παρακάτω κώδικας είναι η οντότητα top-level VHDL, *system3.vhd,* η οποία ενσωματώνει το σύστημα του Nios II, *system_ask3.qsys* .

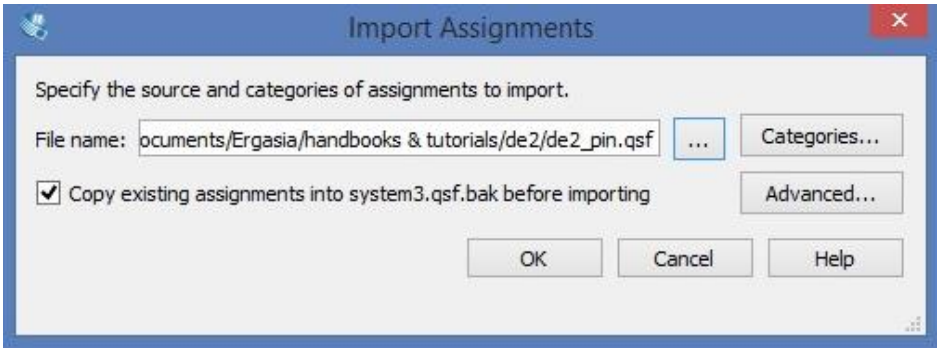
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.all;

ENTITY system3 IS
  PORT(
      CLOCK_50   : IN STD_LOGIC;
      KEY        : IN STD_LOGIC_VECTOR (0 DOWNTO 0)
      );
END system3;

ARCHITECTURE system3_rtl OF system3 IS

//δήλωση δομικών στοιχείων του system_ask3
  COMPONENT system_ask3
      PORT(
            SIGNAL clk_clk        : IN STD_LOGIC;
            SIGNAL reset_reset_n  : IN STD_LOGIC
      );
  END COMPONENT;

BEGIN

//ενεργοποίηση του system_ask3
NiosII : system_ask3
  PORT MAP(
      clk_clk => CLOCK_50,
      reset_reset_n => KEY(0)
      );
END system3_rtl;
```

Χρησιμοποιώντας την ονοματολογία του αρχείου .qsf για τα pin Assignments ορίζουμε την οντότητα *system3.* Με την βοήθεια της δήλωσης port γίνονται οι δηλώσεις των σημάτων διασύνδεσης. Το CLOCK_50 αντιστοιχεί στο ρολόι των

50MHz της πλακέτας DE2 και το KEY(0) είναι ένας από τους 4 διακόπτες buttons ο οποίος κάνει reset την εφαρμογή.

Για να χρησιμοποιήσουμε το σύστημα που δημιουργήσαμε θα κάνουμε import στο project τα αρχεία που δημιούργησε το Qsys και το αρχείο ενσωμάτωσης *system3.vhd*. Επίσης πριν το compile είναι απαραίτητο να προστεθεί στο project το αρχείο *system_ask3.qip* (IP Variation file), και το *system3.sdc*.

Αντιστοίχιση ακροδεκτών



Εικόνα 5.7: Αντιστοίχιση ακροδεκτών της πλακέτας DE2

Η αντιστοίχιση ακροδεκτών γίνεται είτε χειροκίνητα, είτε αυτόματα. Για εκπαιδευτικούς λόγους η Altera περιλαμβάνει ένα .qsf αρχείο *de2_pin.qsf* για την γρήγορη αντιστοίχιση των ακροδεκτών. Επιλέγουμε *Assignments* > *Import Assignments* και επιλέγουμε το παραπάνω αρχείο. [Εικόνα 5.7]

Χρονισμός

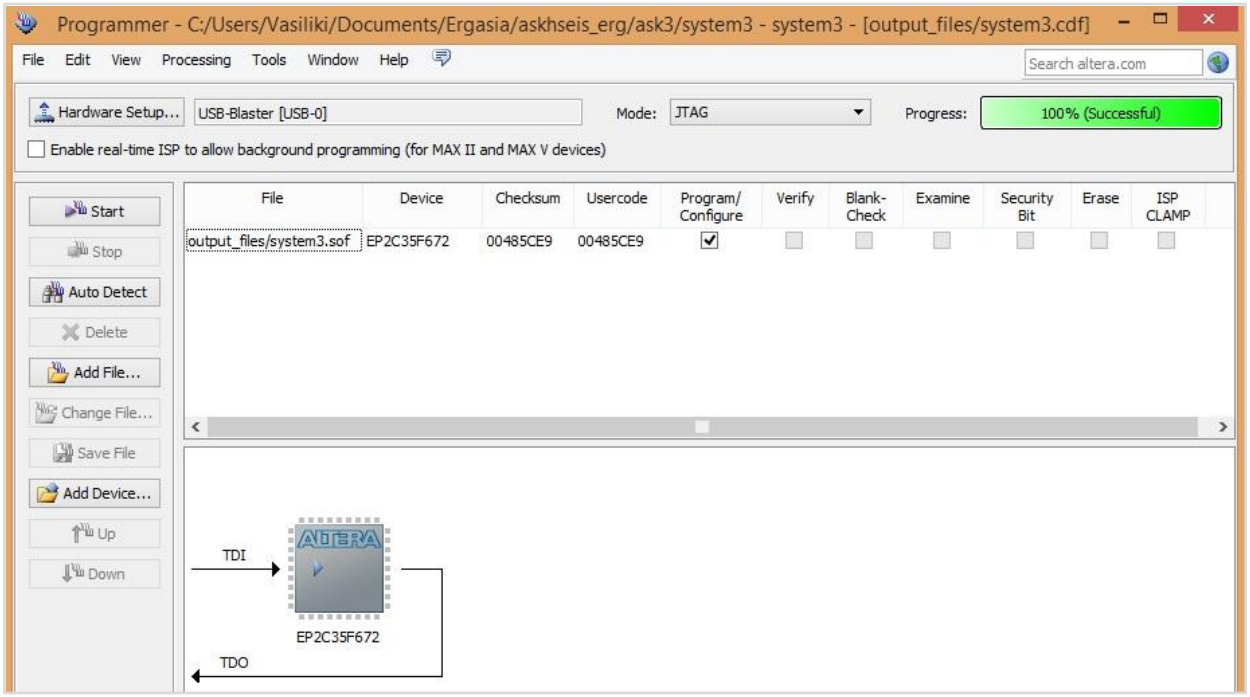
Δεδομένου ότι το σύστημα σχεδιάζουμε πρέπει να λειτουργεί σε συχνότητα χρονισμού 50-MHz, μπορούμε να προσθέσουμε την απαραίτητη εκχώρηση χρονισμού του project του Quartus II, με το *tool TimeQuest Timing Analyzer*. Ωστόσο, για το σύστημα μας, μπορούμε να στηριχθούμε στην προεπιλεγμένη αντιστοίχιση χρονισμού. Ο compiler υποθέτει ότι το κύκλωμα πρέπει να είναι σε θέση να λειτουργεί σε συχνότητα χρονισμού του 1 GHz, και θα παράγει μια εφαρμογή που είτε πληροί αυτή την απαίτηση ή έρχεται όσο πιο κοντά σε αυτό.

5.1.5 Μεταγλώττιση και Προγραμματισμός του FPGA

Έχοντας ολοκληρώσει τα παραπάνω προχωράμε στο Compilation του project μας, επιλέγοντας *Processing > Start Compilation.*

Μόλις ολοκληρωθεί επιτυχώς μπορούμε να περάσουμε στον προγραμματισμό της αναπτυξιακής πλακέτας. Επιλέγουμε *Tools > Programmer*, επιλέγουμε το αρχείο .sof που παράχθηκε από το Compilation, επιλέγουμε το check box Program/Configure. Μόλις αναγνωριστεί το FPGA, από το Auto Detect, πατάμε το Start για να αρχίσει ο προγραμματισμός του επεξεργαστεί. [Εικόνα 5.9]

Εικόνα 5.8: Compilation του project με επιτυχία



Εικόνα 5.9: Μεταφόρτωση του συστήματος στην πλακέτα DE2 μέσω του εργαλείου του Quartus II, Programmer.

Όταν χρησιμοποιούμε το Altera Monitor Program, το βήμα του προγραμματισμού του FPGA μπορεί να παραληφθεί γιατί το Altera Monitor Program κάνει τον προγραμματισμό του FPGA αμέσως μετά την δημιουργία του project της εφαρμογής.

5.1.6 Εκτέλεση της εφαρμογής με το Altera Monitor Program

Το πρόγραμμα που θα δημιουργηθεί σε γλώσσα assembly θα εμφανίζει τον χαρακτήρα Z στο παράθυρο του τερματικού συνεχόμενα μέχρι να σταματήσουμε την εκτέλεση της εφαρμογής. Ανάμεσα στις διαδοχικές εμφανίσεις του χαρακτήρα θα εισάγει μια χρονική καθυστέρηση της τάξης του μισού δευτερολέπτου.

Η JTAG UART μπορεί να στείλει ASCII χαρακτήρες στην κονσόλα του Altera Monitor Program, τους οποίους παρουσιάζει στο παράθυρο του τερματικού. Όταν το WSPACE πεδίο στον καταχωρητή ελέγχου της JTAG UART έχει μια μη μηδενική τιμή τότε το JTAG UART μπορεί να δεχτεί ένα νέο χαρακτήρα, για να αποσταλεί μέσω της σειριακής διασύνδεσης. Για να αποστείλουμε τον χαρακτήρα στον host υπολογιστή, η εφαρμογή που γράφουμε παρακάτω σε assembly διαβάζει συνεχόμενα μέχρι να είναι διαθέσιμο ένα κενό στον FIFO καταχωρητή. Μόλις ένα κενό είναι διαθέσιμο ο ASCII χαρακτήρας μπορεί να γραφτεί στον καταχωρητή δεδομένων της διασύνδεσης JTAG UART.

Ο κώδικας για το πρόγραμμα assembly της εφαρμογής είναι δίνεται αμέσως παρακάτω:

