Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή στα Πληροφοριακά Συστήματα

Ανάπτυξη Υβριδικού Συστήματος με Χρήση Συσκευής FPGA για την Επεξεργασία Δεδομένων με Γλώσσα SQL

Ιωάννης Α. Βενετικίδης

Επιβλέπων Καθηγητές
Βασίλειος Βασσάλος
Αναπληρωτής Καθηγητής ΟΠΑ

Μιχάλης Ψαράκης
Επίκουρος Καθηγητής Πα.Πει

Αύγουστος 2012

v.3 – 10.10.2012

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Ανάπτυξη Υβριδικού Συστήματος με Χρήση Συσκευής FPGA για την Επεξεργασία Δεδομένων με Γλώσσα SQL

Ιωάννης Α. Βενετικίδης

Επιβλέπων Καθηγητές

Βασίλειος Βασσάλος
Αναπληρωτής Καθηγητής ΟΠΑ

Μιχάλης Ψαράκης
Επίκουρος Καθηγητής Πα.Πει

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Αύγουστος 2012

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή, ασχολείται με την τεχνολογία των συσκευών FPGA που αποτελούν μια πολύ σπουδαία εξέλιξη στο τομέα των ενσωματωμένων υπολογιστικών συστημάτων. Καταπιάνεται με μετασχηματισμούς επερωτήσεων από τη γλώσσα SQL σε μορφή γλώσσας περιγραφής υλικού VHDL. Για το σκοπό αυτό χρησιμοποιήθηκε το εργαλείο σχεδίασης CAD ISE και η συσκευή FPGA Spartan 3E της εταιρίας Xilinx. Μέσω της υλοποίησης και μέσω της προσομοίωσης των επερωτήσεων, παράχθηκαν αποτελέσματα ιδιαίτερα ενθαρρυντικά ως προς την αποδοτικότητα και την ευχρηστία των συσκευών FPGA. Επίσης, καθορίζεται ο τρόπος και μεθοδολογία σχεδίασης τέτοιων κυκλωμάτων με πολύ αναλυτικά βήματα, καθιστώντας την μεταπτυχιακή διατριβή ως ένα πλήρη οδηγό χρήσης για μελλοντικές αντίστοιχες προσπάθειες. Η έρευνα που υλοποιήθηκε, θα συμβάλει ιδιαίτερα στην εξέλιξη των μη παραδοσιακών τρόπων επεξεργασία δεδομένων.

Συγκεκριμένα, αναδείχθηκε η χρήση των συσκευών FPGA ως ένα εργαλείο συν-επεξεργασίας δεδομένων για άλλα μεγαλύτερα πληροφοριακά συστήματα, αλλά και ως ανεξάρτητη συσκευή επεξεργασίας δεδομένων. Αρχικά, έγινε μια πρώτη ανασκόπηση στις πράξης της σχεσιακής άλγεβρας όπως είναι αυτή της Επιλογής (SELECT), της Προβολής (PROJECT) και της Συνένωσης (JOIN). Εν συνεχεία, δημιουργήθηκαν οι τέσσερις επερωτήσεις Q1, Q2, Q3 και Q4 και δόθηκε η αναπαράσταση τους στη τρέχουσα τεχνολογία βάσεων δεδομένων, δηλαδή στη γλώσσα επερωτήσεων SQL. Έγινε η συσχέτισή των επερωτήσεων με τις πράξεις της σχεσιακής άλγεβρας και τέλος μετασχηματίστηκαν στη μορφή της γλώσσας περιγραφής υλικού VHDL, η οποία αναγνωρίζεται από τις συσκευές FPGA. Οι επερωτήσεις υλοποιήθηκαν ως λογικά κυκλώματα και προσομοιώθηκαν από το εργαλείο σχεδίασης CAD ISE, για να μετρηθεί η απόδοση τους αλλά και η αποτελεσματικότητα τους στην διεκπεραίωση ερωτημάτων. Όλες οι προσομοιώσεις αναλύθηκαν και παρουσιάστηκαν σε μορφή κυματομορφών και διαγραμμάτων. Τέλος, έγινε μια ανασκόπηση των αποτελεσμάτων και παρήχθησαν τελικές παρατηρήσεις και συμπεράσματα.

Τα αποτελέσματα της ερευνητικής προσπάθειας, είναι ιδιαίτερα ενθαρρυντικά και αποτελούν μια πρώτη προσέγγιση στον μη παραδοσιακό τρόπο επεξεργασίας δεδομένων, με χρήση της τεχνολογίας FPGA. Οι δυνατότητες που διανοίγονται για μελλοντικές επεκτάσεις είναι πάρα πολλές και ενδιαφέρουσες.

Summary

This thesis, deals with the technology of FPGA devices which are a very important development in the field of embedded computing systems. Deals with transformations of SQL queries in VHDL, a hardware description language. For this purpose used the design tool CAD ISE and a device FPGA Spartan 3E from Xilinx company. Through implementation as well as simulation of queries, we generated very encouraging results in terms of efficiency and usability FPGA's devices. In addition, it was determined the manner and design methodology of such circuits with very detailed steps, indicating our research a complete guide for future similar efforts. The implemented inquiry will greatly contribute to the development of non-traditional data processing.

Specifically, exhibited the use of FPGA devices as a tool for co-processing of data for other largest information systems, as well as an independent data processing device. Firstly, it was g;iven a retrospection on the operations of relational algebra as this of selection (SELECT), Projection (PROJECT) and Junction (JOIN). Subsequently, it was created four queries Q1, Q2, Q3 and Q4 and was given representation in the current database technology, namely the query language SQL. The queries was correlated to the operations of the relational algebra and finally transformed to the form of hardware description language VHDL, which is recognized by the devices FPGA. The queries implemented as logic circuits and simulated by the tool design CAD ISE for measuring their performance and efficiency in processing queries. All simulations were analyzed and presented in the form of diagrams and waveforms. Finally, there was a review of the results and presented final remarks and conclusions.

The results of this research effort are very encouraging and represent a first approach to the non-traditional data processing by the use of FPGA technology. The possibilities opened up for future expansions are numerous and interesting.

Ευχαριστίες

Η ετυμολογία της λέξης «Ευχαριστώ», σημαίνει ότι δείχνω σε κάποιον ότι τον ευγνωμονώ για κάτι που μου έκανε ή που μου έδωσε. θα ήθελα, λοιπόν, να αποδώσω αυτόν τον όρο στους ανθρώπους που υπήρξαν βοηθοί και συμπαραστάτες, τόσο στην υλοποίηση της εν λόγω μεταπτυχιακής διατριβής όσο και στη διάρκεια της φοίτησης μου στο Ανοικτό Πανεπιστήμιο Κύπρου.

Αρχικά, θα ήθελα να ευχαριστήσω τους δύο επιβλέποντες καθηγητές μου, τον κ. Βασσάλο Βασίλη αναπληρωτή καθηγητή στο Οικονομικό Πανεπιστήμιο Αθηνών (ΟΠΑ) και τον κ. Ψαράκη Μιχάλη επίκουρο καθηγητή στο Πανεπιστήμιο Πειραιώς (Πα.Πει). Η εμπιστοσύνη που μου επέδειξαν, με βοήθησε στην διεκπεραίωση της μεταπτυχιακής διατριβής, η οποία είχε ερευνητικό χαρακτήρα. Αμφότεροι, με ενθάρρυναν και με παρότρυναν να ασχοληθώ με το αντικείμενο της μη παραδοσιακής επεξεργασίας δεδομένων. Αποτέλεσαν συμπαραστάτες και συνεργάτες στην επίλυση κρίσιμων και στρυφνών προβλημάτων, για την επίλυση των οποίων οι γνώσεις τους ήταν υποστηρικτικές και καθοριστικές. Η επιμονή τους για περαιτέρω εμβάθυνση στο συγκεκριμένο ερευνητικό αντικείμενο, με έκαναν ικανότερο και με εμπλούτισαν με περισσότερες γνώσεις. Θα ήθελα να τους ευχαριστήσω θερμά!

Ένα μεγάλο ευχαριστώ στους γονείς μου και τα αδέρφια μου που με ενθάρρυναν ψυχολογικά και συνεισφέρανε οικονομικά, για συνέχιση των σπουδών μου σε αυτήν την ηλικία. Ευχαριστώ πολύ την μητέρα μου Ελένη , τον πατέρα μου Αβραάμ και τα αδέλφια μου Ανέστη, Χαράλαμπο, Παναγιώτη και Σοφία!

Ευχαριστώ το φίλο και συνάδελφο Χρήστο Φιλιππόπουλο για την αμέριστη συμπαράσταση και βοήθεια που είχα σε όλη την διάρκεια της φοίτησης μου στο Ανοικτό Πανεπιστήμιο Κύπρου. Η επιμονή του στα θέματα επίλυσης εργασιών με έκαναν να είμαι ποιο υπεύθυνος αλλά και πιο απαιτητικός, τόσο με τον εαυτό μου όσο και με τη γνώση που λαμβάνω. Σε αυτά τα 4 χρόνια αγαπητέ Χρήστο, δε βρήκα απλός έναν φίλο αλλά έναν πάρα πολύ καλό φίλο και εύχομαι γιατί όχι, έναν μελλοντικό συνεργάτη. Να είσαι πάντα καλά!

Ευχαριστώ το φίλο και συνεργάτη Μάκη Φωτιάδη, για τη συνεργασία μας σε όλη την διάρκεια της φοίτησης μου στο Ανοικτό Πανεπιστήμιο Κύπρου. Οι συζητήσεις μας και οι συνεργασίες μας με βοήθησαν στην επίλυση αρκετών εργασιών. Ευχαριστώ Μάκη!

Τέλος θα ήθελα να ευχαριστήσω, τελευταία στη λίστα αλλά πρώτη στην καρδιά μου, την αγαπημένη μου Κική που με βοήθησε τόσο με τη συγγραφή ενοτήτων του 2ου Κεφαλαίου όσο και με τη σύνταξη των προτάσεων. Σ' ευχαριστώ και σ' αγαπώ Κίκα!

Ευχαριστείς πάντα αυτούς που σε ευχαριστούν, και στην δική μου περίπτωση με ευχαρίστησαν αρκετοί άνθρωποι. Τους ευχαριστώ όλους από καρδίας!

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η ραγδαία ζήτηση αλλά και ανάπτυξη των πληροφορικών και πληροφοριακών συστημάτων τη δεκαετία του 90, οδήγησε αναπόφευκτα στην εξέλιξη της επιστήμης των υπολογιστών τόσο σε θεωρητικό επίπεδο όσο και σε πρακτικό. Από την εποχή των τεράστιων (σε μέγεθος) υπολογιστικών συστημάτων φτάσαμε στην κατασκευή αντίστοιχων συστημάτων που φτάνουν στο μέγεθος, το πολύ, μιας παλάμης. Παράλληλα με αυτά υπήρξε και μια αρκετά μεγάλη ζήτηση για αποθήκευση αλλά και διαχείριση τεραστίων ποσοτήτων πληροφορίας. Το internet, αλλά και η δυνατότητα δημιουργίας δικτύων ικανών να διακινούν το τεράστιο μέγεθος των πληροφοριών αυτών, έθεσαν τις βάσεις στην εξέλιξη επιπλέον συστημάτων μικρότερου μεγέθους, τα επονομαζόμενα *ενσωματωμένα πληροφορικά συστήματα* (*embedded systems*). Τα συστήματα αυτά δημιουργήθηκαν (ή εξελίχθηκαν) για την εξυπηρέτηση και τη διευκόλυνση του κοινού μιας και η ζήτηση που υπήρξε ήταν πολύ μεγάλη. Παραδείγματα τέτοιων μικρών συστημάτων είναι : ένας μίνι προσωπικός ηλεκτρονικός υπολογιστής (*Notebook*), ένα *σύστημα πλοήγησης* (*GPS, Global Positioning System)*, ένα *"έξυπνο" κινητό τηλέφωνο* (*smart phone*), ένας *ψηφιακός "βοηθός",* το γνωστό μας *PDA* (*Personal Digital Assistant*) κτλ κτλ. Επιπλέον παραδείγματα τέτοιων μικρών συστημάτων (πέραν των προαναφερθέντων) μπορούν να βρίσκονται : σε ένα αυτοκίνητο, σε ένα ψυγείο, σε μια αντλία καυσίμου, σε ένα μετεωρολογικό σταθμό ως αισθητήρες (*sensors*), σε ένα σπίτι ως σύστημα ασφάλειας (συναγερμός, κάμερες,

πυρανίχνευση κτλ), και σε πολλές άλλες περιπτώσεις και χρήσεις. Είναι κοινή πεποίθηση όλων λοιπόν ότι διανύουμε την *Επανάσταση της Πληροφορίας και των Πληροφορικών Συστημάτων*. Μέσα σε όλη αυτή την εξέλιξη ο τομέας των ενσωματωμένων πληροφορικών συστημάτων δεν μπορεί να μη μένει ανεπηρέαστος. Η παγκόσμια βιομηχανία των κατασκευαστών υπολογιστών αλλά και των εταιριών λογισμικού έχουν ρίξει το βάρος τους στην ανάπτυξη των συστημάτων αυτών. Δεν είναι τυχαίο ότι η αγορά των ενσωματωμένων συστημάτων εκτινάχτηκε το 2002 στο τεράστιο αριθμό των 1.122.000.000.000 υπολογιστών έναντι των 131.000.000 επιτραπέζιων υπολογιστών, όπως αναφέρουν στο βιβλίο τους οι Patterson και Hennessy [10]. Αυτός ο αριθμός πιστεύεται ότι είναι πλέον αστρονομικός τη χρονιά που διανύουμε. Παράλληλα μια ακόμη σημαντική εξέλιξη αλλά και ανάπτυξη έχει παρατηρηθεί στο τομέα κατασκευής λογισμικών προγραμμάτων και λειτουργικών συστημάτων για ενσωματωμένα συστήματα. Όλο και περισσότερες εταιρίες λογισμικού και αυτόνομοι προγραμματιστές προσπαθούν να δημιουργούν *προγράμματα – εφαρμογές* (*applications*) για ενσωματωμένες συσκευές. Ένα πολύ πρόσφατο παράδειγμα στις μέρες μας είναι οι εφαρμογές που έχουν δημιουργηθεί για τις φημισμένες συσκευές της Apple το iPad και iPhone.

Ένας πολύ μεγάλος αλλά και γρήγορα αναπτυσσόμενος κλάδος της βιομηχανίας των ενσωματωμένων συστημάτων είναι η τεχνολογία των συσκευών *FPGA* (*Field Programmable Gate Array*), σε ελεύθερη μετάφραση θα μπορούσαμε να πούμε ότι είναι οι *Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών*. Η παρούσα μεταπτυχιακή διατριβή ασχολείται με τη χρήση της τεχνολογίας των συσκευών FPGA, στην επεξεργασία δεδομένων. Συγκεκριμένα μελετήθηκε και αναπτύχθηκε ένας μη παραδοσιακός τρόπος επεξεργασίας δεδομένων με χρήση συσκευών FPGA. Η προσπάθεια εστιάσθηκε στην επεξεργασία αποθηκευμένων δεδομένων που μπορεί να υπάρχουν σε μα *Βάση Δεδομένων ΒΔ* (*Database*), με χρήση ερωτημάτων (*queries*) που μεταφράσθηκαν από *δομημένη γλώσσα επερωτήσεων SQL* (*Structured Query Language*) σε *γλώσσα περιγραφής υλικού VHDL* (*VHSIC Hardware Description Language*). Η μεθοδολογία που αναπτύχθηκε μπορεί να ακολουθηθεί και να αναπτυχθεί από κάποιον, ο οποίος θα ήθελε να κάνει χρήση των συσκευών FPGA στην επεξεργασία των δεδομένων μιας ΒΔ, (από εδώ και στο εξής θα αναφέρεται η Βάση Δεδομένων με την ένδειξη ΒΔ, επιπλέον η ίδια φιλοσοφία ακολουθείται και με άλλα ακρώνυμα μέσα στη διατριβή). Τα αποτελέσματα ήταν άκρος εντυπωσιακά και οι *χρόνοι εκτέλεσης* (*latency*) των πλειάδων, γρήγοροι. Παράλληλα μέσα από τις προσομοιώσεις αποκαλύφθηκε και η ικανότητα των συσκευών FPGA στην παράλληλη επεξεργασία δεδομένων. Δηλαδή την επεξεργασία δεδομένων σε παράλληλη διάταξη, των λογικών κυκλωμάτων που υλοποιήθηκαν μέσα στη συσκευή. Η απόδοση και σε αυτή την

περίπτωση παρέμεινε στα ίδια υψηλά επίπεδα. Αυτή η ικανότητα είναι εξαιρετικά σημαντική μιας και τα σύγχρονα θέματα επεξεργασίας δεδομένων αναζητούν νέες τεχνικές και σχεδιάσεις. Ειδικότερα όταν η επεξεργαστική ισχύς των συστημάτων επεξεργασίας δεδομένων είναι τεράστια, η συσκευές FPGA με τη χαμηλή κατανάλωση ισχύος και με την κατάλληλη τεχνογνωσία, μπορούν να αποφορτίσουν αυτό το μέγεθος.

Έχει γίνει μια πρώτη μελέτη σε αντίστοιχες αναφορές (*papers*) όπως *«Streams on Wires - A Query Compiler for FPGAs»* [20] και *«Data Processing on FPGAs»* [21], οι οποίες μας έδειξαν ενθαρρυντικά αποτελέσματα για τη χρήση των συσκευών FPGA στην επεξεργασία *ροών δεδομένων* (*data stream*) χρηματοοικονομικού περιεχομένου - ροή και πορεία των μετοχών σε μια χρηματιστηριακή συναλλαγή. Στις αναφορές αυτές αναπτύσσεται ο τρόπος και η διαδικασία που πρέπει να γίνει, για να αποτελέσει η συσκευή FPGA αξιόπιστος συνεργάτης στην επεξεργασία δεδομένων. Υπάρχει εκτενής αναφορά στην δυνατότητα των συσκευών FPGA στον παραλληλισμό και την παράλληλη επεξεργασία. Τρόποι βελτιστοποίησης της ταχύτητας επεξεργασίας αλλά και τις σχεδίασης. Στην παρούσα μεταπτυχιακή διατριβή επεκτάθηκαν νέες τεχνικές αλλά και νέα σχεδίαση στην επεξεργασία δεδομένων. Πιστοποιήθηκαν εν μέρη κάποια θέματα αλλά αναπτύχθηκαν και άλλα νέα. Η έρευνα αυτή αποκάλυψε “θησαυρό” ικανοτήτων των συσκευών FPGA στην επεξεργασία δεδομένων.

Για την υλοποίηση της προσπάθειας αυτής έγινε χρήση της συσκευής FPGA *Spartan-3E* της εταιρίας *Xilinx* καθώς και του αντίστοιχου λογισμικού *σχεδίασης μέσω υπολογιστή* (*CAD, Computer Aided Design*) το *ISE*. Τα εργαλεία (λογισμικά) σχεδίασης CAD χρησιμοποιούνται για τη σχεδίαση και το προγραμματισμό των συσκευών FPGA. Αυτές οι συσκευές έχουν ενσωματωμένους επεξεργαστές οι οποίοι σχεδιάζονται (προγραμματίζονται για την ακρίβεια) με χρήση *πυρήνων επεξεργαστών* (*processor cores*). Η σχεδίαση του όποιου κυκλώματος γίνεται με διάφορους τρόπους (περισσότερη λεπτομέρεια στα επόμενα κεφάλαια), ο ένας από αυτούς είναι με χρήση *γλωσσάς περιγραφής υλικού* (*HDL, Hardware Description Language*) (Ενότητα 2.1.1).

1.1 Σκοπός Μεταπτυχιακής Διατριβής

Σε αυτήν τη μεταπτυχιακή διατριβή επιχειρήθηκε η ανάπτυξη ενός υβριδικού συστήματος το οποίο περιέχει μια συσκευή FPGA (*Field Programmable Gate Array*) και ένα ηλεκτρονικό υπολογιστή *Η/Υ*. Στον Η/Υ έχει εγκατασταθεί ένα *σύστημα βάσεων δεδομένων* (*ΣΒΔ*) από όπου αντλούνται τα δεδομένα προς επεξεργασία. Η επεξεργασία των δεδομένων γίνεται με την

εκτέλεση κάποιων σύνθετων ερωτημάτων SQL (*queries*) επάνω στη συσκευή FPGA, τα ερωτήματα αυτά υλοποιούνται με χρήση της γλώσσας περιγραφής υλικού VHDL (*VHSIC Hardware Description Language*).

Σκοπός της διατριβής είναι να βρεθεί ο τρόπους με τον οποίο θα μπορεί μια συσκευή FPGA να χρησιμοποιηθεί, έτσι ώστε να παρέχει επιπλέον επεξεργασία δεδομένων σε ένα ΣΒΔ.. Οι *αναφορές* (*papers*) [14,17,20,21,22,23] που μελετήθηκαν αναφέρουν ότι τα FPGA χρησιμοποιούνται για να *επιταχύνουν* (*accelerate*) κάποιες λειτουργίες στη ΒΔ. Κάποια πράγματα, που αφορούν κυρίως την επεξεργασία δεδομένων, μπορούν λόγω της εγγενής παραλληλίας του υλικού τα κάνουν πιο γρήγορα από ότι σε μια *κεντρική μονάδα επεξεργασίας* (*CPU, Central Processing Unit*).

Σε αυτή τη μεταπτυχιακή διατριβή επιδιώκεται να παρουσιαστούν χρήσιμα συμπεράσματα από τα οποία θα μπορέσει κανείς να αναδείξει τη χρησιμότητα των συσκευών FPGA στην επεξεργασία δεδομένων. Τόσο ως υλικό υποστήριξης ενός ΣΒΔ όσο και ως υλικό επεξεργασίας δεδομένων άλλων Πληροφορικών Συστημάτων. Επίσης θα επιχειρηθεί να δοθεί μια καθαρή εικόνα για το πώς λειτουργούν αυτές οι συσκευές και ποιο σκοπό επιτελεί η παρουσία τους.

1.2 Διάρθρωση Μεταπτυχιακής Διατριβής

Η μεταπτυχιακή διατριβή αποτελείται από δυο μέρη, το θεωρητικό και το πειραματικό. Στο θεωρητικό μέρος θα παρουσιαστεί η γενική αρχιτεκτονική δομή των συσκευών FPGA (Ενότητα 2.2) καθώς και η γλώσσα περιγραφής υλικού VHDL(Ενότητα 2.1). Θα γίνει μια πρώτη θεωρητική προσέγγιση των ερωτημάτων που θα αναλυθούν σύμφωνα με την μορφή τους τόσο στη Σχεσιακή Άλγεβρα (Ενότητα 3.1) όσο και στην γλώσσα επερωτήσεων SQL(Ενότητα 3.2). Επίσης θα αναδειχθεί ο τρόπος επέκτασης της γλώσσα επερωτήσεων SQL (καθώς και των αντίστοιχων ερωτημάτων) σε μορφή VHDL (Ενότητα 3.3), όπως και ο τρόπος εισαγωγής των στοιχείων (δεδομένων) επεξεργασίας και διαχείρισης (Ενότητα 4.1). Στο πειραματικό μέρος επιχειρήθηκε η υλοποίηση και εκτέλεση των ερωτημάτων αυτών τόσο στην προσομοίωση όσο και στην πραγματική εκτέλεση τους στο FPGA (Κεφάλαιο 5). Στο σημείο αυτό να αναφερθεί ότι η εκτέλεση σε πραγματικές συνθήκες δεν ολοκληρώθηκε με επιτυχία μιας και ο χρόνος εκτίμησης των πειραμάτων ήταν άστοχος. Ο χρόνος που εκτιμήθηκε υστερεί κατά πολύ με αυτόν που πραγματικά χρειάζεται. Ενδεικτικά να αναφερθεί ότι δεν υπολογίσθηκε σωστά η απαιτητική και πολύπλοκή σχεδίαση του κυκλώματος, αυτή που πρέπει να υλοποιηθεί, τέτοια ώστε να φέρνει

σε πέρας μερικά από τα αποτελέσματα που βρέθηκαν στην προσομοίωση. Για περισσότερες λεπτομέρειες θα παραπεμφθείτε στην Ενότητα 5.3.

Αναλυτικά έχουμε, στο Κεφάλαιο 2 θα παρουσιαστεί η γλώσσα περιγραφής υλικού VHDL και η γενική αρχιτεκτονική δομή των συσκευών FPGA. Στο Κεφάλαιο 3 θα παρουσιαστεί η γλώσσα επερωτήσεων SQL και επέκταση τους στην αντίστοιχη VHDL. Στο Κεφάλαιο 4 θα παρουσιαστεί ο τρόπος εισαγωγής των στοιχείων προς επεξεργασία, αλλά και τα δεδομένα που θα χρησιμοποιηθούν εσωτερικά του κυκλώματος. Επιπλέον θα αναλυθεί η *διευθυνσιοδοτούμενη μνήμη περιεχομένου* (*CAM, Content Addressable Memory*) καθώς ο τρόπος υπολογισμού της, μιας και αυτή χρειάζεται για να υλοποιηθούν κάποια από τα ζητούμενα. Επίσης θα παρουσιαστεί ο τρόπος διασύνδεσης της συσκευής FPGA με τον Η/Υ με χρήση του πρωτοκόλλου επικοινωνίας *UART* (*Universal Asynchronous Peceiver Transmitter*), όπως ο τρόπος εισαγωγής και εξαγωγής δεδομένων κατά την προσομοίωση. Στο Κεφάλαιο 5 θα παρουσιαστεί ο τρόπος υλοποίησης και εκτέλεσης των ερωτημάτων τόσο στην προσομοίωση όσος και στην πραγματική λειτουργία του FPGA. Επίσης θα γίνει μια πρώτη αποτίμηση (εκτίμηση) των αποτελεσμάτων που προέκυψαν. Τέλος στο Κεφάλαιο 6 θα αναπτυχθούν χρήσιμες παρατηρήσεις και συμπεράσματα για βελτίωση των αποτελεσμάτων, όπως και την καλύτερη μελλοντική επέκταση τους.

1.3 Προαπαιτούμενα

Για να μπορέσει κάποιος να παρακολουθήσει την παρούσα διατριβή θα πρέπει να έχει κάνει τα βασικά μαθήματα της κατεύθυνσης του *υλικού* (*hardware*) του προγράμματος σπουδών ενός τμήματος Πληροφορικής, όπως *Ψηφιακή Σχεδίαση* (*Digital Design*) και η *Αρχιτεκτονική Υπολογιστών* (*Computer Architecture*), αλλά και του μαθήματος των *Βάσεων Δεδομένων* (*Databases*).

Να είναι εξοικειωμένος με έννοιες όπως :

- Λογικές Πύλες (*Logic Gate*)
- Αποκωδικοποιητής (*Decoder*)
- Κωδικοποιητής (*Coder*)
- Πολυπλέκτης (*MUX, Multiplexer*)

- Μνήμες *RAM, ROM, SRAM, DRAM*
- Το βασικό Κύτταρο Μνήμης, *Flip Flop*
- Δίαυλος Δεδομένων (*Data Bus*), Δίαυλος Διευθύνσεως (*Address Bus*)
- Σχεσιακή Άλγεβρα (*Relation Algebra*)
- Πλειάδα (Tuple)
- Σχήμα Σχέσης (Relation Schema), δηλώνεται με $R(A_1, A_2, ..., A_n)$ και αποτελείται από ένα όνομα σχέσης R και μια λίστα από γνωρίσματα $A_1, A_2, ..., A_n$
- Γλώσσα Επερωτήσεων SQL, ερωτήματα (*SQL queries*)
- Απλή Συνένωση, Θήτα, Ισότητας και Φυσική (*Join, Theta Join, Equijoin, Natural Join*)

Επίσης θα πρέπει να έχει γνώση περί διασύνδεσης, τουλάχιστον τα στοιχειώδη, *λογικών κυκλωμάτων* (*Logic Circuit*) και *ολοκληρωμένων κυκλωμάτων ΟΚ* (*IC, Integrate Circuit*).

Σημείωση : Τα ΟΚ στη βιβλιογραφία αναφέρονται και ως *τσιπ* (*Chip*).

Όλα τα παραπάνω αποτελούν βασική γνώση για την παρακολούθηση του θεωρητικού μέρους της διατριβής. Για την παρακολούθηση του πειραματικού μέρους δεν χρειάζεται κάποιος να έχει γνώση της γλωσσάς VHDL. Ο αναγνώστης θα μπορούσε να παρακάμψει το κεφάλαιο 2 και να μεταβεί απευθείας στο κεφάλαιο 5. Όλες οι δοκιμές γίνονται αποκλειστικά μέσα από το πρόγραμμα ISE, όπου για κάθε υλοποίηση παράγεται μια κυματομορφή (γραφική απεικόνιση) η οποία παρουσιάζεται και αναλύεται. Σκοπός των πειραμάτων είναι να φανεί πως ανταποκρίνεται η σχεδίαση των κυκλωμάτων στα δεδομένα προς επεξεργασία, όπως και αν η προσέγγιση αυτή θεωρείται τελικά ιδανική για χρήση. Αν κάποιος όμως έχει καλή γνώση της γλώσσας VHDL θα έχει μια καλύτερη εικόνα ως προς το τι διαδραματίζεται μέσα στη συσκευή. Σε αυτό το σημείο θα αναφερθεί και πάλι ότι δεν είναι προϋπόθεση ότι κάποιος πρέπει να ξέρει τη VHDL για να παρακολουθήσει το δεύτερο μέρος. Όλες οι αναφορές που δημιουργούνται παρουσιάζονται σε πίνακες και διαγράμματα, έτσι ώστε να είναι καλύτερη παρακολούθησή τους.

Κεφάλαιο 2

VHDL - FPGA

Σε αυτό το κεφάλαιο ο αναγνώστης θα έχει την ευκαιρία να γνωρίσει την *γλώσσα περιγραφής υλικού VHDL* (*VHSIC Hardware Description Language*) αλλά και την γενική αρχιτεκτονική δομή των *Επιτόπου Προγραμματιζόμενων Διατάξεων Πυλών* (*FPGA, Field Programmable Gate Array*). Και οι δύο ενότητες προσπαθούν να δώσουν την απαραίτητη γνώση που χρειάζεται για την παρακολούθηση της μεταπτυχιακής διατριβής.

Συγκεκριμένα στην Ενότητα 2.1.1 παρουσιάζεται η γλώσσα περιγραφής υλικού HDL (Hardware Description Language) και στην Ενότητα 2.1.2 η γλώσσα περιγραφής υλικού VHDL (VHSIC Hardware Description Language) με τα παραδείγματα της. Στην Ενότητα 2.2 παρουσιάζεται η γενική αρχιτεκτονική δομή των Επιτόπου Προγραμματιζόμενων Διατάξεων Πυλών (FPGA, Field Programmable Gate Array).

2.1 Γλώσσας Περιγραφής Υλικού VHDL, (VHSIC Hardware Description Language)

Σε αύτη την ενότητα θα παρουσιαστεί η γλώσσα περιγραφής υλικού VHDL, συγκεκριμένα πρώτα θα γίνει μια μικρή εισαγωγή για τις γλώσσες περιγραφής υλικού HDL και κατόπιν θα περιγραφούν κάποια κύρια χαρακτηριστικά που διέπουν τη γλώσσα περιγραφής υλικού VHDL. Να σημειωθεί ότι η συγγραφή των δύο ενοτήτων αποτελούν υλικό δύο βιβλίων, η Ενότητα 2.1 του βιβλίου *«Ψηφιακή σχεδίαση (3 Έκδοση)»* του Morris Mano [16] και η Ενότητα 2.2 του βιβλίου *«Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL»* των Stephen Brown και Zvonko Vranesic [26].

Σκοπός και των δύο ενοτήτων είναι να δει ο αναγνώστης κάποια σημαντικά χαρακτηριστικά της γλώσσας VHDL, αυτά που την κάνουν να είναι διαφορετική από μια γλώσσα προγραμματισμού.

2.1.1 Γλώσσας Περιγραφής Υλικού (HDL, Hardware Description Language)

Η γλώσσα περιγραφής υλικού HDL (Hardware Description Language) είναι μια γλώσσα που περιγράφει τα κυκλώματα των ψηφιακών συστημάτων σε μορφή κειμένου. Θυμίζει τις γλώσσες προγραμματισμού, αλλά είναι ειδικά προσανατολισμένη στην περιγραφή των δομών υλικού (hardware) και τη συμπεριφορά τους. Μπορεί να χρησιμοποιηθεί για την περιγραφή λογικών διαγραμμάτων, λογικών εκφράσεων και άλλων πιο περίπλοκων ψηφιακών κυκλωμάτων. Ως γλώσσα *τεκμηρίωσης*, η HDL χρησιμοποιείται για να παραστήσει και να περιγράψει ψηφιακά συστήματα σε μια μορφή που μπορεί να διαβαστεί τόσο από ανθρώπους όσο και από υπολογιστές και είναι κατάλληλη ως γλώσσα ανταλλαγής πληροφορίας μεταξύ σχεδιαστών. Το περιεχόμενο των προγραμμάτων της γλώσσας αυτής μπορεί να αποθηκευτεί και να ανακτηθεί εύκολα, καθώς και να υποστεί επεξεργασία από λογισμικό υπολογιστών με αποτελεσματικό τρόπο. Υπάρχουν δύο κύριες εφαρμογές της επεξεργασίας HDL: η προσομοίωση και η σύνθεση.

Η προσομοίωση λογισμικών κυκλωμάτων είναι παράσταση της δομής και της συμπεριφοράς των ψηφιακών λογικών συστημάτων με τη χρήση υπολογιστή. Ένας προσομοιωτής μεταφράζει την περιγραφή HDL και παράγει αναγνώσιμη έξοδο, όπως ένα διάγραμμα χρονισμού. Η έξοδος αυτή αποτελεί πρόβλεψη της συμπεριφοράς του κυκλώματος, πριν αυτό κατασκευαστεί στην πράξη. Η προσομοίωση επιτρέπει την ανίχνευση λειτουργικών σφαλμάτων σε ένα κύκλωμα υπό σχεδίαση, χωρίς να χρειάζεται να κατασκευαστεί το φυσικό κύκλωμα. Τα λάθη που εντοπίζονται κατά την προσομοίωση μπορούν να διορθωθούν με μετατροπή των κατάλληλων εντολών HDL. Οι ακολουθίες εισόδου, που ελέγχουν τη λειτουργικότητα της σχεδίασης ονομάζονται *δοκιμαστική είσοδος*. Έτσι, για να προσομοιωθεί ένα ψηφιακό σύστημα, περιγράφεται πρώτα τη σχεδίασή του με χρήση HDL και μετά επαληθεύεται η ορθότητα της σχεδίασης

προσομοιώνοντας και ελέγχοντας τη λειτουργία του με μια δοκιμαστική είσοδο, επίσης γραμμένη σε HDL.