```
.include "nios_macros.s"
.text
.equ UART_BASE,0x8820       /*φόρτωση της διεύθυνσης 0x8820 που
                            αντιπροσωπεύει το JTAG_UART στο
                            UART_BASE.*/

.global _start
_start:
    movia  r8,UART_BASE     /*τοποθέτηση του UART_BASE στον
                                καταχωρητή r8*/
    movi   r16,'Z'          /*τοποθέτηση του γράμματος Z
                                στον r16 */
PUT_CHAR_LOOP:
/*************************************************************
Στον παρακάτω κώδικα προσθέτουμε τα περιεχόμενα του r8 με  το
4 γιατί ο καταχωρητής ελέγχου είναι 4 bytes ψηλότερα από τον
καταχωρητή δεδομένων, και εμείς θέλουμε να χρησιμοποιήσουμε
τον καταχωρητή ελέγχου για να ελέγξουμε    αν το UART είναι
έτοιμο             να             δεχτεί             δεδομένα.
*************************************************************/
    ldwio  r17,4(r8)        /*ελέγχει αν το UART είναι έτοιμο
                                να δεχτεί δεδομένα*/
```

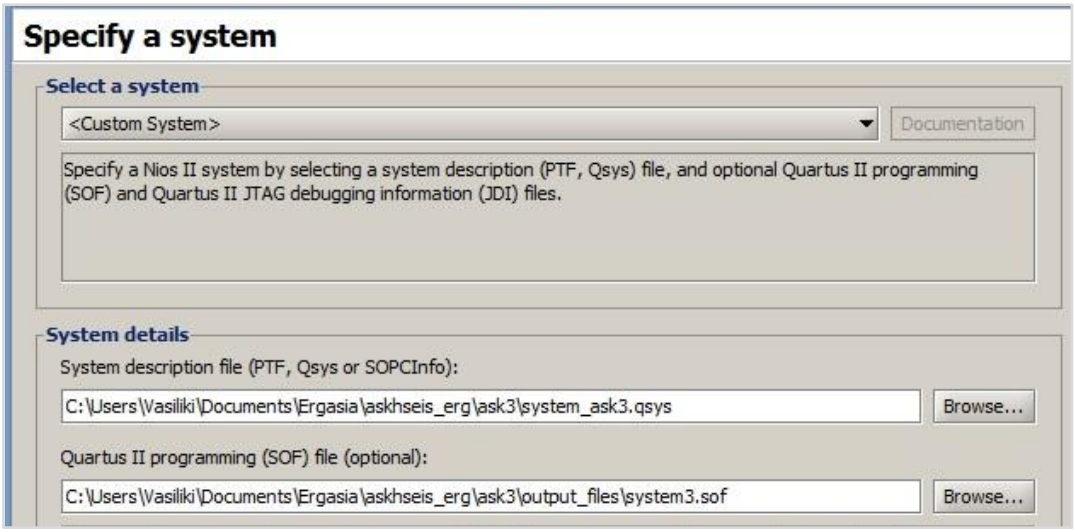
```
    andhi  r17,r17,0xffff   /*ελέγχει αν υπάρχει χώρος για
                           γράψιμο (πεδίο WSPACE μη μηδενικό)*/

    beq r17,r0,PUT_CHAR_LOOP /*συνεχίζει τον έλεγχο μέχρι το
                               UART να είναι έτοιμο να
                               δεχτεί δεδομένα*/

    stwio  r16,0(r8)        /*στέλνει τον χαρακτήρα στην JTAG
                               UART*/
    mov    r18, r0          /*μηδενίζεται ο απαριθμητής r18*/

DELAY_LOOP:                 /*εισάγει την χρονική καθυστέρηση*/
    addi   r18, r18, 1
    andhi  r19, r18, 0x0010
    beq    r19, r0, DELAY_LOOP
    br     PUT_CHAR_LOOP    /*εμφάνιση του χαρακτήρα ξανά*/

.end
```

Αφού έχουμε δημιουργήσει το αρχείο *prog_Z.s* σε γλώσσα assembly, τώρα μπορούμε να περάσουμε στην εκτέλεση της εφαρμογής, αφού πρώτα δημιουργήσουμε το project της εφαρμογής στο Altera Monitor Program.

Επιλέγουμε *File* > *New Project*, όπου ορίζουμε το φάκελο που θα δουλεύουμε και το όνομα του project. [Εικόνα 5.10]

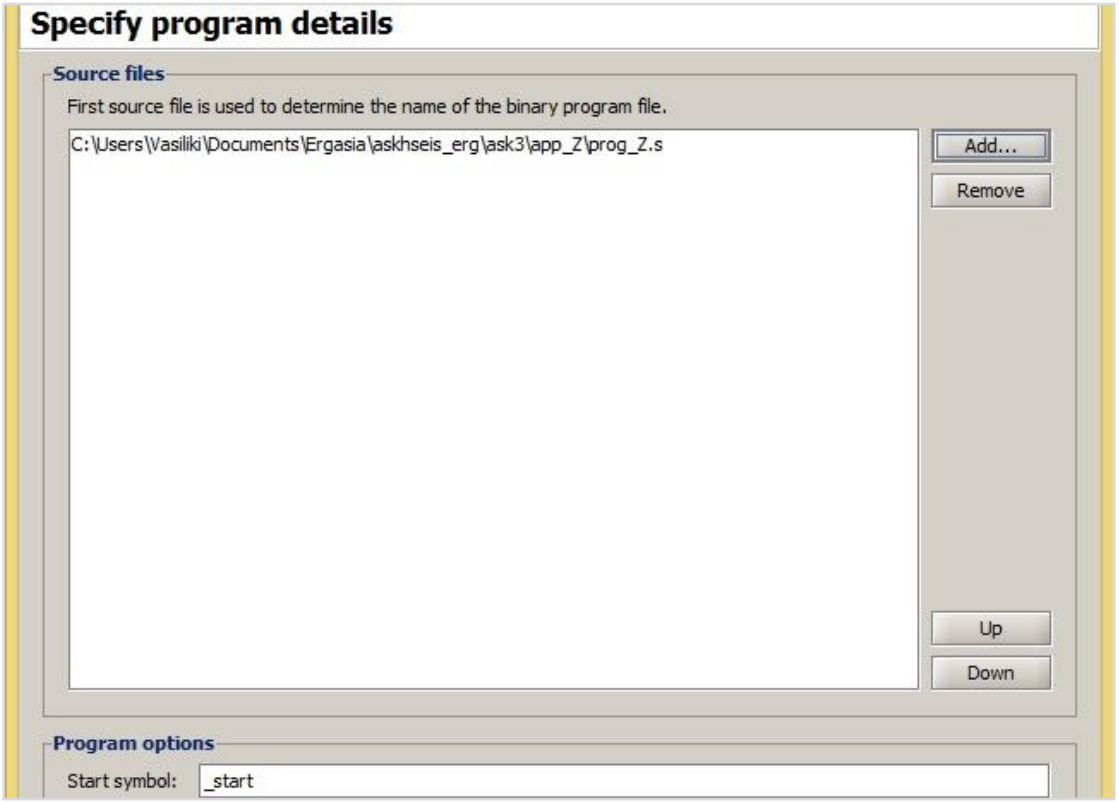
Εικόνα 5.10: Δημιουργία νέου project στο Altera Monitor Program.

Στην επόμενη καρτέλα στο πλαίσιο System: επιλέγω *Custom System* γιατί θέλουμε να δημιουργήσουμε μία δική μας εφαρμογή, στην περιγραφή συστήματος ορίζω το σύστημα που δημιουργήσαμε στο Qsys και το .sof αρχείο που δημιουργήθηκε από το Compilation, όπως φαίνεται στην παρακάτω εικόνα. [Εικόνα 5.12]



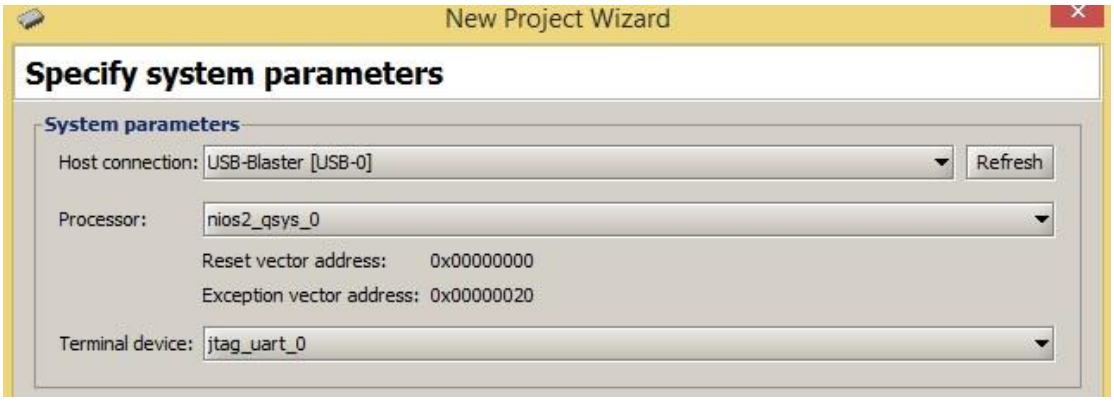
Εικόνα 5.11: Ορισμός συστήματος στο Altera Monitor Program.

Στην επόμενη καρτέλα επιλέγουμε σε τι γλώσσα θα είναι το πρόγραμμά μας. Εδώ συγκεκριμένα χρησιμοποιούμε την γλώσσα assembly. Συνεχίζουμε με την εισαγωγή του αρχείου σε assembly που περιέχει το πρόγραμμα της εφαρμογής μας. [Εικόνα 5.11]



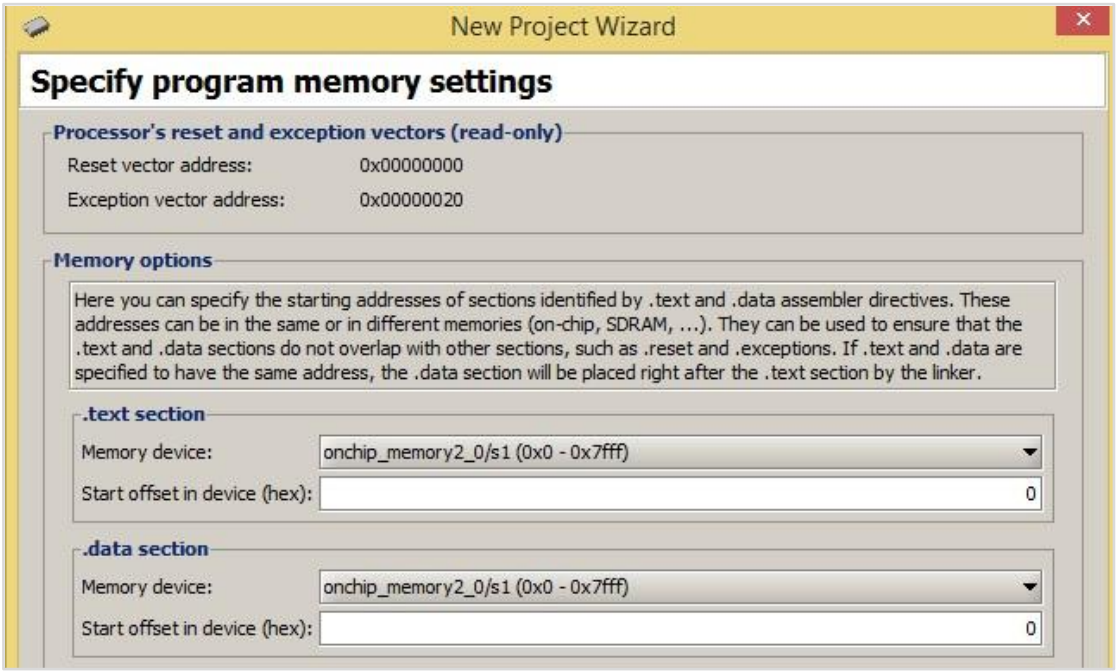
Εικόνα 5.12: Ορισμός του αρχείου που περιέχει το πρόγραμμα της εφαρμογής.

Στην επόμενη καρτέλα προσδιορίζουμε τις παραμέτρους του συστήματος. Όπως βλέπουμε και στην Εικόνα 5.13 προσδιορίζουμε με ποιον τρόπο θα γίνει η σύνδεση host υπολογιστή με την πλακέτα DE2 (συγκεκριμένα μέσω της θύρας usb), στο πλαίσιο Processor επιλέγουμε το nios2_qsys_0 και στο πλαίσιο Terminal Device ορίζουμε το jtag_uart και πατάμε το Next.



Εικόνα 5.13: Προσδιορισμός συστήματος.

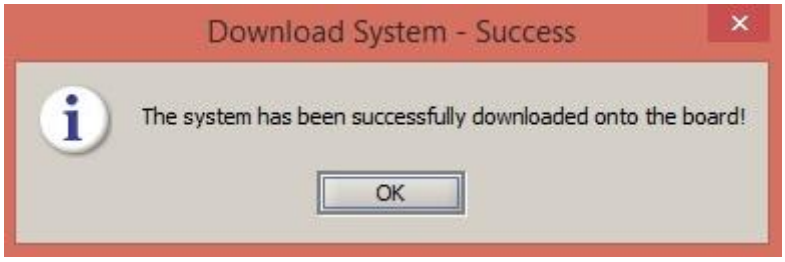
Στην τελευταία καρτέλα γίνεται ο προσδιορισμός και οι ρυθμίσεις της μνήμης. Στην εφαρμογή αυτή δεν χρειάζεται να αλλάξουμε κάτι και αφήνουμε τις προεπιλεγμένες ρυθμίσεις ως έχουν. Πατάμε το Finish για να ολοκληρωθεί η διαδικασία δημιουργίας project. [Εικόνα 5.14]



Εικόνα 5.14: Ολοκλήρωση της δημιουργίας project με τον προσδιορισμό της μνήμης.

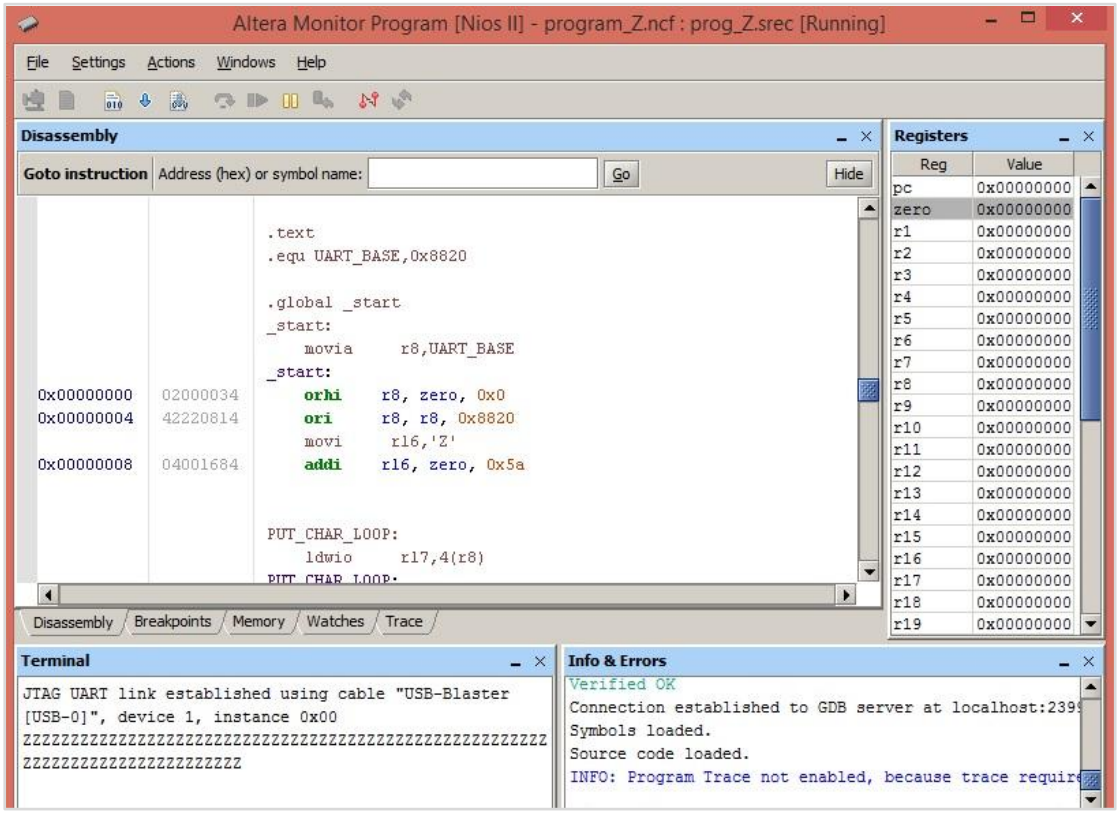
Μόλις ολοκληρώσουμε την παραπάνω διαδικασία εμφανίζεται το μήνυμα της Εικόνας 5.15. Επιλέγουμε Yes και αν μεταφορτωθεί σωστά το σύστημά μας στην πλακέτα τότε εμφανίζεται το μήνυμα της Εικόνας 5.16.

Εικόνα 5.15: Προγραμματισμός του FPGA με το Altera Monitor Program.



Εικόνα 5.16: Επιτυχής μεταφόρτωση του συστήματος στην αναπτυξιακή πλακέτα.

Το επόμενο και τελευταίο βήμα είναι η φόρτωση του προγράμματος σε assembly. Επιλέγουμε *Action > Compile & Load* και αφού επιλέξουμε να εκτελεστεί η εφαρμογή βλέπουμε το αποτέλεσμα της Εικόνας 5.17.



Εικόνα 5.17: Εμφάνιση του χαρακτήρα 'Z' στο τερματικό παράθυρο.

Βλέποντας στην παραπάνω εικόνα την εκτέλεση της εφαρμογής, παρατηρούμε την συνεχόμενη εμφάνιση του χαρακτήρα '*Z*' με μία πολύ μικρή καθυστέρηση ανάμεσα στις διαδοχικές εμφανίσεις λόγω της συνάρτησης DELAY_LOOP στον κώδικα του προγράμματος *prog_Z.s.*

5.2 Δεύτερη Εργαστηριακή Εφαρμογή - Αποστολή Χαρακτήρων στον Nios II

Στην δεύτερη αυτή εφαρμογή, θα δείξουμε ένα παράδειγμα αποστολής χαρακτήρων στον επεξεργαστή Nios II, με σειριακή επικοινωνία, μέσω της διασύνδεσης *JTAG UART*. Θα εμπλουτίσουμε στην ουσία την πρώτη εφαρμογή ζητώντας από την διασύνδεση JTAG_UART να εμφανίσει περισσότερους από ένα χαρακτήρα στο τερματικό παράθυρο.