Η *σύνθεση λογικών κυκλωμάτων* είναι η διαδικασία εύρεσης ενός συνόλου ηλεκτρονικών εξαρτημάτων και των διασυνδέσεων τους που ονομάζονται *κατάλογος συνδέσεων δικτύου (netlist)* από το μοντέλο ενός ψηφιακού συστήματος που περιγράφεται σε HDL. Ο κατάλογος συνδέσεων δικτύου στο επίπεδο των πυλών μπορεί να χρησιμοποιηθεί για να κατασκευαστεί ένα ολοκληρωμένο κύκλωμα ή να σχεδιαστεί φυσικά ένα τυπωμένο κύκλωμα. Η σύνθεση λογικών κυκλωμάτων με τον τρόπο αυτό, έχει ομοιότητες με το μεταγλωττισμό ενός προγράμματος γραμμένου σε μια συμβατική γλώσσα υψηλού επιπέδου. Η διαφορά είναι ότι στη σύνθεση λογικών κυκλωμάτων, αντί να παραχθεί κώδικας μηχανής, παράγεται μια βάση δεδομένων με οδηγίες για το πώς θα κατασκευάσει κανείς ένα φυσικό ψηφιακό κύκλωμα, το οποίο υλοποιεί τις εντολές που περιγράφονται από τον κώδικα HDL. Η λογική σύνθεση βασίζεται σε ακριβείς, τυπικούς αλγορίθμους που υλοποιούν ψηφιακά κυκλώματα, και είναι εκείνο το μέρος της σχεδίασης λογικών κυκλωμάτων, το οποίο μπορεί να αυτοματοποιηθεί με χρήση λογισμικού.

Υπάρχουν πολλές εκδόσεις της HDL στη βιομηχανία, αναπτυγμένες από εταιρείες οι οποίες σχεδιάζουν ολοκληρωμένα κυκλώματα ή βοηθούν στο σχεδιασμό αυτών, στο Πίνακα Α.1 στο Παράρτημα Α φαίνονται κάποιες από αυτές. Δύο πρότυπες εκδόσεις της HDL, η VHDL και η Verilog HDL, υποστηρίζονται από το IEEE (Institute of Electrical and Electronic Engineers-Ινστιτούτο Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών). Η VHDL είναι γλώσσα που δημιουργήθηκε κατόπιν εντολής του Αμερικάνικου Υπουργείου Αμύνης. (Το V στο VHDL είναι το πρώτο γράμμα της συντόμευσης VHSIC, που προκύπτει: από Very High Speed Integrated Circuits – Ολοκληρωμένα κυκλώματα πολύ υψηλής ταχύτητας). Η Verilog ξεκίνησε αρχικά ως μια ιδιόκτητη HDL που προωθήθηκε από την εταιρεία με Cadence Data Systems αλλά η Cadence μετέφερε τον έλεγχο της Verilog σε μια ομάδα εταιρειών και πανεπιστημίων γνωστή ως Open Verilog International (OVI).

2.1.2 Γλώσσας Περιγραφής Υλικού VHDL, (VHSIC Hardware Description Language)

Σε πολλές περιπτώσεις η γλώσσα VHDL χρησιμοποιεί ασυνήθιστη σύνταξη για την περιγραφή των λογικών κυκλωμάτων. Ο κύριος λόγος γι' αυτό είναι ο τι η γλώσσα VHDL αποσκοπούσε

αρχικά στην καταγραφή και προσομοίωση κυκλωμάτων και όχι στην περιγραφή κυκλωμάτων με σκοπό τη σύνθεση.

Αντικείμενα Δεδομένων

Οι πληροφορίες παριστάνονται στη γλώσσα VHDL ως *αντικείμενα δεδομένων (data objects).* Υπάρχουν τρία είδη αντικειμένων δεδομένων: *σήματα, σταθερές και μεταβλητές*. Για την περιγραφή λογικών κυκλωμάτων τα πιο σημαντικά αντικείμενα δεδομένων είναι τα σήματα. Αυτά αντιπροσωπεύουν τα λογικά σήματα (δηλαδή τα καλώδια) του κυκλώματος. Οι σταθερές και οι μεταβλητές είναι επίσης συχνά χρήσιμες για την περιγραφή λογικών κυκλωμάτων, αλλά δεν χρησιμοποιούνται συχνά.

Ονόματα Αντικειμένων Δεδομένων

Οι κανόνες για να οριστούν τα ονόματα των αντικειμένων δεδομένων είναι απλοί. Μπορεί να χρησιμοποιηθεί στο όνομα οποιοσδήποτε αλφαριθμητικός χαρακτήρας, καθώς και χαρακτήρας "_". Υπάρχουν τέσσερις περιορισμοί. Ένα όνομα δεν είναι δυνατό να είναι κάποια λέξη-κλειδί της γλώσσας VHDL, πρέπει να ξεκινά από ένα γράμμα, δεν είναι δυνατό να τελειώνει με το χαρακτήρα "_" και δεν είναι δυνατό να έχει δύο διαδοχικούς χαρακτήρες "_". Επομένως, παραδείγματα έγκυρων ονομάτων είναι τα *x, x1, x_y* και *Byte*. Μερικά παραδείγματα εσφαλμένων ονομάτων είναι τα *1x, _y, x__y* και *entity.* Το τελευταίο όνομα απαγορεύεται επειδή αποτελεί λέξη-κλειδί της γλώσσας VHDL. Πρέπει επίσης να σημειώσουμε ότι η γλώσσα VHDL δεν διακρίνει ανάμεσα στα κεφαλαία και τα πεζά γράμματα, με αποτέλεσμα τα ονόματα *X* και *x* να είναι γι' αυτή το ίδιο, και η λέξη ENTITY ίδια λέξη με το *entity*.

Τιμές Αντικειμένων Δεδομένων και Αριθμοί

Χρησιμοποιούνται αντικείμενα δεδομένων τύπου *σήματος* (*SIGNAL*) για να παριστάνουν τα επιμέρους λογικά σήματα σ' ένα κύκλωμα, πολλά λογικά σήματα και δυαδικούς αριθμούς (ακέραιους). Η τιμή ενός δεδομένου τύπου σήματος καθορίζεται με τη βοήθεια αποστρόφων, όπως '0' ή '1'. Η τιμή ενός δεδομένου τύπου σήματος με μήκος πολλών bits δίνεται με διπλά εισαγωγικά. Ένα παράδειγμα σήματος μήκους τεσσάρων bits είναι "1001" και ένα δεδομένο τύπου σήματος με μήκος οκτώ bits είναι "10011000". Τα διπλά εισαγωγικά μπορούν να χρησιμοποιηθούν για να δηλώσουν ένα δυαδικό αριθμό. Επομένως, ενώ η τιμή "1001" μπορεί να αντιπροσωπεύει τις τέσσερις τιμές '1', '0', '0' και '1' τύπου σήματος, μπορεί επίσης να δηλώνει τον

ακέραιο αριθμό $(1001)_2 = (9)_{10}$. Οι ακέραιοι αριθμοί μπορούν να αναγραφούν με δεκαδική μορφή και χωρίς τη χρήση εισαγωγικών, όπως στους αριθμούς 9 ή 152. Οι τιμές των αντικειμένων δεδομένων τύπου *σταθεράς* (*CONSTANT*) και *μεταβλητής* (*VARIABLE*) εμφανίζονται με τον ίδιο τρόπο όπως τα δεδομένα τύπου σήματος.

Αντικείμενα Δεδομένων Τύπου Σήματος (SIGNAL)

Τα αντικείμενα δεδομένων τύπου *σήματος* (*SIGNAL*) αντιπροσωπεύουν λογικά σήματα στα καλώδια ενός κυκλώματος. Υπάρχουν τρεις θέσεις, στις οποίες τα σήματα μπορούν να δηλώνονται σε ένα πρόγραμμα VHDL: σε μία *δήλωση οντότητας (entity declaration)* στο τμήμα δηλώσεων της αρχιτεκτονικής δομής και στο τμήμα δηλώσεων ενός πακέτου. Ένα σήμα πρέπει να δηλώνεται μέσω του τύπου (*type*) που έχει, με τον ακόλουθο τρόπο:

$$SIGNAL\ όνομα\_σήματος : τύπος\_σήματος \qquad (2.1)$$

Ο *τύπος_σήματος* του σήματος προσδιορίζει τις έγκυρες τιμές που μπορεί να λάβει το σήμα και τους έγκυρους τρόπους χρήσης του στα προγράμματα της γλώσσας VHDL. Στην παρούσα ενότητα περιγράφονται δέκα τύποι σήματος, τους ακόλουθους: *BIT,BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, SIGNED, UNSIGNED, INTEGER, ENUMERATION και BOOLEAN.*

Τελεστές

Η γλώσσα VHDL διαθέτει λογικούς τελεστές, αριθμητικούς τελεστές και τελεστές σχέσεων. Αυτοί διακρίνονται σε κατηγορίες με έναν ασυνήθιστο τρόπο, όπως φαίνεται στον Πίνακα 2.1, σύμφωνα με την προτεραιότητα τους. Σημειώνεται ότι οι τελεστές της ίδιας κατηγορίας δεν έχουν προτεραιότητα ο ένας ως προς τον άλλο. Δεν υπάρχει προτεραιότητα ανάμεσα στους λογικούς τελεστές. Επομένως μια λογική έκφρασης της μορφής:

$$x1\ AND\ x2\ AND\ x3\ OR\ x4 \qquad (2.2)$$

δεν έχει τη σημασία της έκφρασης $\chi 1 \bullet \chi 2 \bullet \chi 3 + \chi 4$, όπως θα αναμενόταν, επειδή η πράξη AND δεν έχει προτεραιότητα ως προς την πράξη OR. Στην πραγματικότητα, η έκφραση αυτή δεν είναι καθόλου έγκυρη στη γλώσσα VHDL με τον τρόπο που αναγράφεται. Για να είναι έγκυρη και να έχει την επιθυμητή σημασία, θα πρέπει να γραφεί ως εξής:

$$(x1\ AND\ x2\ AND\ x3)\ OR\ x4 \qquad (2.3)$$

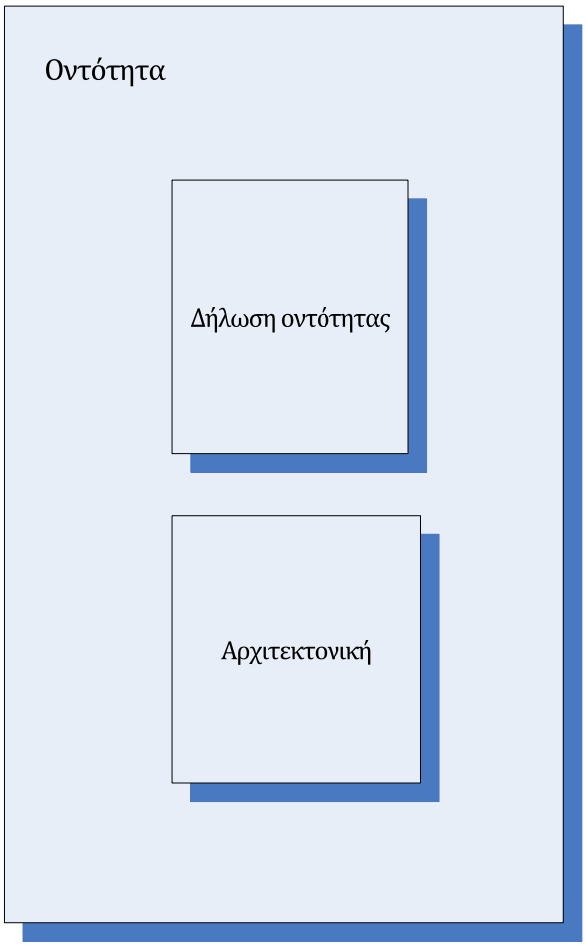
	Κατηγορία τελεστή	**Τελεστής**
Υψηλή προτεραιότητα	Διάφοροι	**, ABS, NOT
	Πολλαπλασιασμού	*, /, MOD, REM
	Προσήμου	+, -
	Πρόσθεσης	+, -, &
	Σχέσεων	=, /=, <, <=, >, >=
Χαμηλή προτεραιότητα	Λογικοί	AND, OR, NAND, NOR, XOR, XNOR

Πίνακας 2.1: Οι τελεστές της γλώσσας VHDL

Όσον αφορά τους τελεστές σχέσεων, ο τελεστής "/=" σημαίνει *όχι ίσος,* ο τελεστής "<=" σημαίνει *μικρότερος ή ίσος* και ο τελεστής ">=" σημαίνει *μεγαλύτερος ή ίσος.*

Οντότητες Σχεδίων στη Γλώσσα VHDL

Ένα κύκλωμα ή υποκύκλωμα που περιγράφεται με ένα πρόγραμμα της γλώσσας VHDL ονομάζεται *οντότητα σχεδίου (design entity),* ή απλά *οντότητα.* Στην Εικόνα 2.1 εικονίζεται η γενική δομή μίας οντότητας. Αυτή αποτελείται από δύο κύρια μέρη: τη *δήλωση οντότητας (entity declaration),* η οποία καθορίζει τα σήματα εισόδου και εξόδου, καθώς και την *αρχιτεκτονική (architecture),* η οποία δίδει τις λεπτομέρειες του κυκλώματος.



Εικόνα 2.1: Γενική δομή μίας οντότητας σχεδίου της γλώσσας VHDL

Δήλωση Οντότητας (ENTITY)

Τα σήματα εισόδου και εξόδου μίας οντότητας καθορίζονται με τη βοήθεια της δήλωσης οντότητας, που γίνεται με τη λέξη-κλειδί *ENTITY*, με τον τρόπο που παρουσιάζεται στην Εικόνα 2.2. Το όνομα της οντότητας μπορεί να είναι οποιοδήποτε έγκυρο όνομα της γλώσσας VHDL. Οι αγκύλες δηλώνουν ένα προαιρετικό τμήμα. Τα σήματα εισόδου και εξόδου καθορίζονται με τη βοήθεια της λέξης-κλειδί *PORT* (*θύρα*). Εάν μία θύρα είναι είσοδος, έξοδος ή διπλής κατεύθυνσης καθορίζεται από την *κατάσταση (mode)* της θύρας. Οι δυνατές καταστάσεις παρουσιάζονται στον Πίνακα 2.2. Εάν η κατάσταση μίας θύρας δεν καθορίζεται, θεωρείται ότι αυτή είναι *θύρα εισόδου* (IN).

```
ENTITY entity_name IS
        PORT ( [SIGNAL] signal_name {, signal_name} : [mode] type-name{;
               [SIGNAL] signal_name {, signal_name} : [mode] type-name}
             );
END entity-name ;
```

Εικόνα 2.2: Γενική μορφή της δήλωσης μίας οντότητας

Κατάσταση	Σκοπός
IN	Χρησιμοποιείται για ένα σήμα που αποτελεί είσοδο σε μια οντότητα
OUT	Χρησιμοποιείται για ένα σήμα που αποτελεί έξοδο από μία οντότητα. Η τιμή του σήματος δεν μπορεί να χρησιμοποιηθεί μέσα στην οντότητα. Αυτό σημαίνει ότι σε μία εντολή αντιστοίχησης το σήμα μπορεί να εμφανιστεί στο αριστερό τμήμα μόνο του τελεστή "<=".
INOUT	Χρησιμοποιείται για ένα τμήμα που αποτελεί είσοδο αλλά και έξοδο μίας οντότητας
BUFFER	Χρησιμοποιείται για ένα σήμα που αποτελεί έξοδο από μία οντότητα. Η τιμή του σήματος μπορεί να χρησιμοποιηθεί μέσα στην οντότητα, που σημαίνει ότι σε μία εντολή αντιστοίχησης το σήμα μπορεί να εμφανιστεί στο αριστερό αλλά και το δεξιό τμήμα του τελεστή "<=".

Πίνακας 2.2: Δυνατές καταστάσεις σημάτων που είναι θύρες οντοτήτων

Αρχιτεκτονική (ARCHITECTURE)

Η *Αρχιτεκτονική* (ARCHITECTURE) μίας οντότητας παρέχει τις λεπτομέρειες του κυκλώματος της οντότητας. Η γενική δομή μίας αρχιτεκτονικής παρουσιάζεται στην Εικόνα 2.3. Αυτή αποτελείται από δύο κύρια μέρη: την *περιοχή δήλωσης (declarative region)* και το *σώμα της αρχιτεκτονικής (architecture body).* Η περιοχή δήλωσης εμφανίζεται πριν από τη λέξη-κλειδί *BEGIN*. Αυτή μπορεί να χρησιμοποιηθεί για να δηλωθούν σήματα, τύποι ορισμένοι από το χρήστη και σταθερές. Μπορεί επίσης να χρησιμοποιηθεί για να δηλωθούν *συνιστώσες* (*COMPONENTS*) και να καθοριστούν *χαρακτηριστικά* (*ATTRIBUTES*).

```
ARCHITECTURE architecture_name OF entity_name IS
        [SIGNAL declarations]
        [CONSTANT declarations]
        [TYPE declarations]
        [COMPONENT declarations]
        [ATTRIBUTE specifications]
BEGIN
      {[COMPONENT instantiation statement;}
      {[CONCURRENT ASSIGNMENT statement;}
      {[PROCESS statement;}
      {[GENERATE statement;}
END [architecture_name];
```

Εικόνα 2.3: Γενική μορφή μίας αρχιτεκτονικής

Η λειτουργία μίας οντότητας καθορίζεται στο σώμα της αρχιτεκτονικής, το οποίο έπεται της λέξης BEGIN. Το τμήμα αυτό περιλαμβάνει εντολές που καθορίζουν τις λογικές συναρτήσεις που εκτελεί το κύκλωμα, οι οποίες μπορούν να εκφραστούν με διάφορους τρόπους. Θα περιγράψουμε έναν αριθμό δυνατοτήτων στις ενότητες που ακολουθούν.