Ας θυμηθούμε πρώτα την λειτουργία των δύο καταχωρητών (data και control registers) της θύρας JTAG γιατί είναι πολύ σημαντικοί για την δημιουργία του προγράμματος της εφαρμογής. [Πίνακες 5.2, 5.3]

Bits	Όνομα	Πρόσβαση	Περιγραφή
[7:0]	DATA	R/W	Η τιμή για την μεταφορά προς/από τον πυρήνα JTAG. Κατά το γράψιμο, το πεδίο DATA κρατάει έναν χαρακτήρα για να τοποθετηθεί στο FIFO εγγραφής. Κατά την ανάγνωση, το πεδίο DATA κρατάει ένα χαρακτήρα που διαβάζεται από το FIFO ανάγνωσης.
[15]	RVALID	R	Δείχνει εάν το πεδίο DATA είναι έγκυρο. Αν RVALID = 1, το πεδίο DATA είναι έγκυρο, διαφορετικά το πεδίο DATA είναι απροσδιόριστο.
[32:16]	RAVAIL	R	Ο αριθμός των χαρακτήρων που απομένουν στο Read FIFO (μετά την τρέχουσα ανάγνωση).

Πίνακας 5.2: Τα bits του καταχωρητή δεδομένων [21]

Bits	Όνομα	Πρόσβαση	Περιγραφή
0	RE	R/W	Bit ενεργοποίησης διακοπής για διακοπές ανάγνωσης.
1	WE	R/W	Bit ενεργοποίησης διακοπής για διακοπές εγγραφής.
8	RI	R	Δείχνει ότι η διακοπή ανάγνωσης είναι σε εκκρεμότητα.
9	WI	R	Δείχνει ότι η διακοπή εγγραφής είναι σε εκκρεμότητα.
10	AC	R/C	Δείχνει ότι υπήρξε δραστηριότατα JTAG αφού το bit είχε διαγραφεί. Γράφοντας 1 στο AC το κάνει 0.
[32:16]	WSPACE	R	Δείχνει τον αριθμό των κενών που υπάρχουν στο FIFO εγγραφής.

Πίνακας 5.3: Τα bits του καταχωρητή ελέγχου [21]

5.2.1 Διαδικασία σχεδιασμού και υλοποίησης του υλικού της 2ης εργαστηριακής εφαρμογής

Στην συγκεκριμένη εφαρμογή μπορούμε να χρησιμοποιήσουμε το σύστημα που δημιουργήσαμε στο Qsys για την πρώτη εργαστηριακή εφαρμογή, διότι δεν θέλουμε να αλλάξουμε κάτι στο σύστημα, παρά μόνο στο πρόγραμμα της εφαρμογής που είναι σε assembly.

Επομένως, η διαδικασία που θα ακολουθήσουμε είναι ίδια με τις ενότητες 5.1.1 έως και 5.1.5. Τώρα όμως που έχουμε ήδη δημιουργήσει το σύστημά μας δεν χρειάζεται να ξανακάνουμε την διαδικασία δημιουργίας συστήματος.

Άρα, θα συνεχίσουμε με το πρόγραμμα Altera Monitor Program.

5.2.2 Λήψη του συστήματος στο FPGA και εκτέλεση του προγράμματος της εφαρμογής με το Altera Monitor Program

Αρχικά για να μπορέσουμε να κάνουμε ένα νέο project στο Altera Monitor Program, όπως είπαμε ήδη στην προηγούμενη εφαρμογή, πρέπει να δημιουργήσουμε το πρόγραμμα της εφαρμογής σε assembly, το οποίο θα στέλνει

σε επανάληψη χαρακτήρες στον επεξεργαστή Nios II μέσω της διασύνδεσης JTAG UART και θα τους εμφανίζει στο παράθυρο του τερματικού. Θυμίζουμε ότι όταν το WSPACE πεδίο στον καταχωρητή ελέγχου της JTAG UART έχει μια μη μηδενική τιμή τότε το JTAG UART μπορεί να δεχτεί ένα νέο χαρακτήρα, για να αποσταλεί μέσω της σειριακής διασύνδεσης. Για να αποστείλουμε χαρακτήρα στον host υπολογιστή, η εφαρμογή που γράψαμε σε assembly διαβάζει συνεχόμενα μέχρι να είναι διαθέσιμο ένα κενό στον FIFO καταχωρητή. Μόλις ένα κενό είναι διαθέσιμο ο ASCII χαρακτήρας μπορεί να γραφτεί στον καταχωρητή δεδομένων της διασύνδεσης JTAG UART.

Το πρόγραμμα σε assembly παρουσιάζει διαδοχικά τα γράμματα H, e, l, l, o, !, !, ! στο παράθυρο του τερματικού και επαναλαμβάνει την ίδια διαδικασία από την αρχή μέχρι να σταματήσουμε την εκτέλεση της εφαρμογής. Ανάμεσα σε διαδοχικές εμφανίσεις εισάγει μια χρονική καθυστέρηση της τάξης του μισού δευτερολέπτου. Ο κώδικας για το πρόγραμμα assembly της εφαρμογής είναι ο εξής:

```
.include "nios_macros.s"
.text                       /*ακολουθεί εκτελέσιμος κώδικας.*/
.equ UART_BASE,0x8820       /*φόρτωση της διεύθυνσης 0x8820
                            που  αντιπροσωπεύει το JTAG_UART
                            στο UART_BASE.*/

.global _start
_start:
    movia  r8,UART_BASE     /*τοποθέτηση του UART_BASE στον
                            καταχωρητή r8.*/

MY_LOOP:
    movi   r16,'H'          /*τοποθέτηση του γράμματος 'H'
                            στον r16. */
    call   PUT_CHAR         /*καλεί την υπορουτίνα PUT_CHAR
                            και εμφανίζει τον χαρακτήρα. */
    movi   r16,'e'          /*τοποθέτηση του 'e' στον r16. */

    call   PUT_CHAR
    movi   r16,'l'          /*τοποθέτηση του 'l' στον r16. */
    call   PUT_CHAR
    movi   r16,'l'          /*τοποθέτηση του 'l' στον r16. */
    call   PUT_CHAR
    movi   r16,'o'          /*τοποθέτηση του 'o' στον r16. */
```

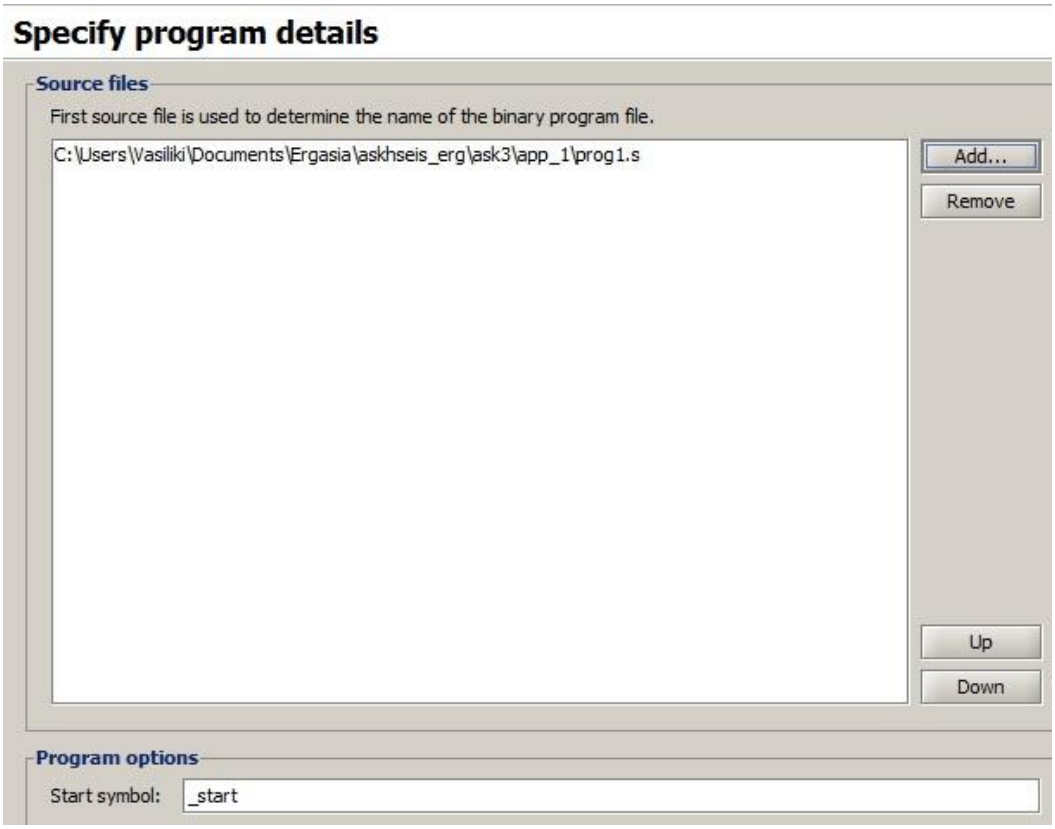
```
    call   PUT_CHAR
    movi   r16,'!'          /*τοποθέτηση του '!' στον r16. */
    call   PUT_CHAR
    movi   r16,'!'          /*τοποθέτηση του '!' στον r16. */
    call   PUT_CHAR
    movi   r16,'!'          /*τοποθέτηση του '!' στον r16. */
    call   PUT_CHAR
    br MY_LOOP              /*εκτελεί από την αρχή την MY_LOOP.*/

.global PUT_CHAR
PUT_CHAR:
/* Στον παρακάτω κώδικα προσθέτουμε τα περιεχόμενα του r8 με
    το 4 γιατί ο καταχωρητής ελέγχου είναι 4 bytes ψηλότερα
    από τον καταχωρητή δεδομένων, και εμείς θέλουμε να
    χρησιμοποιήσουμε τον καταχωρητή ελέγχου για να ελέγξουμε
    αν το UART είναι έτοιμο να δεχτεί δεδομένα.*/
    ldwio  r17,4(r8)        /*ελέγχει αν το UART είναι
                            έτοιμο να δεχτεί δεδομένα.*/

    andhi  r17,r17,0xffff   /*ελέγχει αν υπάρχει χώρος για
                            γράψιμο (πεδίο WSPACE μη
                            μηδενικό).*/
    beq    r17,r0,PUT_CHAR  /*συνεχίζει τον έλεγχο μέχρι το
                            UART να είναι έτοιμο να
                            δεχτεί δεδομένα.*/
    stwio  r16,0(r8)        /*αποθηκεύει το χαρακτήρα του r16
                            στην διεύθυνση r8 δηλαδή
                            στην UART BASE*/
    mov    r18, r0          /*μηδενίζεται ο απαριθμητής r18.*/

DELAY_LOOP:                 /*εισάγει την χρονική καθυστέρηση*/
    addi   r18, r18, 1
    andhi  r19, r18, 0x0010
    beq    r19, r0, DELAY_LOOP
    ret