Παράδειγμα 2.1

Στην Εικόνα 2.4 παρουσιάζεται το πρόγραμμα γλώσσας VHDL για μία οντότητα που ονομάζεται *Fulladd,* η οποία αντιπροσωπεύει έναν πλήρη αθροιστή. Η δήλωση της οντότητας ορίζει τα σήματα εισόδου και εξόδου. Η θύρα εισόδου *Cin* αποτελεί το κρατούμενο εισόδου και τα Bits που πρέπει να προστεθούν είναι οι θύρες *x* και *y.* Οι θύρες εξόδου είναι το άθροισμα *s* και το κρατούμενο εξόδου *Cout.*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY fulladd IS
      PORT ( Cin, x, y   : IN      STD_LOGIC ;
                s, Cout   : OUT  STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE  LogicFunc OF fullad IS
BEGIN
   s <= x XOR y XOR Cin ;
   Cout <= (x AND y) OR (x AND Cin) OR (y AND Cin) ;
END LogicFunc ;
```

Εικόνα 2.4: Πρόγραμμα για έναν πλήρη αθροιστή

Τα σήματα εισόδου και εξόδου ονομάζονται *θύρες* (*ports*) της οντότητας. Ο όρος αυτός έχει υιοθετηθεί από το λεξιλόγιο των ηλεκτρολόγων και αναφέρεται σε μία σύνδεση εισόδου η εξόδου σε ένα ηλεκτρικό κύκλωμα.

Η αρχιτεκτονική καθορίζει τη λειτουργία του πλήρη αθροιστή με τη βοήθεια λογικών εξισώσεων. Η ονομασία της αρχιτεκτονικής μπορεί να είναι οποιοδήποτε έγκυρο όνομα της γλώσσας VHDL. Σε αυτό το απλό παράδειγμα έχουμε επιλέξει το όνομα *LogicFunc*. Αναφορικά με τη γενική μορφή της αρχιτεκτονικής της Εικόνας 2.3, μία λογική εξίσωση αποτελεί έναν τύπο σύμφωνης εντολής αντιστοίχησης.

Χρήση Υποκυκλωμάτων

Μία οντότητα της γλώσσας VHDL που ορίζεται σε ένα αρχείο προγράμματος μπορεί να χρησιμοποιηθεί ως *υποκύκλωμα* (*subcircuit*) σε ένα άλλο αρχείο προγράμματος. Στο λεξιλόγιο της γλώσσας VHDL ένα υποκύκλωμα ονομάζεται *συνιστώσα* (*component*). Ένα υποκύκλωμα πρέπει να δηλώνεται με τη βοήθεια μίας *δήλωσης συνιστώσας* (*component declaration*). Η εντολή αυτή καθορίζει το όνομα του υποκυκλώματος και δίνει ονόματα στις θύρες εισόδου και εξόδου του. Η δήλωση συνιστώσας μπορεί να υπάρχει είτε στην περιοχή δηλώσεων της αρχιτεκτονικής είτε σε μία δήλωση πακέτου. Η γενική μορφή της εντολής παρουσιάζεται στην Εικόνα 2.5. Η σύνταξη που ακολουθείται είναι παρόμοια με αυτήν που χρησιμοποιείται για τη δήλωση οντοτήτων.

```
COMPONENT component_name
          [GENERIC (parameter_name : integer := default_value { ;
                    parameter_name : integer := default_value } ) ;]
          PORT ( [SIGNAL] signal_name {, signal_name} : [mode] type_name {;
                 [SIGNAL] signal_name {, signal_name} :[mode] type_name });
END COMPONENT ;
```

Εικόνα 2.5: Γενική μορφή της δήλωσης συνιστώσας

Εφόσον γίνει η δήλωση μίας συνιστώσας, μπορεί να δημιουργηθεί το στιγμιότυπο της συνιστώσας ως υποκύκλωμα του σχεδίου. Αυτό γίνεται σε μία εντολή *δημιουργίας στιγμιοτύπου συνιστώσας (component instantiation*), η οποία έχει τη γενική μορφή που φαίνεται στην Εικόνα 2.6.

```
Όνομα στιγμιοτύπου : όνομα συνιστώσας PORT MAP (
    επίσημο_όνομα => πρακτικό_όνομα  {επίσημο_όνομα => πρακτικό_όνομα } ) ;
```

Εικόνα 2.6: Γενική μορφή δημιουργίας στιγμιοτύπου συνιστώσας (component instantiation)

Κάθε *επίσημο_όνομα* αποτελεί το όνομα μίας θύρας του υποκυκλώματος. Κάθε *πραγματικό_όνομα* αποτελεί το όνομα ενός σήματος του προγράμματος που δημιουργεί στιγμιότυπο του υποκυκλώματος. Η έκφραση *"επίσημο_όνομα =>"* αναγράφεται, έτσι ώστε η ακολουθία των σημάτων που αναφέρονται μετά από τις λέξεις–κλειδιά *PORT MAP* να μην είναι ίδια με την ακολουθία των θυρών της αντίστοιχης δήλωσης συνιστώσας. Στη γλώσσα VHDL αυτό ονομάζεται *συσχέτιση μέσω ονομάτων (named association)*. Εάν τα ονόματα των σημάτων που ακολουθούν τις λέξεις-κλειδιά PORT MAP αναγράφονται με την ίδια σειρά όπως στη δήλωση συνιστώσας, τότε δεν απαιτείται η έκφραση *"επίσημο_όνομα => "*. Αυτή η μέθοδος ονομάζεται *συσχέτιση θέσης (positional association).*

Ένα παράδειγμα χρήσης της μίας συνιστώσας (δηλαδή ενός υποκυκλώματος) παρουσιάζεται στην Εικόνα 2.7. Εκεί δίνεται το πρόγραμμα ενός αθροιστή διάδοσης κρατουμένου μήκους τεσσάρων bits, ο οποίος δομείται χρησιμοποιώντας τέσσερα στιγμιότυπα του υποκυκλώματος *fulladd.* Οι είσοδοι του αθροιστή είναι το κρατούμενο εισόδου *Cin* και οι αριθμοί μήκους τεσσάρων bits *X* και *Y* ενώ έξοδοι είναι το άθροισμα μήκους τεσσάρων bits *S* και το κρατούμενο εξόδου *Cout.* Έχει επιλεγεί η ονομασία *Structure* για την αρχιτεκτονική, επειδή το ιεραρχικό ύφος ενός προγράμματος που χρησιμοποιεί υποκυκλώματα ονομάζεται συχνά δομημένο ύφος (*structural style*). Παρατηρείται ότι ένα σήμα *C* μήκους τριών *bits* δηλώνεται ότι αντιπροσωπεύει τα κρατούμενα εξόδου των σταδίων 0, 1 και 2. Το σήμα αυτό δηλώνεται στην αρχιτεκτονική και όχι στη δήλωση οντότητας, επειδή χρησιμοποιείται εσωτερικά και δεν αποτελεί θύρα εισόδου ή εξόδου.

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

ENTITY adder IS
   PORT (   Cin   : IN  STD_LOGIC ;
            X, Y  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            S     : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) ;
            Cout  : OUT STD_LOGIC) ;
END adder ;

ARCHITECTURE Structure OF adder IS
SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;
COMPONENT fulladd
    PORT (  Cin, x, y : IN      STD_LOGIC ;
             s, Cout  : OUT STD_LOGIC);
END COMPONENT ;
BEGIN
   stage0: fulladd PORT MAP ( Cin , X(0),  Y(0),  S(0),  C(1) ) ;
   stage1: fulladd PORT MAP ( C(1), X(1),  Y(1),  S(1),  C(2) ) ;
   stage2: fulladd PORT MAP ( C(2), X(2),  Y(2),  S(2),  C(3) ) ;
   stage3: fulladd PORT MAP (
   x => X(3) , y =>  Y(3),  Cin => C(3), s =>  S(3), Cout =>  Cout ) ;
END Structure ;
```

Εικόνα 2.7: Πρόγραμμα για έναν αθροιστή μήκους τεσσάρων bits με τη βοήθεια στιγμιοτύπων των συνιστωσών

Η επόμενη εντολή της αρχιτεκτονικής δίνει τη δήλωση συνιστωσών του υποκυκλώματος *fulladd.* Το σώμα της αρχιτεκτονικής δημιουργεί τέσσερα στιγμιότυπα του υποκυκλώματος του πλήρη αθροιστή. Στις τρεις πρώτες εντολές δημιουργίας στιγμιοτύπων χρησιμοποιείται συσχέτιση θέσης, επειδή τα σήματα καταγράφονται με την ίδια σειρά, με την οποία παρουσιάζονται στη δήλωση της συνιστώσας *fulladd* στην Εικόνα 2.4. Η τελευταία εντολή δημιουργίας στιγμιοτύπου αποτελεί ένα παράδειγμα συσχέτισης μέσω ονομάτων. Σημειώνεται ότι είναι έγκυρο να χρησιμοποιείται το ίδιο όνομα για ένα σήμα στην αρχιτεκτονική με αυτό που χρησιμοποιείται για το όνομα μιας θύρας σε μία συνιστώσα. Ένα παράδειγμα γι' αυτό το σημείο είναι το σήμα *Cout*. Τα ονόματα σημάτων που χρησιμοποιείται στις εντολές δημιουργίας στιγμιοτύπων καθορίζουν με σαφήνεια πως διασυνδέονται μεταξύ τους τα στιγμιότυπα για να δημιουργήσουν την οντότητα του αθροιστή.

Ένα δεύτερο παράδειγμα δημιουργίας στιγμιοτύπου μίας συνιστώσας παρουσιάζεται στην Εικόνα 2.8. Στο πρόγραμμα περιλαμβάνεται ένα πακέτο που ονομάζεται *lpm_components,* το οποίο ανήκει στη βιβλιοθήκη *lpm.* Αυτό το πακέτο αντιπροσωπεύει μία συλλογή συνιστωσών που ονομάζονται *Βιβλιοθήκη Παραμετροποιημένων Κυκλωμάτων (Library of Parameterized Modules, LPM),* η οποία είναι μία προτυποποιημένη βιβλιοθήκη που περιέχει δομικές μονάδες κυκλωμάτων, οι οποίες είναι εν γένει χρήσιμες για την υλοποίηση λογικών κυκλωμάτων. Η συγκεκριμένη βιβλιοθήκη περιλαμβάνεται στο λογισμικό σχεδίασης *MAX+plus II* της εταιρίας Altera [04], όπου περιλαμβάνει τις συνιστώσες της βιβλιοθήκης LPM ως πρότυπες δομικές μονάδες για τη δημιουργία λογικών κυκλωμάτων.

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

ENTITY adderLPM IS
       PORT ( Cin  : IN   STD_LOGIC ;
              X,Y  : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
                S  : OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
             Cout  : OUT  STD_LOGIC) ;
END adderLPM ;