.end
```

Αφού δημιουργήσαμε το αρχείο prog1.s τώρα μπορούμε να κάνουμε ένα νέο project στο Altera Monitor Program. (File -> New Project), όπου ορίζουμε το φάκελο που θα δουλεύουμε και το όνομα του project. [Εικόνα 5.10]

Εικόνα 5.18: Δημιουργία project στο Altera Monitor Program

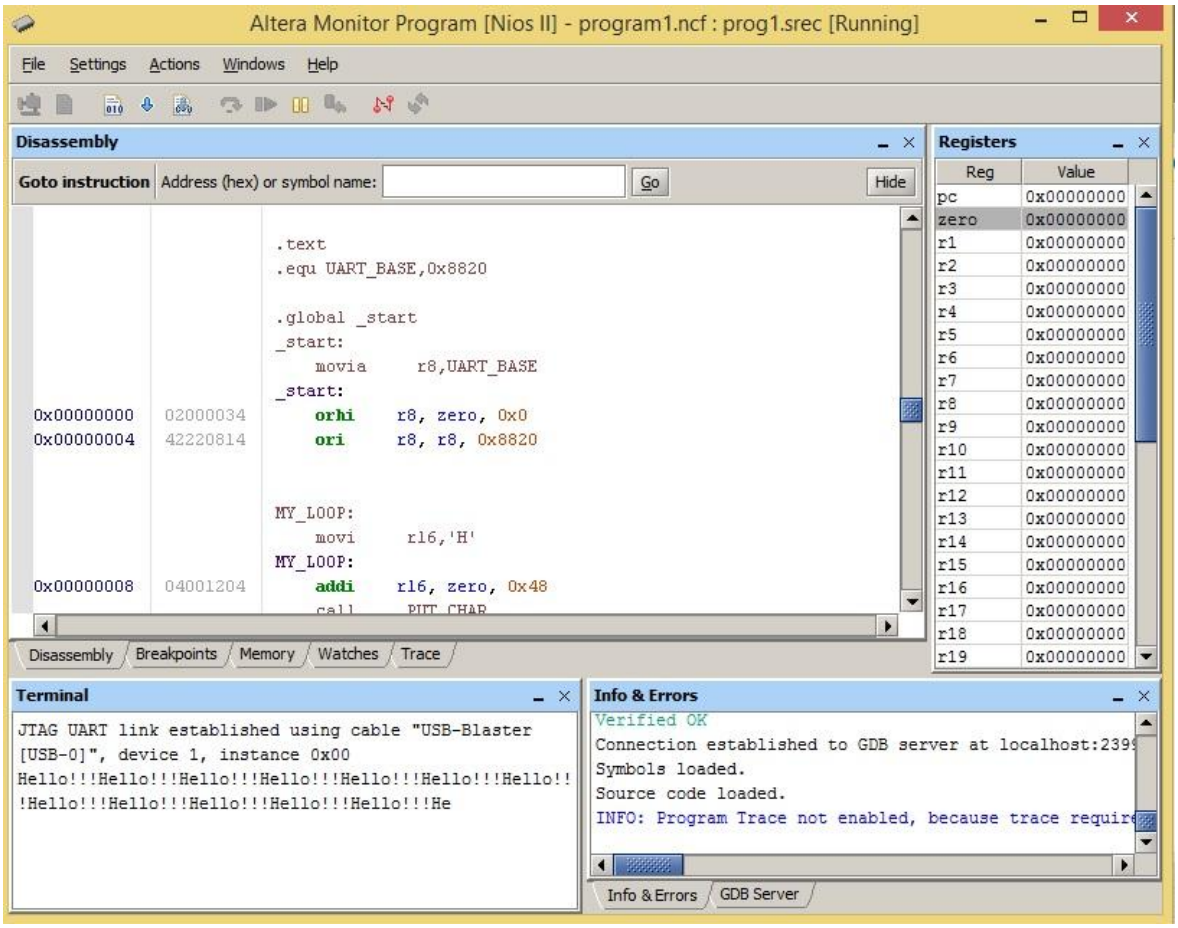
Στην επόμενη καρτέλα γίνεται ο προσδιορισμός του συστήματος και του αρχείο .sof που θα χρησιμοποιήσουμε. [**Εικόνα 5.11**]

Στην επόμενη καρτέλα επιλέγουμε σε τι γλώσσα θα είναι το πρόγραμμά μας. Επιλέγουμε την γλώσσα assembly. Στην επόμενη καρτέλα εισάγουμε στο project το αρχείο .s με το πρόγραμμα της εφαρμογής. [Εικόνα 5.19]



Εικόνα 5.19: Εισαγωγή του προγράμματος της εφαρμογής

Στις δύο επόμενες καρτέλες δεν αλλάζουμε κάτι και πατάμε finish [**Εικόνα 5.13, Εικόνα 5.14**]. Τότε εμφανίζεται ένα pop-up box που μας ρωτάει αν θέλουμε να φορτώσουμε το σύστημά μας στην πλακέτα DE2. Μετά την φόρτωση στο fpga επιλέγω *Action > Compile & Load* ώστε να εκτελεστεί η εφαρμογή μας.



Εικόνα 5.20: Εμφάνιση των χαρακτήρων "Hello!!!" στο τερματικό παράθυρο του Altera Monitor Program.

Παρατηρούμε την εμφάνιση των χαρακτήρων που θέσαμε με τη σειρά με καθυστέρηση μισού δευτερολέπτου, Hello!!! σε συνεχόμενη επανάληψη, στο παράθυρο του τερματικού [Εικόνα 5.20].

5.3 Τρίτη Εργαστηριακή Εφαρμογή - Αποστολή και λήψη δεδομένων στον Nios II

Σε αυτή την εργαστηριακή άσκηση θα μελετήσουμε την αποστολή αλλά και την λήψη δεδομένων από το host υπολογιστή προς τον Nios II επεξεργαστή μέσω της διασύνδεσης JTAG UART και αντίθετα. Δηλαδή, ο κώδικας θα διαβάζει τα δεδομένα χαρακτήρων που έχουν ληφθεί από την JTAG UART, και τυπώνει αυτά τα δεδομένα πίσω στην UART για μετάδοση. Όταν χρησιμοποιούμε το Altera Monitor Program, με την εκτέλεση της εφαρμογής όποιος χαρακτήρας του πληκτρολόγιου πληκτρολογείται μέσα στο τερματικό παράθυρο θα τυπώνεται πίσω προκαλώντας τον χαρακτήρα να εμφανιστεί στο τερματικό παράθυρο, ενώ κανονικά δεν μπορούμε να πληκτρολογήσουμε μέσα στην περιοχή του τερματικού παράθυρου. Πρόκειται, δηλαδή, για εφαρμογή *γραφομηχανής*.

5.3.1 Διαδικασία σχεδιασμού και υλοποίησης του υλικού της 3ης εργαστηριακής εφαρμογής

Και σε αυτή την εφαρμογή μπορούμε να χρησιμοποιήσουμε το σύστημα που δημιουργήσαμε στο Qsys για την πρώτη και την δεύτερη εργαστηριακή εφαρμογή, αλλά και γενικά οποιοδήποτε αρχείο δημιουργήθηκε στο project του Quartus II, διότι δεν θέλουμε να αλλάξουμε κάτι στο σύστημα, παρά μόνο στο πρόγραμμα της εφαρμογής που είναι σε assembly.

Επομένως, η διαδικασία που θα ακολουθήσουμε είναι ίδια με τις ενότητες 5.1.1 έως και 5.1.5. Άρα, θα συνεχίσουμε με το πρόγραμμα Altera Monitor Program.

5.3.2 Altera Monitor Program: Λήψη του συστήματος στο FPGA και εκτέλεση του προγράμματος της εφαρμογής

Πρέπει να δημιουργήσουμε το project της εφαρμογής του Altera Monitor Program και το assembly πρόγραμμα της εφαρμογής.

Ο κώδικας για το πρόγραμμα assembly της εφαρμογής είναι ο εξής:

```
.include "nios_macros.s"
.text
.equ UART_BASE,0x8820       /*Φόρτωση της διεύθυνσης 0x8820 που
                            αντιπροσωπεύει τη βασική διεύθυνση
                            του JTAG_UART. */

.global _start
_start:
    movia  r8,UART_BASE     /*Τοποθέτηση του UART_BASE στον
                            καταχωρητή r8.*/

GET_CHAR_LOOP:
    ldwio  r17,0(r8)        /*Ελέγχει αν το UART έχει νέα
                            έγκυρα δεδομένα.*/

    andi   r20,r17, 0x8000 /*Διαβάζεται ο χαρακτήρας, ο 0x8000
                            αντιπροσωπεύει το RV (bit15 του
                            καταχωρητή δεδομένων.*/

    beq    r20,r0,GET_CHAR_LOOP /*Συνεχίζει τον έλεγχο μέχρι να
                                έχει η UART δεδομένα.*/

    andi   r16,r17,0x00ff   /*Απομονώνει το πεδίο δεδομένων.*/

PUT_CHAR_LOOP:
    Ldwio  r17,4(r8)        /* Γίνεται έλεγχος στον καταχωρητή
                            ελέγχου αν το UART είναι έτοιμο να
                            λάβει δεδομένα.*/

    Andhi  r17,r17,0xffff   /*Ελέγχει αν υπάρχει χώρος για
                            γράψιμο.*/

    beq    r17,r0,PUT_CHAR_LOOP /*Συνεχίζει τον έλεγχο μέχρι
                                που το UART μπορεί να δεχτεί
                                δεδομένα προς αποστολή.*/
    stwio  r16,0(r8)  /*Αποθήκευση του χαρακτήρα στην UART.*/