ARCHITECTURE Structure OF adderLPM IS
BEGIN
   instance: lpm_add_sub
   GENERIC MAP (LPM_WIDTH => 4)
       PORT MAP(
                  dataa  => X,
                  datab  => Y,
                  Cin    => Cin,
                  result => S,
                  Cout   => Cout
                );
END Structure ;
```

Εικόνα 2.8: Δημιουργία στιγμιοτύπου ενός αθροιστή μήκους τεσσάρων bits με βάση τη βιβλιοθήκη LPM

Το πρόγραμμα της Εικόνας 2.8 δημιουργεί στιγμιότυπο της συνιστώσας της βιβλιοθήκης LPM που ονομάζεται *lpm_add_sub*. Αυτή αντιπροσωπεύει ένα κύκλωμα αθροιστή / αφαιρέτη. Η λέξη-κλειδί GENERIC χρησιμοποιείται για να ορίσει τον αριθμό των bits του αθροιστή / αφαιρέτη σε 4. Η λειτουργία της κάθε θύρας (PORT) της συνιστώσας *lpm_add_sub* είναι προφανής με βάση τα ονόματα των θυρών που αναγράφονται στην εντολή δημιουργίας στιγμιοτύπου.

Για περαιτέρω εμβάθυνση στη γλώσσα VHDL παρακαλείται ο αναγνώστης να ανατρέξει στη βιβλιογραφία [11,26] ή όποιο άλλο βιβλίο για τη γλώσσα VHDL βρει διαθέσιμο.

2.2 Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών FPGA, (Field Programmable Gate Array)

Ο αναγνώστης πριν προχωρήσει στην ανάγνωση της αρχιτεκτονικής των *Επιτόπου Προγραμματιζόμενων Διατάξεων Πυλών* (*FPGA, Field Programmable Gate Array*), καλό θα ήταν να ανατρέξει στο Παράρτημα Β να δει λίγο την ιστορική αναδρομή των συσκευών αυτών, πως ξεκίνησαν και πως ελίχθηκαν. Πάρα ταύτα μπορεί να συνεχίσει απευθείας στο κείμενο.

2.2.1 Εισαγωγή

Στην αγορά υπάρχει μια πολύ μεγάλη ποικιλία ΟΚ τα οποία εκτελούν διάφορες λειτουργίες από τις πιο απλές μέχρι τις πιο πολύπλοκες. Έτσι υπάρχουν ΟΚ πολύ απλής λειτουργικότητας έως εξαιρετικά περίπλοκης. Από αυτά μπορεί κανείς να συμπεράνει ότι κάποια από τα ΟΚ λόγω της ιδιαιτερότητας τους και λόγω διαφορετικών απαιτήσεων, χρειάζεται να σχεδιαστούν από την αρχή μιας και επιτελούν συγκεκριμένες λειτουργίες. Σε αυτά τα προβλήματα δίνουν λύση τρεις μορφές ΟΚ, τα τυπικά ΟΚ (Παράρτημα Β.2), οι *διατάξεις προγραμματιζόμενης λογικής* (*PLDs, Programmable Logic Devices* (Παράρτημα Β.3) και τα *ειδικά ολοκληρωμένα κυκλώματα εφαρμογής* (*ASICs, Application-Oriented Integrated Circuit*, Παράρτημα Β.4). Η συσκευές FPGA βρίσκονται στη δεύτερη κατηγορία, με έντονα χαρακτηριστικά όμως που εμφανίζονται στην τρίτη κατηγόρια.

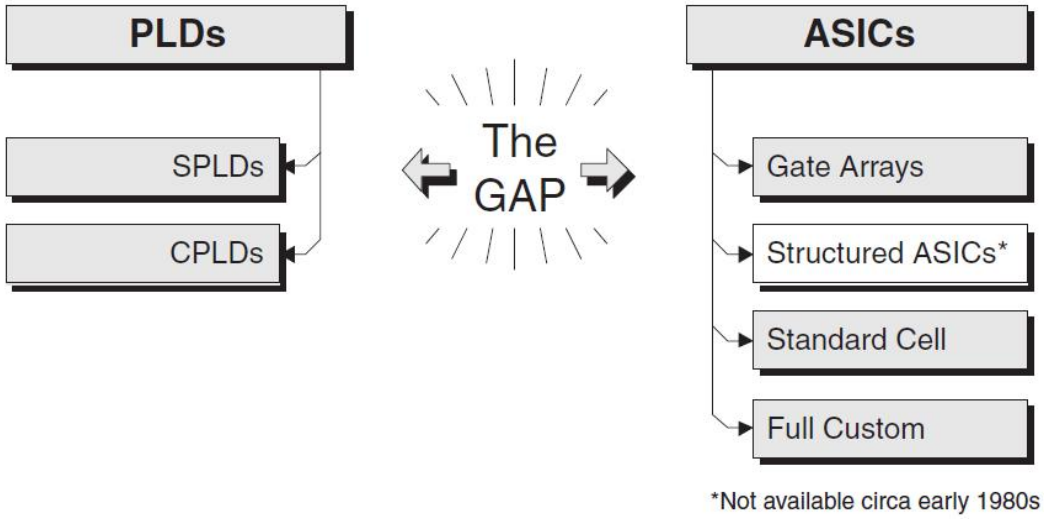
Τα FPGAs, όπως αναφέρθηκε είναι *διατάξεις προγραμματιζόμενης λογικής* (*PLDs*), υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων και επιτρέπουν πλήρη ελευθερία όσον αφορά τη διαδικασία σχεδίασης. Είναι συσκευές γενικής χρήσης, οι οποίες εμπεριέχουν ένα μεγάλο αριθμό *λογικών στοιχείων* (*LE, Logic Element,* Ενότητα 2.2.7), καλωδίων διασύνδεσης και διακοπτών. Οι

συσκευές FPGA σχεδιάστηκαν για να προσφέρουν ευελιξία στο σχεδιαστή ΟΚ μιας και η αρχιτεκτονική τους επιτρέπει (με πολύ μικρό κόστος), την επανεξέταση και επαναδιαμόρφωση του κυκλώματος που πρέπει να δημιουργηθεί. Παράλληλα, η ταχύτητα και η απόδοση των FPGA δεν υστερεί σε τίποτα από αυτή που είναι στα *ειδικά ολοκληρωμένα κυκλώματα εφαρμογής* (*ASICs*). Με άλλα λόγια τα FPGA επιτρέπουν τη δημιουργία σύγχρονων ΟΚ με πολύ μικρό κόστος ελέγχου αλλά και χρόνο κατασκευής. Ενδεικτικά να αναφερθεί ότι ο χρόνος ολοκλήρωσης και διανομής ενός ΟΚ ASICs, από τη σχεδίαση μέχρι την παραγωγή, κυμαίνεται από 6 έως 8 μήνες. Αυτός ο χρόνος είναι μηδενικός στα FPGA μιας και το κύκλωμα που θα υλοποιηθεί μπορεί να αντιγραφεί (όσες φορές χρειάζεται) και να χρησιμοποιηθεί από άλλες συσκευές FPGA.

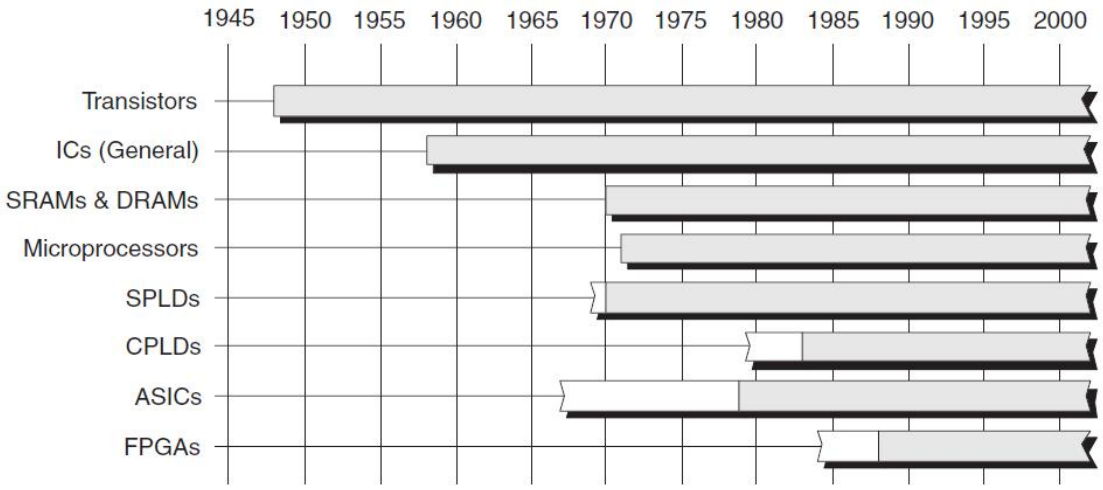
Επίσης ένα ακόμη σημαντικό πλεονέκτημα των συσκευών FPGAs έναντι των ASICs είναι η ανεκτικότητα στις μελλοντικές αλλαγές. Αν υπάρξει κάποια αλλαγή στο κύκλωμα (προστεθεί ή αφαιρεθεί λειτουργία) μιας συσκευής ASICs, θα πρέπει το κύκλωμα να επανεξεταστεί, να ξανασχεδιαστεί, να ελεγχθεί, να κατασκευαστεί κτλ κτλ. Δηλαδή θα χρειαστεί (πάλι) το λιγότερο από 6 έως 8 μήνες. Αυτός ο χρόνος είναι ελάχιστος για τις συσκευές FPGA μιας και το μόνο που θα χρειαστεί είναι στον ήδη υπάρχον πρόγραμμα VHDL (αυτό που μεταφράζεται σε κύκλωμα), να προστεθεί ο επιπλέον κώδικας που θα υλοποιήσει την πρόσθετη λειτουργία. Η επαναδιαμόρφωση (προγραμματισμός) και ο έλεγχος της συσκευής έχει μηδενικό κόστος.

2.2.2 Αρχιτεκτονική Δομή

Η πρώτη εμφάνιση των διατάξεων FPGAs έγινε αρχές της δεκαετίας του ’80 και συγκεκριμένα το 1984 όταν η εταιρία Xilinx θέλησε να μετριάσει το χάσμα που επικρατούσε μεταξύ των διατάξεων PLDs και ASICs (Εικόνα 2.9). Αποτέλεσμα αυτής της προσπάθειας ήταν η πρώτη διάταξη FPGA, η οποία άργησε αρκετά να απορροφηθεί από την αγορά μιας και οι μηχανικοί δεν τα είχαν ακόμη σε εκτίμηση. Όλα αυτά υφίστανται μέχρι τις αρχές του 1990 όπου και ξεκίνησε σιγά σιγά η απορρόφηση τους. Στη σημερινή αγορά υπάρχει μια πληθώρα από διαφορετικές ποικιλίες συσκευών, όπου ολοένα αναπτύσσονται και εξελίσσονται με γρήγορους ρυθμούς. Παράλληλα φαίνεται ένα ιδιαίτερο ενδιαφέρον από ακαδημαϊκής πλευράς στη μελέτη αυτής της τεχνολογίας μιας και αποτελεί τη σύγχρονη μορφή των ψηφιακών συστημάτων και της ψηφιακής σχεδίασης. Στην Εικόνα 2.10 φαίνεται η εξέλιξη των ΟΚ, από τις πρώτες μορφές των τρανζίστορ μέχρι τις τελευταίες αυτές των FPGA.



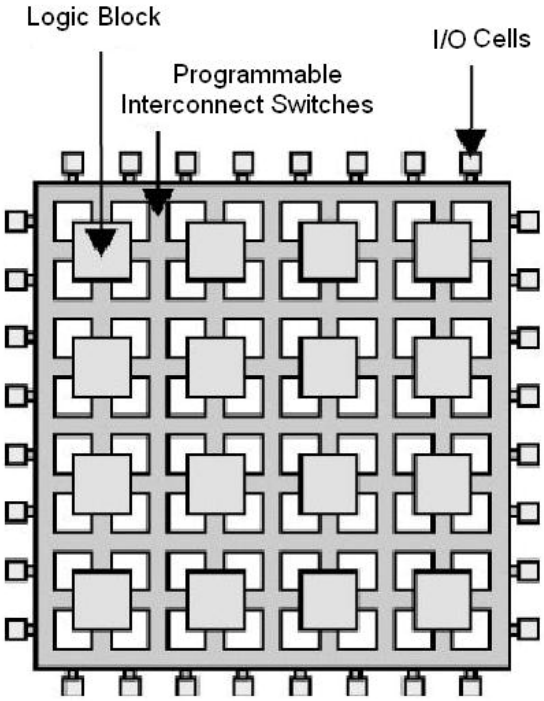
Εικόνα 2.9: [09] Το χάσμα μεταξύ διατάξεων PLDs και ASICs



Εικόνα 2.10: [09] Χρονική εξέλιξη της τεχνολογίας των ολοκληρωμένων κυκλωμάτων, (οι ημερομηνίες είναι κατά προσέγγιση)

Η γενική αρχιτεκτονική δομή των FPGAs είναι αυτή της Εικόνας 2.11, όπου μπορεί κανείς να παρατηρήσει ότι αποτελείται από τρεις χαρακτηριστικές ομάδες. Η πρώτη ομάδα αποτελείται από τους *ακροδέκτες εισόδου εξόδου* (*pins, I/O Cells*), τα οποία είναι και η διασύνδεση της συσκευής με τον έξω κόσμο, αποτελούν δηλαδή την είσοδο και την έξοδο του FPGA. Η δεύτερη ομάδα αποτελείται από τις *λογικές βαθμίδες* (*LB*, *Logic Block*) όπου το καθένα από αυτά περιέχει ένα σύνολο από *λογικά στοιχεία* (*LE, Logic Element*), ο αριθμός αυτός διαφέρει αν αρχιτεκτονική σχεδίαση. Και τέλος η τρίτη ομάδα με τους *προγραμματιζόμενους διακόπτες διασύνδεσης* (*PIS*, *Programmable Interconnect Switches*) ή αλλιώς *κανάλια δρομολόγησης* (*Routing Channels*). Η

τελευταία ομάδα είναι αυτή που διασύνδεει τις λογικές βαθμίδες (LB) μεταξύ τους όπως και με τους ακροδέκτες (I/O). Η ποσότητα αυτών των χαρακτηριστικών (I/O, LE, LB, PIS κτλ) διαφέρει από αρχιτεκτονική σε αρχιτεκτονική, όμως η γενική δομή παραμένει η ίδια.



Εικόνα 2.11: Γενική αρχιτεκτονική δομή FPGA

Επιπλέον ενσωματωμένα χαρακτηριστικά τα οποία δε φαίνονται στην Εικόνα 2.11 είναι τα παρακάτω:

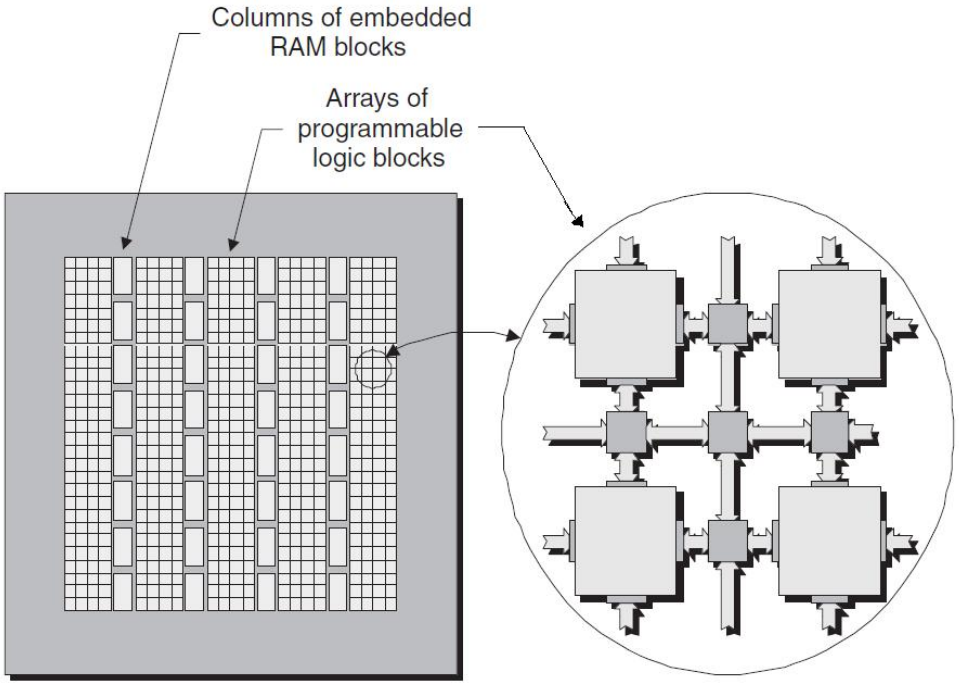
- Μνήμες (*Memories*) RAM
- Αριθμητικές μονάδες (*Arithmetic Units*)
- Επεξεργαστές (*Microprocessors*)
- Διευθυντές ρολογιού (*CLKm, Clock Manager*)
- Πίνακας Αναζήτησης (*LUT, Lookup Table*)
- Λογικό στοιχείο (*LE, Logic Element*)
- Γρήγορες αλυσίδες κρατουμένου (*Fast Carry Chains*)

Όλα τα χαρακτηριστικά θα αναλυθούν εκτενέστερα παρακάτω, εκτός αυτά των ακροδεκτών και των διασυνδέσεων μιας και αυτά διαφέρουν κατά πολύ ανά αρχιτεκτονική.

2.2.3 Μνήμες RAM (Memories)

Μνήμες (*Memories*) ή *βαθμίδες μνημών* (*MB, Memory Block*), μπορεί να αποτελούν μέρος της αρχιτεκτονικής δομής, ανάμεσα σε λογικές βαθμίδες (LB) ή περιφερειακά της συσκευής FPGA (Εικόνα 2.12). Αυτές οι βαθμίδες μνημών (MB) εξυπηρετούν τοπικές ανάγκες του κυκλώματος παρέχοντας επιπλέον μνήμη στις λογικές βαθμίδες. Είναι συνήθως μνήμες RAM και λόγω της ανάγκης για μεγάλη διάθεση μνήμης, πολλές από τις συσκευές FPGA ενσωματώνουν βαθμίδες από RAM (*RAM Block*). Κάθε βαθμίδα RAM μπορεί να χρησιμοποιηθεί ανεξάρτητα ή να συνδυαστούν πολλές μαζί ώστε να υλοποιήσουν μία μεγαλύτερη μνήμη. Επίσης οι μνήμες αυτές μπορούν να χρησιμοποιηθούν σε ποικιλία εφαρμογών όπως

- Single Port Ram
- Dual Port Ram
- FIFO, κτλ

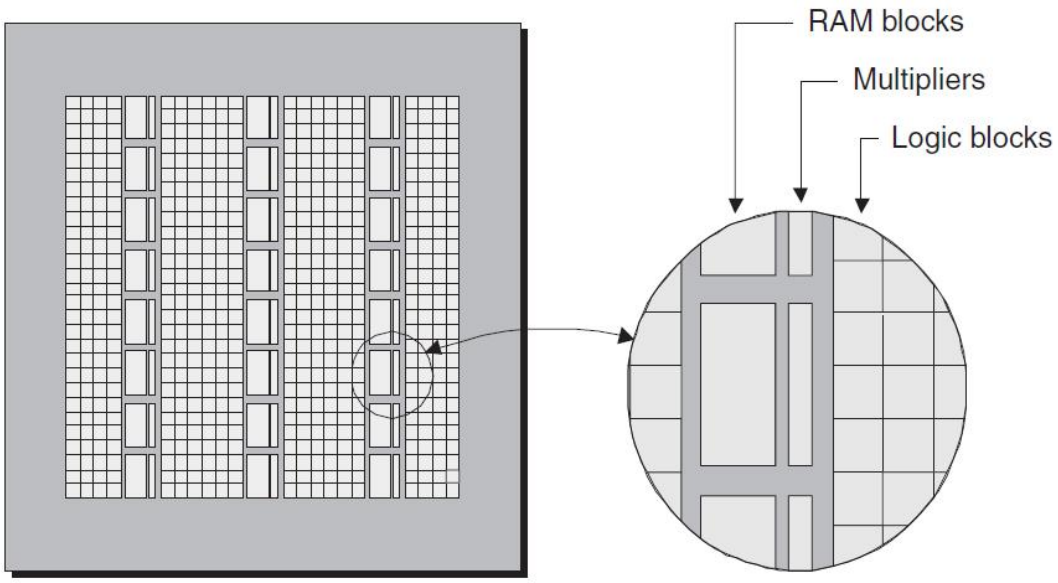


Εικόνα 2.12: [09] Ενσωματωμένες μνήμες RAM σε FPGA

2.2.4 Αριθμητικές Μονάδες (Arithmetic Units)

Αριθμητικές μονάδες (*Arithmetic Units*), χρήσιμες για ένα υπολογιστικό σύστημα το οποίο καυχιέται ότι εκτελούνται γρήγορα όλες οι αριθμητικές πράξης. Μια τέτοια μονάδα λοιπόν βρίσκεται και στα FPGA. Ο λόγος της ύπαρξης τους είναι ότι κάποιες αριθμητικές συναρτήσεις, όπως ο πολλαπλασιασμός, είναι αργές όταν υλοποιηθούν από ένα αριθμό από λογικά στοιχειά. Για αυτό ενσωματώνονται αριθμητικές μονάδες στην αρχιτεκτονική δομή των FPGAs, έτσι ώστε να επιταχύνουν τη διαδικασία (Εικόνα 2.13). Πολλές διατάξεις FPGA ενσωματώνουν τέτοιες αριθμητικές μονάδες όπως είναι :

- Πολλαπλασιαστές (*Multipliers*)
- Αθροιστές (*Adders*)
- Πολλαπλασιαστές - Συσσωρευτές (*MAC, Multiple-Accumulator*)

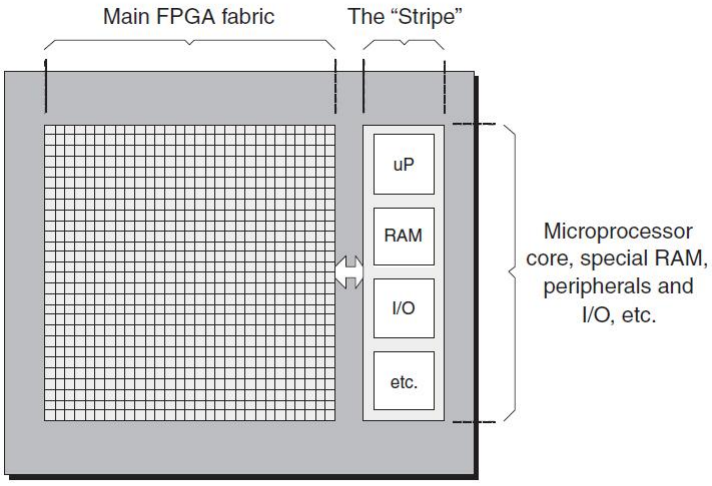


Εικόνα 2.13: [09] Ενσωματωμένες αριθμητικές μονάδες σε FPGA

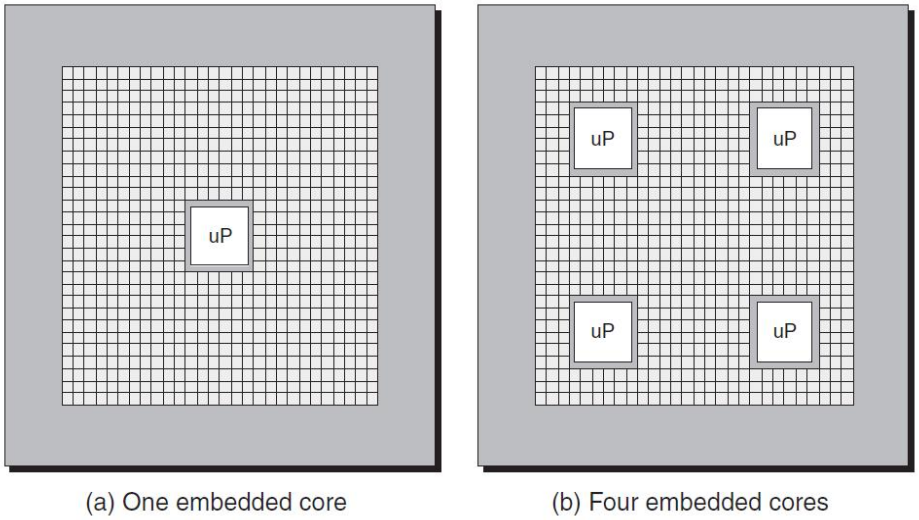
Οι ενσωματωμένες αριθμητικές μονάδες μαζί με τις ενσωματωμένες μνήμες κάνουν ιδανική τη χρήση των FPGAs σε εφαρμογές *ψηφιακής επεξεργασίας σήματος* (*DSP, Digital Signal Processing*). Άλλωστε δεν είναι τυχαίο ότι ένα μεγάλο μερίδιο της αγοράς των FPGAs, αποτελεί προϊόν για οπτικοακουστικούς καταναλωτές.

2.2.5 Επεξεργαστές (Microprocessors)

Επεξεργαστές (*Microprocessors*), οι προηγμένες διατάξεις FPGAs ενσωματώνουν *πυρήνες επεξεργαστών* (*Microprocessors Cores*), για χρήση σε πολύ απαιτητικές λειτουργίες όπου χρειάζεται ένας γρήγορος επεξεργαστής. Υπάρχουν δύο τύποι ενσωματωμένων επεξεργαστών, οι *σκληρού πυρήνα* (*Hard Core*) και οι *μαλακού πύρινα* (*Soft Core*). Οι επεξεργαστές μαλακού πύρινα υλοποιούνται μέσω των λογικών στοιχείων της διάταξης και μπορούν να τροποποιηθούν και να προσαρμοστούν στις ανάγκες του χρήστη. Οι σκληρού πυρήνα ενσωματώνονται στη διάταξη σαν ένα επιπλέον ΟΚ και δεν είναι δυνατόν να τροποποιηθεί από το χρήστη (Εικόνα 2.14). Ο αριθμός των επεξεργαστών που μπορούν να ενσωματωθούν εξαρτάται από την αρχιτεκτονική, στην Εικόνα 2.15 φαίνονται κάποια από αυτά.



Εικόνα 2.14: [09] Σκληρός πυρήνας (Hard Core) ενός FPGA έξω από την κύρια αρχιτεκτονική δομή του



Εικόνα 2.15: [09] Ενσωματωμένοι επεξεργαστές σε διαφορετικές αρχιτεκτονικές δομές

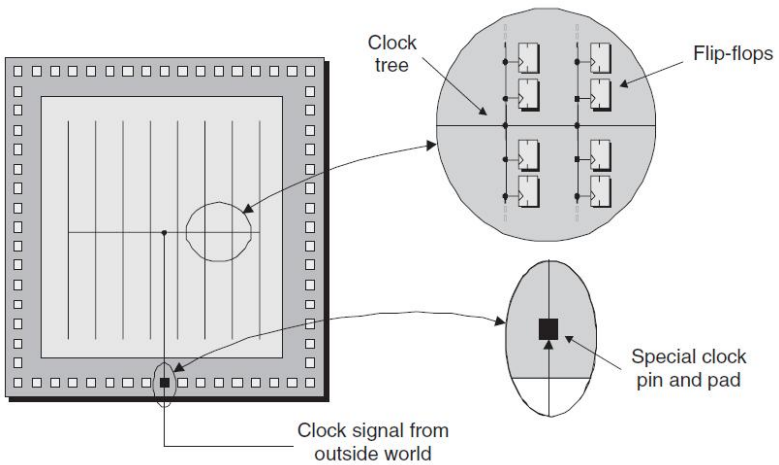
2.2.6 Διευθυντές Ρολογιού (CLKm, Clock Managers)

Διευθυντές ρολογιού (*CLKm, Clock Managers*), ένα από τα σημαντικότερα χαρακτηριστικά σε ένα FPGA. Οι διευθυντές ρολογιού (CLKm) παρέχουν σε όλη την αρχιτεκτονική δομή του FPGA το σήμα του ρολογιού ή σήματα διαφορετικών ρολογιών.

Σκοπός της παρουσία τους είναι:

- Αποφυγή του φαινομένου *Clock Skew*, το οποίο είναι η μη σωστή και ταυτόχρονη άφιξη του σήματος του ρολογιού σε όλα τα λογικά στοιχειά της διάταξης.
- Αφαίρεση *παραμόρφωσης χρονισμού* (*jitter removal*).
- *Σύνθεση συχνότητας* (*frequency synthesis*), υπάρχουν ποικιλίες συχνοτήτων που μπορούν να διαμορφωθούν με έναν διευθυντή ρολογιού (CLKm).
- *Ολίσθηση φάσης* (*phase shifting*).
- *Διόρθωση αυτό - απόκλισης* (*auto - skew correction*).

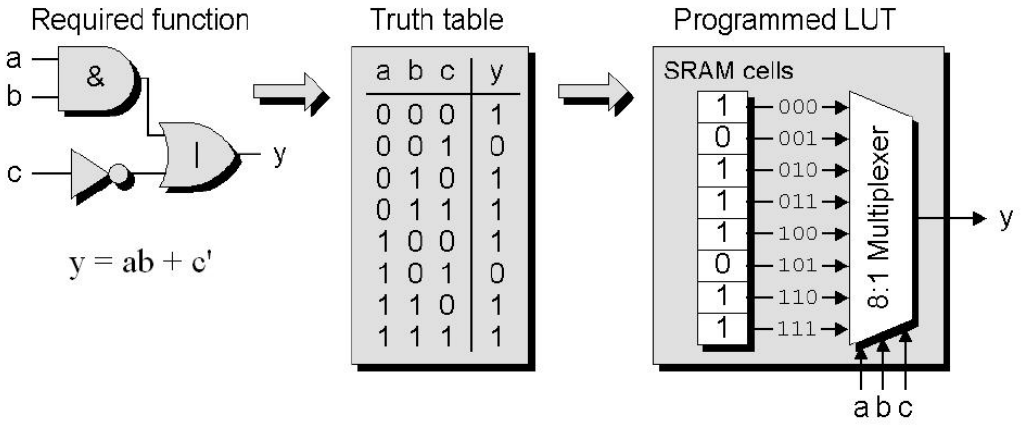
Ο τρόπος αποφυγής μερικών από τα παραπάνω προβλήματα επιτυγχάνεται με την τοποθέτηση *δέντρων ρολογιού* (*clock trees*), πολλαπλών *ακροδεκτών* (*clock pins*) και πολλαπλών *πεδίων ρολογιού* (*clock domains*), Εικόνα 2.16. Σε μερικές περιπτώσεις, υπάρχουν περισσότερα από ένα ή δυο πεδία ρολογιού όπου το καθένα από αυτά έχει το δικό του δέντρο και τους δικούς του ακροδέκτες.



Εικόνα 2.16: [09] Αναπαράσταση ενός δέντρου ρολογιού

2.2.7 Πίνακας Αναζήτησης (*LUT, Lookup Table*)

Η αρχιτεκτονική δομή των FPGAs διαφέρει σημαντικά από αυτή των διατάξεων SPLD και CPLD (Εικόνα Β.1, Παράρτημα Β.3) επειδή δεν περιέχουν πύλες AND και OR, αλλά περιέχει *λογικές μονάδες* για την υλοποίηση των ζητούμενων συναρτήσεων. Η πιο ευρέως χρησιμοποιημένη λογική μονάδα στις συσκευές FPGAs είναι ο *πίνακας αναζήτησης* (*LUT, Lookup Table*), Εικόνα 2.17. Ο πίνακας αναζήτησης LUT περιέχει *κυψέλες αποθήκευσης* (*storage cells*) που χρησιμοποιούνται για την υλοποίηση μιας μικρής συνάρτησης. Κάθε κυψέλη είναι μια μνήμη τύπου *SRAM* (*Static Random Access Memory*) και μπορεί να αποθηκεύσει μια λογική τιμή 0 ή 1. Η αποθηκευμένη τιμή μεταφέρεται στην έξοδο της κυψέλης αποθήκευσης. Μπορούν να αναπτυχθούν πίνακες LUT σε διάφορα μεγέθη, όπου το μέγεθος ορίζεται από τον αριθμό των εισόδων. Ένα LUT n-εισόδων μπορεί να υλοποιήσει όλες τις πιθανές συνδυαστικές συναρτήσεις των n-εισόδων, προσθέτοντας μία ακόμη είσοδο είναι δυνατή η αναπαράσταση πιο σύνθετων συναρτήσεων. Μελέτες έχουν δείξει ότι τα LUT 4-εισόδων είναι μία "καλή" λύση [40].



Εικόνα 2.17: [09] Πίνακας αναζήτησης (LUT, Lookup Table) αρχιτεκτονικής FPGA

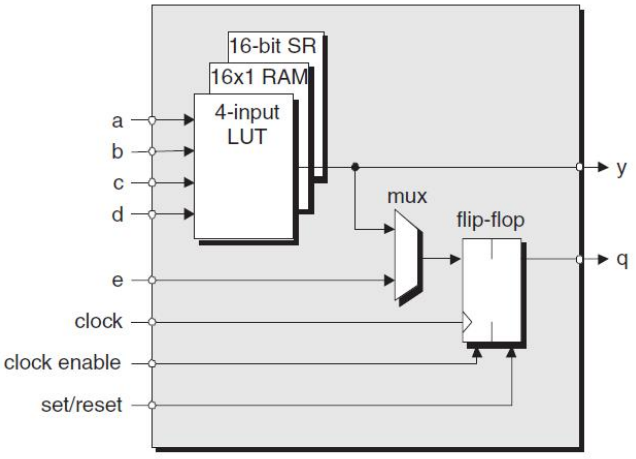
Οι κυψέλες αποθήκευσης των πινάκων LUT είναι *πτητικές* (ή *μη μόνιμες, volatile memory*), αυτό συνεπάγεται ότι χάνουν τα δεδομένα που περιέχουν εάν διακοπεί η τροφοδοσία του ΟΚ. Δεδομένου αυτού του προβλήματος, θα έπρεπε το FPGA να προγραμματίζεται κάθε φορά που εφαρμόζεται η τροφοδοσία στο κύκλωμα. Για την αποφυγή αυτού του προβλήματος, συμπεριλαμβάνεται συχνά μαζί με τη μητρική πλακέτα του FPGA και ένα μικρό ΟΚ *μη-πτητικής μνήμης* (*non-volatile memory*). Αυτή η μνήμη είναι της μορφής *PROM* ή *EPROM* ή *EEROM* ή οτιδήποτε άλλο (Παράρτημα Β.3). Η μνήμη αυτή διατηρεί σε μόνιμη βάση τα δεδομένα της διάταξης τα οποία έχουν εισέλθει μέσω της θύρας JTAG (Εικόνα Β.10, Παράρτημα Β.3.4). Κάθε

φορά που γίνεται τροφοδοσία στη συσκευή FPGA, διαβάζεται η μνήμη και διαμορφώνεται στη συσκευή η συγκεκριμένη λειτουργία για την οποία σχεδιάστηκε.

2.2.8 Λογικό Στοιχείο (LE, Logic Element)

Λογικό στοιχείο (*LE, Logic Element*), σε αυτό το σημείο θα πρέπει να αναλυθεί λίγο την ύπαρξη του λογικού στοιχείου. Το λογικό στοιχείο αποτελεί την ελάχιστη λογική μονάδα στην οποία μπορεί να αποθηκευτεί μια συνάρτηση ή ένα δεδομένο στο FPGA. Στη βιβλιογραφία όπως και στην αγορά υπάρχει μία πληθώρα ονομασία αυτών των στοιχείων. Μπορεί κανείς να τα δει ως *λογικό κύτταρο* (*logic cell*), ως λογικό στοιχείο (logic element), ως *κύτταρο* (*cell*) κτλ. Σε όλη τη μεταπτυχιακή διατριβή θα αναφέρεται αυτή τη λογική μονάδα ως λογικό στοιχείο (LE).

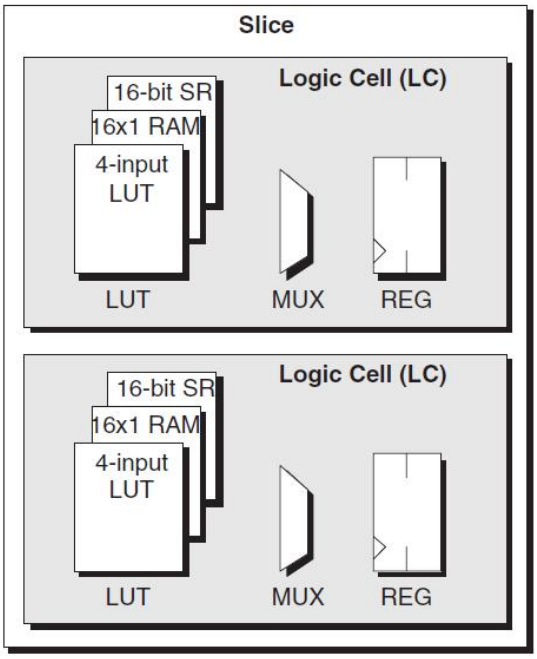
Συνήθως κάθε λογικό στοιχείο περιέχει ένα πίνακα αναζήτησης (LUT). Ο πίνακας LUT αποτελεί μέρος ενός λογικού στοιχείου, άρα η ελάχιστη λογική μονάδα στην οποία μπορεί να αποθηκευθεί ένα δεδομένο στο FPGA είναι το λογικό στοιχείο (LE). Ένας πίνακας LUT μπορεί να διαφέρει στον αριθμό των εισόδων ανά αρχιτεκτονική σχεδίαση, όπως φαίνεται στην Εικόνα 2.17 ο πίνακας LUT είναι 3 εισόδων ενώ στην Εικόνα 2.18 είναι 4 εισόδων. Πέραν των προαναφερθέντων, ένα λογικό στοιχείο περιέχει επίσης έναν αριθμό από λογικές πύλες, κυκλώματα, όπως και κύτταρα μνήμης flip flop (Εικόνα 2.18). Επιπλέον ένας πίνακας LUT, υπό τη λογική δομή του λογικού στοιχείου, μπορεί να χρησιμοποιηθεί ως μνήμη RAM ή ως καταχωρητής ολίσθησης. Στην Εικόνα 2.18 φαίνεται ότι το συγκεκριμένο λογικό στοιχείο μπορεί να τροποποιηθεί και ως μνήμη RAM 16×1 ή ως καταχωρητής ολίσθησης 16-bit. Είναι αναμενόμενο λοιπόν ότι η αρχιτεκτονική ενός λογικού στοιχείο να διαφέρει ανά εταιρία και οικογένεια συσκευών.



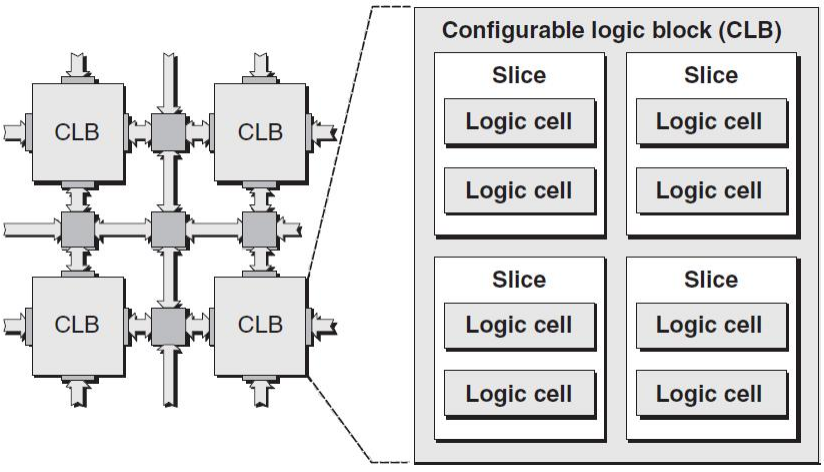
Εικόνα 2.18: [09] Λογικό στοιχείο της εταιρίας Xilinx

2.2.9 Διαμορφωμένη Λογική Βαθμίδα (CLB, Configurable Logic Block)

Η αμέσως επόμενη μεγαλύτερη λογική μονάδα είναι ένα σύνολο από λογικά στοιχεία. Υπάρχουν διαφορετικές αρχιτεκτονικές όπου κάποιες από αυτές περιλαμβάνουν δυο λογικά στοιχεία και αποτελούν μία "φέτα" (slice), Εικόνα 2.19. Κάποιες άλλες απλώς έχουν πολύ περισσότερα λογικά στοιχεία, γύρω στις 8 με 10 και αποτελούν μια *διαμορφωμένη λογική βαθμίδα* (*CLB, Configurable Logic Block*), Εικόνα 2.20. Οι διαμορφωμένες λογικές βαθμίδες (CLB) έχουν αναφερθεί και ως λογικές βαθμίδες (LB), αλλά μπορεί να εμφανιστούν και ως *συστοιχία λογικών βαθμίδων* (*LAB, Logic Array Block*). Στην παρούσα μεταπτυχιακή διατριβή θα αναφέρονται ως διαμορφωμένες λογικές βαθμίδες (CLB).



Εικόνα 2.19: [09] Ένα slice που περιέχει δυο λογικά στοιχεία της εταιρίας Xilinx



Εικόνα 2.20: [09] Διαμορφωμένη λογική βαθμίδα (CLB, Configurable Logic Block) της εταιρίας Xilinx

Ένα άλλο χαρακτηριστικό που έχει να κάνει με τα λογικά στοιχεία και ειδικά με τους πίνακες αναζήτησης (LUT), είναι ότι συγκεκριμένα κάποια από αυτά μπορούν να λειτουργήσουν ως μνήμη ή ως καταχωρητές ολίσθησης. Χρησιμοποιείται η λέξη κάποια επειδή υπάρχουν εταιρίες που δεν ενσωματώνουν αυτή την παραλλαγή στην αρχιτεκτονική τους. Όπως φαίνεται και στην Εικόνα 2.18 ένα πίνακα αναζήτησης 4-εισόδων μπορεί να λειτουργήσει ως μνήμη 16×1 RAM ή ως 16-bit καταχωρητής ολίσθησης. Με αυτόν τον τρόπο μια διαμορφωμένη λογική βαθμίδα (CLB) μπορεί να επεκτείνει αυτή τη λειτουργία σε όλη τη λογική της δημιουργώντας μεγαλύτερη μνήμη όπως και μεγαλύτερο καταχωρητή ολίσθησης. Το μέγεθος της μνήμης και του καταχωρητή ολίσθησης εξαρτάται από το μέγεθος των λογικών στοιχείων που περιέχει κάθε διαμορφωμένη λογική βαθμίδα σε μία αρχιτεκτονική σχεδίαση.

2.2.10 Γρήγορες Αλυσίδες Κρατουμένου (Fast Carry Chains)

Γρήγορες αλυσίδες κρατουμένου (*Fast Carry Chains*), είναι μία ακόμη πρόσθετη λογική που υλοποιούν τα λογικά στοιχεία. Παράδειγμα, από τα προηγούμενα που έχουν αναφερθεί, για να επεκταθεί το μέγεθος τους πρέπει να συνδεθούν οι αλυσίδες δυο λογικών στοιχείων (LE) μεταξύ τους, στη συνέχεια δυο "φέτες" (slices) και τέλος δύο διαμορφωμένες λογικές βαθμίδες (CLB). Σε κάθε περίπτωση αυτή η σειρά σύνδεσης των αλυσίδων κρατουμένου, εξαρτάται από την αρχιτεκτονική δομή του εκάστοτε FPGA. Οι γρήγορες αλυσίδες κρατουμένου βελτιώνουν την απόδοση αριθμητικών κυκλωμάτων όπως είναι οι αθροιστές κτλ.

Σε αυτό το σημείο να αναφερθεί ότι κάποια επιπλέον χαρακτηριστικά μπορούν να υπάρξουν μεμονωμένα και ανεξάρτητα σε κάποιες αρχιτεκτονικές δομές FPGA. Αυτά δεν αποτελούν γενικό χαρακτηριστικό αλλά ειδικό, κάποια από αυτά παρουσιάζονται στην αρχιτεκτονική της πλακέτας Spartan-3 και Spartan-3E στο Παράρτημα Γ.

2.3 Αναφορές

[09] Clive "Max" Maxfield. «The Design Warrior's Guide to FPGAs». Mentor Graphics Corporation and Xilinx, Inc. Elsevier, ISBN 0-7506-7604-3, Copyright 2004.

[16] Morris Mano. «Ψηφιακή σχεδίαση, 3 Έκδοση». Εκδόσεις Παπασωτηρίου, ISBN 960-7530-63-2, Αθήνα 2003.

[26] Stephen Brown, Zvonko Vranesic. «Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL». Εκδόσεις Τζιόλα Θεσσαλονίκη, ISBN 960-8050-50-2O, Θεσσαλονίκη 2001.

[40] Μιχάλης Ψαράκης. «Σημειώσεις / Διαφάνειες Διδασκαλίας, Προηγμένης Ψηφιακής Σχεδίασης». Πανεπιστήμιο Πειραιώς. Τμήμα Πληροφορικής, Πρόγραμμα Μεταπτυχιακών Σπουδών "Προηγμένα συστήματα πληροφορικής", κατεύθυνση "Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων". Πειραιάς 2009.

Κεφάλαιο 3

Γλώσσα Επερωτήσεων SQL και Επέκταση τους σε VHDL

Σε αυτό το κεφάλαιο παρουσιάζεται το πώς μια επερώτηση σε γλώσσα SQL μεταφράζεται σε γλώσσα VHDL. Για την καλύτερη κατανόηση του κεφαλαίου θα γίνει μια πρώτη ανασκόπηση σε έννοιες της Σχεσιακής Άλγεβρας και κατ' επέκταση της γλώσσας SQL. Και κατόπιν αυτού θα παρουσιαστούν οι μετατροπές και τα οφέλη αυτής της προσέγγισης.

Αναλυτικά στην Ενότητα 3.1 παρουσιάζεται η Σχεσιακή Άλγεβρας και συγκεκριμένα οι πράξεις της *Επιλογής- SELECT* (Ενότητα 3.1.1), της *Προβολής - PROJECT* (Ενότητα 3.1.2) και η πράξη της *Συνένωση - JOIN* με τις επεκτάσεις της (Ενότητα 3.1.3). Στην Ενότητα 3.2 παρουσιάζεται η γλώσσα επερωτήσεων SQL και συγκεκριμένα παρουσιάζονται τέσσερις επερωτήσεις η Q1 (Ενότητα 3.2.2), η Q2 (Ενότητα 3.2.3), η Q3 (Ενότητα 3.2.4) και η Q4 (Ενότητα 3.2.5). Στην Ενότητα 3.3 παρουσιάζεται η επεκτάσεις των τεσσάρων επερωτήσεων της SQL στη γλώσσα περιγραφής υλικού VHDL, συγκεκριμένα παρουσιάζονται οι επεκτάσεις της Q1 (Ενότητα 3.3.1), της Q2 (Ενότητα 3.3.2), της Q3 (Ενότητα 3.3.3) και της Q4 (Ενότητα 3.3.4).

3.1 Σχεσιακή Άλγεβρα

Το βασικό σύνολο των πράξεων του σχεσιακού μοντέλου είναι η σχεσιακή άλγεβρα. Οι πράξεις αυτές δίνουν τη δυνατότητα στο χρήστη να προσδιορίσει βασικά αιτήματα ανάκτησης. Το αποτέλεσμα μιας ανάκτησης είναι πάλι μια νέα σχέση, αυτή μπορεί να σχηματιστεί από μία ή περισσότερες άλλες σχέσεις. Η σχεσιακή άλγεβρα είναι πολύ σημαντική μιας και αποτελεί τη βάση για άλλα *σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων* (*ΣΣΔΒΔ*), όπως επίσης και μερικές από τις έννοιες της ενσωματώνονται στην SQL. Η SQL από την άλλη, είναι η τυπική γλώσσα επερωτήσεων για ΣΣΔΒΔ. Αν και κανένα από τα εμπορικά ΣΣΔΒΔ δεν υποστηρίζει διεπαφή για επερωτήσεις σχεσιακής άλγεβρας, ο πυρήνας των πράξεων και των συναρτήσεων του κάθε σχεσιακού συστήματος βασίζεται στις πράξεις της σχεσιακής άλγεβρας [19]. Παρακάτω θα αναφερθούν μερικές από τις πράξεις αυτές, καθώς και την αλληλουχία των βημάτων που επιτελούν τόσο σε μια κοινή CPU όσο και στο μοντέλο της VHDL

3.1.1 Η Πράξη της Επιλογής (SELECT)

Η πράξη *Επιλογή* (*SELECT*) χρησιμοποιείται για την επιλογή ενός υποσύνόλου πλειάδων μιας σχέσης που ικανοποιεί μια *συνθήκη επιλογής*. Η πράξη της Επιλογής συμβολίζεται με το ελληνικό γράμμα ‘ σ ’ και είναι της μορφής που φαίνεται στον Τύπο 3.1

$$\sigma_{<\text{συνθήκη επιλογής}>}(<\text{όνομα σχέσης}>) \qquad (3.1)$$

- Όπου ‘σ’ είναι ο μοναδιαίος τελεστής (δηλ εφαρμόζεται σε μια σχέση) της πράξης της Επιλογής,
- <*συνθήκη επιλογής* > είναι η συνθήκη που θα εφαρμοστεί ανεξάρτητα σε κάθε πλειάδα της σχέσης. Αν η συνθήκη είναι *αληθής* (*true*) τότε επιλέγεται η πλειάδα, αν είναι *ψευδής* (*false*) τότε απορρίπτεται.
- Και <*όνομα σχέσης*> είναι το όνομα της σχέσης στην οποία θα εφαρμοστεί η Επιλογή.

Αν τώρα ορισθεί το όνομα μια σχέσης ως **R** και τα γνωρίσματα αυτής ως (**A_1, A_2, A_3**) όπως αυτή που φαίνεται στην Εικόνα 3.1, τότε αν χρειαστεί να βρεθούν οι πλειάδες της σχέσης (R) που έχουν το χαρακτηριστικό γνώρισμα $A_2 = 2$, τότε θα έπρεπε να γράφει η πράξη της επιλογής που φαίνεται στον Τύπο 3.2. Το αποτέλεσμα αυτής της πράξης είναι εμφανές στην Εικόνα 3.2.

$$\sigma_{A_2=2}(R) \qquad (3.2)$$

R

A_1	A_2	A_3
1	4	5
7	9	8
5	3	5
4	2	1
2	7	4
1	2	2

Εικόνα 3.1: Η Σχέση R με τα γνωρίσματα της (A_1, A_2, A_3)

Π_1

A_1	A_2	A_3
4	2	1
1	2	2

Εικόνα 3.2: Τα αποτελέσματα της πράξης της Επιλογής (Select) του Τύπου 3.2 στη σχέση R

Αν χρειαστεί τώρα να προστεθεί μια ακόμη συνθήκη επιλογής, τότε αυτή προστίθεται δίπλα από την πρώτη χρησιμοποιώντας τον κατάλληλο τελεστή. Σε συνέχεια του παραδείγματος θα επιλέγει και το γνώρισμα A3 = 1. Η πράξη της επιλογής θα γίνει ίδια με αυτή του Τύπου 3.3 και το αποτέλεσμα αυτής φαίνεται στην Εικόνα 3.3.

$$\sigma_{A_2=2\ AND\ A_3=1}(R) \qquad (3.3)$$

Π_2

A_1	A_2	A_3
4	2	1

Εικόνα 3.3: Τα αποτελέσματα της πράξης της Επιλογής (Select) του Τύπου 3.3 στη σχέση R

3.1.2 Η Πράξη της Προβολής (PROJECT)

Αν θεωρήσουμε μια σχέση ως πίνακα, η πράξη της επιλογής επιλέγει ορισμένες γραμμές από τον πίνακα σύμφωνα με τη συνθήκη επιλογής. Η πράξη *Προβολή* (*PROJECT*) επιλέγει ορισμένες στήλες από τον πίνακα σύμφωνα με τη *λίστα γνωρισμάτων*. Η πράξη της Προβολής συμβολίζεται με το ελληνικό γράμμα ' π ' και είναι της μορφής που φαίνεται στον Τύπο 3.4

$$\pi_{<\lambda ίστα\ γνωρισμάτων>}(<\acute{o}νομα\ σχέσης>) \qquad (3.4)$$

- Όπου '*π*' είναι ο μοναδιαίος τελεστής (δηλ εφαρμόζεται σε μια σχέση) της πράξης της Προβολής,
- <*λίστα γνωρισμάτων*> είναι μια λίστα από γνωρίσματα της σχέσης,
- Και <*όνομα σχέσης*> είναι το όνομα της σχέσης στην οποία θα εφαρμοστεί η Προβολής.

Η προκύπτουσα σχέση έχει μόνο τα γνωρίσματα που προσδιορίζονται από τη <*λίστα γνωρισμάτων*>, εμφανίζονται με την ίδια διάταξη με αυτήν που δηλώθηκαν στη <*λίστα γνωρισμάτων*>. Άρα ο βαθμός της προβολή είναι ίσος με το πλήθος των γνωρισμάτων στη <*λίστα γνωρισμάτων*>. Η πράξη της προβολής απομακρύνει διπλές πλειάδες, έτσι ώστε το αποτέλεσμα να είναι ένα σύνολο πλειάδων και επομένως μια έγκυρη σχέση. Αν ορισθεί το όνομα μια σχέσης ως **S** και τα γνωρίσματα αυτής ως (**B_1, B_2, B_3**) όπως αυτή που φαίνεται στην Εικόνα 3.4, τότε αν χρειαστεί να εμφανιστούν τα γνωρίσματα B1 και B2 της σχέσης (S), τότε θα έπρεπε να γράφει η πράξη της προβολής όπως φαίνεται στον Τύπο 3.5. Το αποτέλεσμα αυτής της πράξης είναι εμφανές στην Εικόνα 3.5.

$$\pi_{B_1,B_2}(S) \qquad (3.5)$$

S

B_1	B_2	B_3
2	4	8
1	1	1
2	5	5
1	2	1
2	5	4
1	2	3
4	6	9
8	4	2
1	1	3
3	7	2
5	3	1

Εικόνα 3.4: Η Σχέση S με τα γνωρίσματα της (B_1, B_2, B_3)

Π_3

B_1	B_2
2	4
1	1
2	5
1	2
4	6
8	4
3	7
5	3

Εικόνα 3.5: Τα αποτελέσματα της πράξης της Προβολής (Project) του Τύπου 3.5 στη σχέση S

Αν χρειαστεί τώρα να προστεθεί (ή να αφαιρεθεί) ένα ακόμη γνώρισμα, τότε αυτό προστίθεται (ή αφαιρείται) από τη λίστα γνωρισμάτων. Σε συνέχεια του παραδείγματος θα αφαιρεθεί το γνώρισμα B2. Η πράξη της προβολής θα γίνει ίδια με αυτή του Τύπου 3.6 και το αποτέλεσμα αυτής φαίνεται στην Εικόνα 3.6. Όπως διαπιστώνεται οι διπλές πλειάδες έχουν απαλειφθεί.

$$\pi_{B_1}(S) \qquad (3.6)$$

Π_4

B_1
2
1
4
8
3
5

Εικόνα 3.6: Τα αποτελέσματα της πράξης της Προβολής (Project) του Τύπου 3.6 στη σχέση S

3.1.3 Η Πράξη της Συνένωσης (JOIN)

Η πράξη *Συνένωση* (*JOIN*) χρησιμοποιείται για να συνδυαστούν σε ενιαίες πλειάδες κάποιες *σχετιζόμενες πλειάδες* από δυο σχέσεις. Είναι μια πολύ σημαντική πράξη για κάθε σχεσιακή βάση δεδομένων, μιας και επιτρέπει την *επεξεργασία συσχετίσεων* μεταξύ σχέσεων, χρησιμοποιείται πολύ συχνά όταν προσδιορίζονται επερωτήσεις σε βάση δεδομένων. Η πράξη της Συνένωσης συμβολίζεται με το ⋈ και είναι της μορφής που φαίνεται στον Τύπο 3.7

$$(<\text{όνομα σχέσης } A>) \triangleright\triangleleft_{<\text{συνθήκη συνένωσης}>} (<\text{όνομα σχέσης } B>) \quad (3.7)$$

- Όπου '⋈' είναι ο τελεστής της πράξης της Συνένωσης,
- <*συνθήκη συνένωσης*> είναι η συνθήκη που θα εφαρμοστεί ανεξάρτητα για κάθε μια πλειάδα των σχέσεων Α και Β και η σύγκριση γίνεται μεταξύ των γνωρισμάτων των εμπλεκομένων σχέσεων,
- <*όνομα σχέσης Α*> και <*όνομα σχέσης Β*> είναι τα ονόματα των σχέσεων στα οποία θα εφαρμοστεί η <*συνθήκη συνένωσης*>.

Η προκύπτουσα σχέση έχει μόνο τους συνδυασμούς των πλειάδων που ικανοποιούν την <*συνθήκη συνένωσης*>. Οι πλειάδες των οποίων τα γνωρίσματα συνένωσης έχουν τιμή null δεν εμφανίζονται.

Σε συνέχεια των παραδειγμάτων, επιλέγεται η σχέση **R** (Εικόνα 3.1) και η σχέση **S** (Εικόνα 3.4) για την εφαρμογή της πράξης της Συνένωσης. Η <*συνθήκη συνένωσης*> που θα εφαρμοστεί είναι αυτή της ταύτισης των γνωρισμάτων A_1 και B_1, η πράξη της συνένωσης που πρέπει να

διατυπωθεί είναι αυτή που φαίνεται στον Τύπο 3.8. Το αποτέλεσμα αυτής της πράξης είναι εμφανές στην Εικόνα 3.7.

$$(R) \triangleright\triangleleft_{A_1 = B_1} (S) \qquad (3.8)$$

Π_5

A_1	A_2	A_3	B_1	B_2	B_3
1	4	5	1	1	1
1	4	5	1	2	1
1	4	5	1	2	3
1	4	5	1	1	3
5	3	5	5	3	1
4	2	1	4	6	9
2	7	4	2	4	8
2	7	4	2	5	5
2	7	4	2	5	4
1	2	2	1	1	1
1	2	2	1	2	1
1	2	2	1	2	3
1	2	2	1	1	3

Εικόνα 3.7: Τα αποτελέσματα της εφαρμογής της πράξης της Συνένωσης (JOIN) του Τύπου 3.8 επάνω στις σχέσεις R και S.

Συνένωση Ισότητας (Equijoin)

Το παραπάνω αποτέλεσμα της πράξης (3.8) επειδή περιλαμβάνει μόνο σύγκριση ισότητας (δηλ χρησιμοποιείται μόνο ο τελεστής '=') ονομάζεται και αλλιώς *Συνένωση Ισότητας* (*Equijoin*). Χαρακτηριστικό αυτής της πράξης είναι ότι τα γνωρίσματα τα οποία βρίσκονται στην <*συνθήκη συνένωσης*> έχουν τις ίδιες τιμές σε κάθε πλειάδα στην προκύπτουσα σχέση. Στο παράδειγμα που υλοποιήθηκε (Εικόνα 3.7) φαίνεται ότι τα γνωρίσματα A1 και B1 έχουν τις ίδιες τιμές.

Θήτα Συνένωση

Αν τώρα χρειάζεται να χρησιμοποιηθούν επιπλέον τελεστές σύγκρισης $\{\leq, \geq, \neq, >, <, =\}$ στην <*συνθήκη συνένωσης*>, τότε θα πρέπει να είναι της μορφής :

$$< συνθήκη > AND < συνθήκη > AND.....AND < συνθήκη >$$

Σε αυτήν την περίπτωση η πράξη της συνένωσης με τόσο γενική μορφή λέγεται *Θήτα Συνένωση* (*Theta Join*). Το αποτέλεσμα της πράξης Θήτα Συνένωση είναι μια νέα σχέση όπου το περιεχόμενο της ικανοποιεί κάθε συνθήκη της <*συνθήκη συνένωσης*>, δηλαδή αποτιμάται αληθής (true) κάθε μια συνθήκη. Τώρα ο κάθε συνδυασμός πλειάδων για τον οποίο η <*συνθήκη συνένωσης*> αποτιμάται αληθής (true) για τις τιμές των γνωρισμάτων του, συμπεριλαμβάνεται στο τελικό αποτέλεσμα ως μια πλειάδα.

Σε συνέχεια των παραδειγμάτων, επιλέγεται η σχέση **R** (Εικόνα 3.1) και η σχέση **S** (Εικόνα 3.4) για την εφαρμογή της πράξης της Θήτα Συνένωσης. Η <*συνθήκη συνένωσης*> που θα εφαρμοστεί είναι : το γνώρισμα A_2 να είναι μεγαλύτερο ή ίσο με το γνώρισμα B_3, και το γνώρισμα A_3 να είναι μικρότερο από το γνώρισμα B_2. Η πράξη της θήτα συνένωσης που πρέπει να διατυπωθεί είναι αυτή που φαίνεται στον Τύπο 3.9. Το αποτέλεσμα αυτής της πράξης είναι εμφανές στην Εικόνα 3.8.

$$(R) \triangleright\!\triangleleft_{A_2 \geq B_3 \ AND \ A_3 < B_2} (S) \quad (3.9)$$

Π_6

A_1	A_2	A_3	B_1	B_2	B_3
1	4	5	3	7	2
5	3	5	3	7	2
4	2	1	1	2	1
4	2	1	8	4	2
4	2	1	3	7	2
4	2	1	5	3	1
2	7	4	2	5	5
2	7	4	2	5	4
2	7	4	3	7	2
1	2	2	8	4	2
1	2	2	3	7	2
1	2	2	5	3	1

Εικόνα 3.8: Τα αποτελέσματα της εφαρμογής της πράξης της Θήτα Συνένωσης (Theta Join) του Τύπου 3.9 επάνω στις σχέσεις R και S

Φυσική Συνένωση (Natural Join)

Όπως διαπιστώθηκε στην πράξη της *Συνένωση Ισότητας* (*Equijoin*), στην προκύπτουσα σχέση υπήρχε πλεονάζον πληροφορία μιας και ένα ζεύγος γνωρισμάτων είχε ίδιες τιμές. Για την απαλοιφή αυτής της ιδιορρυθμίας δημιουργήθηκε μια νέα πράξη η οποία διώχνει αυτόν τον πλεονασμό. Η νέα πράξη λέγεται *Φυσική Συνένωση* (*Natural Join*) η οποία δεν περιλαμβάνει στο αποτέλεσμα της δεύτερο ίδιο γνώρισμα (για κάθε πλειάδα). Ουσιαστικά η φυσική συνένωση είναι συνένωση ισότητας ακολουθούμενη από απαλοιφή των γνωρισμάτων που πλεονάζουν.

Ο τυπικός ορισμός της φυσικής συνένωσης απαιτεί τα δύο γνωρίσματα της συνένωσης να έχουν το ίδιο όνομα. Αν αυτό δεν ισχύει τότε θα πρέπει να εφαρμοστεί πρώτα η πράξη της *Μετονομασίας* (*Rename*) στο ένα από τα δύο γνωρίσματα που εμπλέκονται.

Σημείωση: Η πράξη της *Μετονομασίας* (*Rename*) συμβολίζεται με το ελληνικό γράμμα 'ρ' και μπορεί να μετονομάσει το όνομα μιας σχέσης, ή τα ονόματα των γνωρισμάτων, ή και τα δύο. Η γενική πράξη Μετονομασίας όταν εφαρμοστεί σε μια σχέση **W** βαθμού **n** (γνωρισμάτων) συμβολίζεται με τον παρακάτω τρόπο (μια από τις τρεις μορφές) :

$$\rho_{G\left(Z_1,Z_2,Z_3.....Z_n\right)}(W) \quad (3.10)$$

Όπου Z_1,Z_2,Z_3 ….. Z_n είναι τα νέα ονόματα των γνωρισμάτων και G το νέο όνομα της σχέσης.

Η πράξη της Φυσικής Συνένωσης λοιπόν συμβολίζεται με το * και είναι της μορφής που φαίνεται στον Τύπο 3.11

$$(<\text{όνομα σχέσης } A>) *_{\rho<\text{νέα ονόματα}>} (<\text{όνομα σχέσης } B>) \quad (3.11)$$

- Όπου ' * ' είναι ο τελεστής της πράξης της Φυσικής Συνένωσης,
- ρ<*νέα ονόματα*> είναι τα νέα ονόματα των γνωρισμάτων, είναι προαιρετικό και χρησιμοποιείται εκεί που δεν υπάρχουν γνωρίσματα με το ίδιο όνομα. Αν υπάρχουν γνωρίσματα με το ίδιο όνομα τότε η πράξη γίνεται χωρίς την παρουσία του τελεστή 'ρ'.

- <*όνομα σχέσης Α*> και <*όνομα σχέσης Β*> είναι τα ονόματα των σχέσεων στα οποία θα εφαρμοστεί η πράξη της Φυσικής Συνένωσης.

Η προκύπτουσα σχέση έχει μόνο τους συνδυασμούς των πλειάδων που ικανοποιούν τη συνένωση ισότητας καθώς και οι πλειάδες των οποίων τα γνωρίσματα έχουν ίδια τιμή δεν εμφανίζονται.

Σε συνέχεια των παραδειγμάτων, επιλέγεται η σχέση **R** (Εικόνα 3.1) και η σχέση **S** (Εικόνα 3.4) για την εφαρμογή της πράξης της Φυσικής Συνένωσης. Θα μετονομάσουμε το γνώρισμα B1 σε A1 ώστε να γίνει η πράξη της φυσικής συνένωσης και να προκύψει το επιθυμητό αποτέλεσμα. Η πράξη της φυσικής συνένωσης που πρέπει να διατυπωθεί είναι αυτή που φαίνεται στον Τύπο 3.12. Το αποτέλεσμα αυτής της πράξης είναι εμφανές στην Εικόνα 3.9.

$$(R) *_{\rho<A_1,B_2,B_3>} (S) \qquad (3.12)$$

Π_7

A_1	A_2	A_3	B_2	B_3
1	4	5	1	1
1	4	5	2	1
1	4	5	2	3
1	4	5	1	3
5	3	5	3	1
4	2	1	6	9
2	7	4	4	8
2	7	4	5	5
2	7	4	5	4
1	2	2	1	1
1	2	2	2	1
1	2	2	2	3
1	2	2	1	3

Εικόνα 3.9: Τα αποτελέσματα της εφαρμογής της πράξης της Φυσικής Συνένωσης (Natural Join) του Τύπου 3.12 επάνω στις σχέσεις R και S

Κλείνοντας αυτή τη ενότητα να αναφερθεί ότι υπάρχουν πολλές ιδιότητες και πράξεις τελεστών μπορούν να εφαρμοστούν στις πράξεις της Σχεσιακής άλγεβράς όπως αυτή της Επιλογής, της Προβολής και της Συνένωσης με των παράγωγων της. Από αυτές όμως θα εφαρμοστούν κάποιες ελάχιστες στην παρούσα μεταπτυχιακή διατριβή, συγκεκριμένα θα χρησιμοποιηθούν αυτοί οι τελεστές που χρειάζονται για την υλοποίηση των ερωτημάτων. Για περισσότερη λεπτομέρεια στους τελεστές και τις ιδιότητες ο αναγνώστης μπορεί να δει το ενδεικτικό βιβλίο, *«Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων»* [19] και *«Βάσεις Δεδομένων»* [39] που παραθέτεται στη βιβλιογραφία αλλά και όποιο άλλο βιβλίο βάσεων δεδομένων βρει διαθέσιμο.

3.2 Γλώσσα Επερωτήσεων SQL

Όπως αναφέρθηκε στην Ενότητα 3.1 η σχεσιακή άλγεβρα αποτελεί τη βάση για άλλα *σχεσιακά συστήματα διαχείρισης βάσεων δεδομένων* (*ΣΣΔΒΔ*), όπως επίσης και μερικές από τις έννοιες της ενσωματώνονται στην SQL (Structured Query Language, Δομημένη Γλώσσα Επερωτήσεων). Αυτό σημαίνει ότι η γλώσσα SQL στηρίζεται σε ένα βαθμό από τις έννοιες της Σχεσιακής Άλγεβρας και ως εκ τούτου είναι μια από τις επέκτασής της. Αποτελεί μια πιο εύκολη προσέγγιση στο να υποβάλει ο χρήστης κάποιες πράξεις (που μεταφράζονται ως ερωτήματα) σε ένα *συστήματα διαχείρισης βάσεων δεδομένων* (*ΣΔΒΔ*). Η γλώσσα SQL αποτελεί ένα διεθνές πρότυπο με το οποίο μπορεί κανείς, κάτω από ένα πλαίσιο κανόνων, να περιγράψει μια σειρά από ερωτήματα με απώτερο σκοπό την τροποποίηση δεδομένων σε ένα ΣΔΒΔ. Η επιτυχία της γλώσσας SQL οφείλεται στο γεγονός ότι είναι πολύ φιλική προς το χρήστη και οι αλλαγές που μπορεί υποστεί στη δομή της από τα διάφορα εμπορικά ΣΔΒΔ, του επιτρέπει να τη διαβάσει και να την προσαρμόσει στο αντίστοιχο περιβάλλον ενός άλλου ΣΔΒΔ. Αν τώρα η μεταφορά γίνει από ένα σύστημα ΔΒΔ Α’ σε ένα σύστημα Β’, όπου και τα δύο συστήματα έχουν την ίδια SQL, τότε τα ερωτήματα και οι εφαρμογές που θα μεταφερθούν, θα λειτουργήσουν χωρίς κάποια δυσκολία και με ελάχιστες τροποποιήσεις.

Μια επερώτηση στη σχεσιακή άλγεβρα γράφεται σαν μια ακολουθία από πράξεις που όταν εκτελεσθούν παράγουν το επιθυμητό αποτέλεσμα. Επομένως ο χρήστης πρέπει να προσδιορίσει τη σειρά εκτέλεσης των πράξεων για να ικανοποιηθεί η επερώτηση. Αντίθετα στην SQL το ΣΔΒΔ παρέχει μια υψηλού επιπέδου *δηλωτική* διεπαφή γλώσσας, τέτοια ώστε ο χρήστης να προσδιορίζει *μόνο* ποιο θα είναι το αποτέλεσμα και αφήνοντας στο ΣΔΒΔ να κάνει τη βελτιστοποίηση και τις αποφάσεις για το πώς θα εκτελεστεί η επερώτηση.

Η SQL χρησιμοποιεί τους όρους *πίνακες* (*table*), *γραμμή* (*row*) και *στήλη* (*column*) για τις έννοιες της σχέσης, πλειάδας και γνωρίσματος αντίστοιχα, όπως αυτές αναφέρθηκαν στη σχεσιακή άλγεβρα. Στην παρούσα μεταπτυχιακή διατριβή θα χρησιμοποιηθούν αυτοί οι όροι ως ισοδύναμοι. Η SQL είναι μια πλήρης γλώσσα βάσεων δεδομένων, διαθέτει εντολές για ορισμό δεδομένων, ερωτήσεις αλλά και ενημερώσεις. Επομένως είναι *Γλώσσα Ορισμού Δεδομένων* (*ΓΟΔ*) αλλά και *Γλώσσα Χειρισμού Δεδομένων* (*ΓΧΔ*). Υπάρχουν πολλά ερωτήματα και εντολές που μπορούν να υλοποιηθούν με την SQL, στην παρούσα ενότητα θα μελετηθεί (αναφερθεί) μόνο η εντολή *SELECT*, η οποία είναι εντολή για ανάκτηση πληροφοριών από τη βάση δεδομένων. Η εντολή SELECT δεν έχει σχέση με την πράξη της Επιλογής (Select) της σχεσιακής άλγεβρας που μελετήθηκε στην Ενότητα 3.1.1.

3.2.1 Εντολή SELECT

Οι επερωτήσεις στην SQL μπορεί να γίνουν πολύ απλές αλλά και πολύ πολύπλοκες, στην παρούσα μεταπτυχιακή διατριβή ο αναγνώστης θα δει τέσσερις επερωτήσεις οι οποίες θα χρησιμεύσουν στα επόμενα στάδια ως δείγμα για την μετατροπή τους σε γλώσσα VHDL. Για περαιτέρω μελέτη και εμβάθυνση στις επερωτήσεις, ο αναγνώστης καλείται να δει την το ενδεικτικό βιβλίο, *«Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων»* [19] και *«Βάσεις Δεδομένων»* [39] που παραθέτεται στη βιβλιογραφία αλλά και όποιο άλλο βιβλίο βάσεων δεδομένων βρει διαθέσιμο.

Η βασική μορφή της εντολής SELECT σχηματίζεται από τρεις προτάσεις **SELECT**, **FROM** και **WHERE** και έχει την μορφή που φαίνεται στον Τύπο 3.13 :