NO_NEW_LETTER:
    br     GET_CHAR_LOOP    /*Παίρνει τον επόμενο χαρακτήρα.*/

.end
```

Το JTAG UART μπορεί να δεχτεί ASCII χαρακτήρες από το παράθυρο του τερματικού, όπως και να γράψει. Το RVALID bit, b_{15}, στον καταχωρητή δεδομένων, δείχνει αν η τιμή στο πεδίο δεδομένων είναι ένας έγκυρος λαμβανόμενος χαρακτήρας. Αν περισσότεροι χαρακτήρες περιμένουν ακόμη για να διαβαστούν, το πεδίο RAVAIL θα έχει μια μη μηδενική τιμή. Το πρόγραμμα διαβάζει κάθε

χαρακτήρα που λαμβάνεται από το JTAG UART από τον host υπολογιστή και τον εμφανίζει στο παράθυρο τερματικού του προγράμματος Altera Monitor Program. Χρησιμοποιείται η μέθοδος polling για να αποφασιστεί αν ένας νέος χαρακτήρας είναι διαθέσιμος από το JTAG UART. Όταν υπάρχει νέος χαρακτήρας, απομονώνεται το πεδίο δεδομένων στον καταχωρητή δεδομένων, και αποθηκεύεται στον καταχωρητή r16.

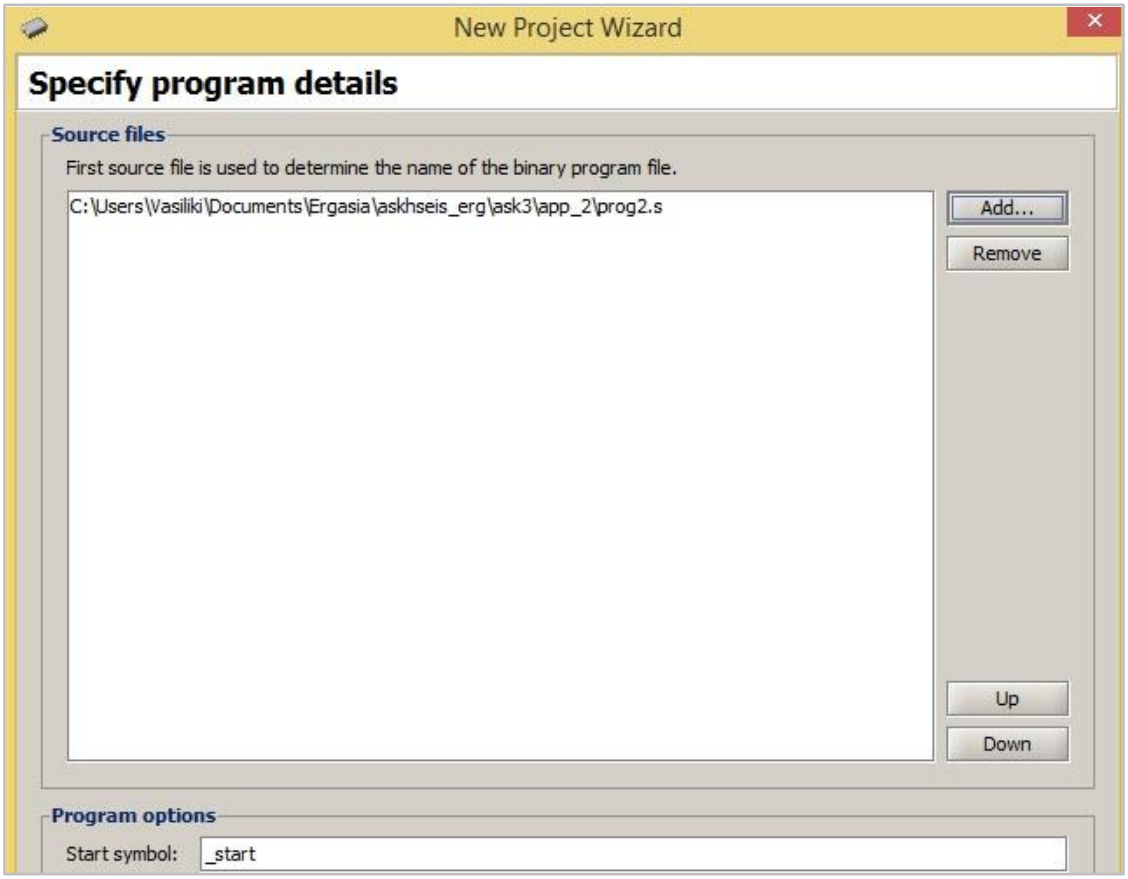
Στην συνέχεια, η εφαρμογή ελέγχει αν η UART μπορεί να στείλει δεδομένα, αν δηλαδή υπάρχει κενό στον FIFO register για αποστολή. Όταν υπάρχει κενό, αποθηκεύει το περιεχόμενο του r16 (που περιέχει τον χαρακτήρα που μόλις λήφθηκε από την UART) στην βασική διεύθυνση της UART, προς αποστολή στον υπολογιστή. Έτσι, ο χαρακτήρας επιστρέφει στην κονσόλα του τερματικού.

Δημιουργία Project στο Altera Monitor Program

Αφού δημιουργήσαμε τον κώδικα του προγράμματος assembly της εφαρμογής μπορούμε να δημιουργήσουμε και ένα καινούριο project στο Altera Monitor Program. Επιλέγοντας, όπως είπαμε, *File > New Project*, ορίζουμε όπως στην Εικόνα 5.21 τον φάκελο εργασίας και το όνομα του project.

Εικόνα 5.21: 1ο βήμα δημιουργίας project.

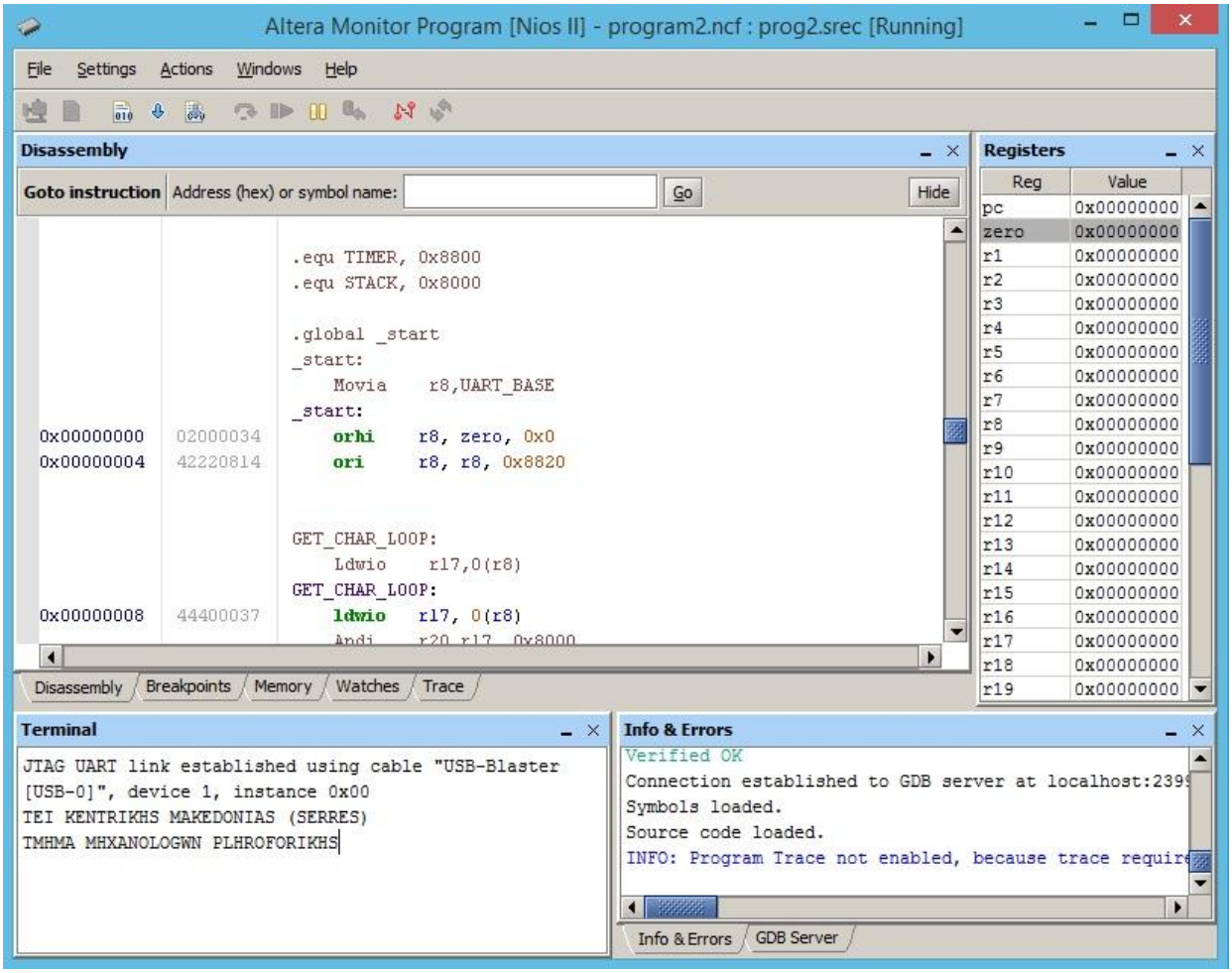
Στην επόμενη καρτέλα γίνεται ο προσδιορισμός του συστήματος και του αρχείο .sof που θα χρησιμοποιήσουμε [**Εικόνα 5.11**]. Στη συνέχεια, επιλέγουμε την γλώσσα assembly ως γλώσσα του προγράμματος της τρίτης εφαρμογής. Στην επόμενη καρτέλα εισάγουμε στο project το αρχείο .s με το πρόγραμμα της εφαρμογής.



Εικόνα 5.22: Ορισμός του προγράμματος της τρίτης εφαρμογής.

Στις δύο επόμενες καρτέλες δεν αλλάζουμε κάτι και πατάμε finish [**Εικόνα 5.13, Εικόνα 5.14**]. Μετά την φόρτωση του προγράμματος στο fpga επιλέγω *Action > Compile & Load* ώστε να εκτελεστεί η εφαρμογή μας. [Εικόνα 5.23]

Παρατηρούμε ότι μπορούμε να πληκτρολογήσουμε στο παράθυρο του τερματικού και να εμφανιστεί κάθε χαρακτήρας που πληκτρολογούμε. Αυτό συμβαίνει διότι, σύμφωνα με τον κώδικα, διαβάζει τα δεδομένα χαρακτήρων που έχουν ληφθεί από την JTAG UART, και τυπώνει αυτά τα δεδομένα πίσω στην UART για μετάδοση.



Εικόνα 5.23: Η φόρτωση και εκτέλεση του προγράμματος "γραφομηχανής" στο Altera Monitor Program.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην εργασία αυτή περιγράφηκαν λεπτομερώς οι διαδικασίες σχεδιασμού λογισμικού και υλικού ενσωματωμένων συστημάτων. Οι βασικοί στόχοι επιτεύχθηκαν μέσω της δημιουργίας του συστήματος των τελικών εφαρμογών σε host υπολογιστή και την επίτευξη της σειριακής επικοινωνίας μεταξύ αυτού του υπολογιστή και του επεξεργαστή Nios II μέσω της διασύνδεσης JTAG UART. Όπως επίσης επιτεύχθηκε και εξοικείωση με την σχεδίαση υλικού και λογισμικού με τα αντίστοιχα λογισμικά (Quartus, Qsys, Altera Monitor Program).

Ο Nios II είναι ένας επεξεργαστής ο οποίος είναι κατάλληλος και για εκπαιδευτική χρήση. Η πτυχιακή αυτή, μέσω των παραδειγμάτων των προγραμμάτων που χρησιμοποιήθηκαν, και της ανάλυσης των βημάτων για την υλοποίηση απλών συστημάτων που περιγράφηκαν μπορεί να χρησιμοποιηθεί για ακαδημαϊκούς εργαστηριακούς σκοπούς, αφού ακόμα και ένας αρχάριος ενδιαφερόμενος θα μπορεί να σχεδιάσει ένα σύστημα με τον Nios II, να το υλοποιήσει με τη βοήθεια του λογισμικού Quartus II και τέλος να προγραμματίσει την on chip memory του FPGA ώστε να εκτελέσει μια συγκεκριμένη εφαρμογή στον επεξεργαστή Nios II. Επίσης, χρησιμοποιώντας αυτή τη πτυχιακή γίνεται ευκολότερη η επέκταση του συστήματος για χρήση σε πολυπλοκότερες εφαρμογές. Ένα παράδειγμα επέκτασης θα ήταν η αλλαγή του τρόπου επικοινωνίας του host υπολογιστή με το FPGA, όπως παράλληλη επικοινωνία.

Κλείνοντας νιώθω την ανάγκη και την υποχρέωση να ευχαριστήσω τον καθηγητή μου και επιβλέποντα της πτυχιακής αυτής εργασίας, Δρ. Σπυρίδων Καζαρλή, τόσο για την καθοδήγηση κατά τη διάρκεια της εκπόνησής της, αλλά και για όλες τις γνώσεις που μου μεταλαμπάδευσε κατά τη διάρκεια των σπουδών μου.

Θα ήταν άδικο βέβαια να μην ευχαριστήσω και τον Δρ. Ιωάννη Καλόμοιρο, για την πολύτιμη βοήθειά του όποτε και αν την χρειάστηκα, του οποίου τα συγγράμματα με βοήθησαν αρκετά στην εκπόνηση της πτυχιακής μου εργασίας.

Αναφορές

1. http://en.wikipedia.org/wiki/Apollo_Guidance_Computer
2. http://en.wikipedia.org/wiki/D-17B
3. http://en.wikipedia.org/wiki/Microprocessor
4. Υπουργείο Εξωτερικών http://agora.mfa.gr/, "Ο κλάδος των ενσωματωμένων συστημάτων - embedded systems - στη Γερμανία", Ντίσελντορφ, Ιούνιος 2011
5. Ιωάννης Καλόμοιρος, "Εισαγωγή στην γλώσσα VHDL", Έκδοση 1.2, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών, 2012
6. Καθ. Απόστολος Δόλλας, "ΗΡΥ 401 Ενσωματωμένα Συστήματα Μικροεπεξεργαστών", Πανεπιστημιακές Σημειώσεις, Έκδοση 1.1, Πολυτεχνείο Κρήτης, Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών, Σεπτέμβριος 2006
7. Μαρία Κ. Μιχαήλ, "ΗΜΥ - 210: Σχεδιασμός Ψηφιακών Συστημάτων", Πανεπιστήμιο Κύπρου, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
8. Ιωάννης Καλόμοιρος, "Διαφάνειες για το μάθημα Προηγμένα Ψηφιακά Συστήματα", Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών, 2008
9. http://en.wikipedia.org/wiki/Complex_programmable_logic_device
10. http://saaubi.people.wm.edu/TeachingWebPages/Physics351_Fall2008/Week0/Physics351_Fall2008_ad.html
11. Carl Hamacher – Zvonko Vranesic – Naraig Manjikian, "Computer Organization and Embedded Systems", Sixth Edition, The McGraw-Hill Companies, 2012
12. http://www.altera.com
13. http://www.xilinx.com
14. Altera, "Nios II Classic Processor Reference Guide", 02/04/2015
15. Altera, "Introduction to the Altera Nios II Soft Processor", Ιούλιος 2010

16. Altera, "DE2 Development and Educational User Manual", 2012

17. Altera, "Quartus II Introduction", Σεπτέμβριος 2010

18. https://www.altera.com/products/design-software/embedded-software-developers/nios-ii-eds.html

19. Altera, "Introduction to the Altera Qsys System Integration Tool", Μάιος 2013

20. Altera, "Laboratory Exercise 1 for Quartus II", 2011

21. Altera, "Embedded Peripherals IP User Guide", 12-06-2015

Βιβλιογραφία

Η εταιρεία Altera στην ιστοσελίδα της (www.altera.com) παρέχει συνοδευτικό και εκπαιδευτικό υλικό, το οποίο και χρησιμοποίησα, για ένα μεγάλο μέρος των θεμάτων που ανέπτυξα στην εργασία μου στην αγγλική γλώσσα. Ενδεικτικά:

- DE2 Development and Educational User Manual, 2012
- SOPC Builder User Guide, Δεκέμβριος 2010
- Quartus II Introduction, Μάιος 2013
- Introduction to the Altera Qsys System Integration Tool, Μάιος 2013
- http://www.altera.com/products/software/quartus-ii/subscription-edition/qsys/qts-qsys.html
- https://www.altera.com/products/design-software/embedded-software-developers/nios-ii-eds.html
- Altera Monitor Program Tutorial, Μάιος 2011
- Nios II Classic Processor Reference Guide, 02-04-2015
- Introduction to the Altera Nios II Soft Processor, Ιούλιος 2010
- Embedded Peripherals IP User Guide, 12-06-2015
- JTAG UART Core, Νοέμβριος 2009

Επιπλέον μελέτησα και άντλησα υλικό από τις παρακάτω πηγές για την συγγραφή της πτυχιακής εργασίας:

Carl Hamacher – Zvonko Vranesic – Naraig Manjikian, "Computer Organization and Embedded Systems", Sixth Edition, The McGraw-Hill Companies, 2012

Ιωάννης Καλόμοιρος, “Διαφάνειες για το μάθημα Προηγμένα Ψηφιακά Συστήματα”, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών, 2008

Ιωάννης Καλόμοιρος, “Σημειώσεις εργαστηρίου Προηγμένων Ψηφιακών Συστημάτων”, Έκδοση 2η, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών, 2010

Ιωάννης Καλόμοιρος, “Ενσωματωμένα συστήματα για εφαρμογές πραγματικού χρόνου”, διαφάνεια από την θεωρία *Αρχές Προγραμματισμού Πραγματικού Χρόνου*, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών

Ιωάννης Καλόμοιρος, “Εισαγωγή στην γλώσσα VHDL”, Έκδοση 1.2, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Σερρών, Τμήμα Πληροφορικής και Επικοινωνιών, 2012

---

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

---

http://en.wikipedia.org/wiki/Programmable_logic_device

Καθ. Απόστολος Δόλλας, “ΗΡΥ 401 Ενσωματωμένα Συστήματα Μικροεπεξεργαστών”, Πανεπιστημιακές Σημειώσεις, Έκδοση 1.1 (τελευταία ενημέρωση: Σεπτέμβριος 2006), Πολυτεχνείο Κρήτης, Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών, Εργαστήριο Μικροεπεξεργαστών και Υλικού

Μαρία Κ. Μιχαήλ, "ΗΜΥ - 210: Σχεδιασμός Ψηφιακών Συστημάτων", Πανεπιστήμιο Κύπρου, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Υπουργείο Εξωτερικών http://agora.mfa.gr/, “Ο κλάδος των ενσωματωμένων συστημάτων - embedded systems - στη Γερμανία”, Ντίσελντορφ, Ιούνιος 2011