```
SELECT    <λίστα γνωρισμάτων>
FROM      <λίστα πινάκων>
WHERE     <συνθήκη>                    (3.13)
```

- Η <λίστα γνωρισμάτων> είναι τα γνωρίσματα όπου οι τιμές τους πρέπει να ανακτηθούν από τους εμπλεκόμενους πίνακες (η εμφάνιση τους εξαρτάται από τη σειρά δήλωσης τους στη λίστα),

- Η <λίστα πινάκων> περιέχει τους εμπλεκόμενους πίνακες της επερώτησης, από τους πίνακες αυτούς θα εξαχθεί η ζητούμενη πληροφορία της επερώτησης,

- Και <συνθήκη> είναι η λογική έκφραση αναζήτησης η οποία προσδιορίζει τις πλειάδες (γραμμές) που πρέπει να ανακτηθούν από την επερώτηση.

Στην SQL οι λογικοί τελεστές σύγκρισης τιμών $\{<=, >=, <>, >, <, =\}$ αντιστοιχούν με αυτούς της σχεσιακής άλγεβρας $\{\leq, \geq, \neq, >, <, =\}$ που αναφέρθηκαν στην Ενότητα 3.1.3.

3.2.2 Ερώτημα Q1

Η πρώτη επερώτηση **Q1** που θα υλοποιηθεί είναι αυτή που φαίνεται στον Τύπο 3.14.

```
Q1: SELECT   A1, A2, A3
    FROM     R
    WHERE    A3 > '2'          (3.14)
```

Ως συνέχεια των παραδειγμάτων, επιλέγεται η σχέση **R** (Εικόνα 3.1) για την εφαρμογή της πρώτης επερώτησης **Q1**. Η <λίστα γνωρισμάτων> περιλαμβάνει την εμφάνιση των γνωρισμάτων A_1, A_2 και A_3 της σχέσης R. Η <λίστα πινάκων> περιλαμβάνει μόνο τη σχέση R. Τέλος η <*συνθήκη*> που θα εφαρμοστεί είναι αυτή της ταύτισης των γνωρισμάτων A_3 της σχέσης R να είναι μεγαλύτερο του '2'. Το αποτέλεσμα της επερώτησης Q1 είναι εμφανές στην Εικόνα 3.10.

Q1

A_1	A_2	A_3
1	4	5
7	9	8
5	3	5
2	7	4

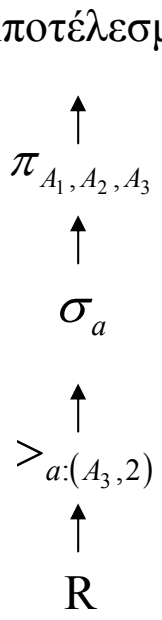
Εικόνα 3.10: Τα αποτελέσματα της επερώτησης Q1 του Τύπου 3.14 επάνω στη σχέση S.

Η επερώτηση Q1 είναι παρόμοια με την παρακάτω έκφραση της σχεσιακής άλγεβρας (Τύπος 3.15), εκτός από το ότι οι διπλότυπες (αν υπάρχουν) δεν θα απαλειφθούν.

$$\pi_{A_1, A_2, A_3}\left(\sigma_{A_3 > '2'}(R)\right) \qquad (3.15)$$

Επομένως, μία απλή επερώτηση SQL με ένα όνομα σχέσης στην πρόταση FROM είναι παρόμοια με ένα ζευγάρι πράξεων επιλογής – προβολής της σχεσιακής άλγεβρας. Η πρόταση SELECT της SQL προσδιορίζει τα *γνωρίσματα προβολής* (*projection attributes*) και η πρόταση WHERE τη *συνθήκη επιλογής* (*selection condition*). Η μόνη διαφορά είναι ότι με την SQL επερώτηση μπορεί να πάρουμε διπλότυπες πλειάδες στο αποτέλεσμα, αφού δεν ισχύει ο περιορισμός να είναι το αποτέλεσμα σύνολο [19].

Το αλγεβρικό πλάνο της επερώτησης Q1 είναι αυτό που φαίνεται στην Εικόνα 3.11. Το αλγεβρικό πλάνο μας καθορίζει πέραν της γραφικής απεικόνισης της επερώτησης και τη χρονική ακολουθία που υφίσταται ώστε να δώσει το επιθυμητό αποτέλεσμα. Στην παρούσα μεταπτυχιακή διατριβή η συγκεκριμένη πληροφορία είναι πολύ σημαντική μιας και θα αποτελεί μέτρο σύγκρισης των αποτελεσμάτων που θα παραχθούν κατά την εκτέλεση τους στη γλώσσα VHDL.



Εικόνα 3.11: Το αλγεβρικό πλάνο της επερώτησης Q1 του Τύπου 3.14 επάνω στη σχέση R

3.2.3 Ερώτημα Q2

Η δεύτερη επερώτηση **Q2** που θα υλοποιηθεί είναι αυτή που φαίνεται στον Τύπο 3.16.

$$\begin{aligned} &\text{Q2: SELECT} && B_1, B_2, B_3 \\ &\quad \text{FROM} && S \\ &\quad \text{WHERE} && B_1 = \text{'1' AND } B_2 > \text{'1'} \end{aligned} \qquad (3.16)$$

Ως συνέχεια των παραδειγμάτων, επιλέγεται η σχέση **S** (Εικόνα 3.4) για την εφαρμογή της δεύτερης επερώτησης **Q2**. Η <λίστα γνωρισμάτων> περιλαμβάνει την εμφάνιση των γνωρισμάτων B_1, B_2 και B_3 της σχέσης S. Η <λίστα πινάκων> περιλαμβάνει μόνο τη σχέση S. Τέλος η <*συνθήκη*> που θα εφαρμοστεί είναι αυτή της ταύτισης των γνωρισμάτων B_1 και B_2 της σχέσης S να είναι το μεν B_1 ίσο με '1' και B_2 να είναι μεγαλύτερο του '1'. Το αποτέλεσμα της επερώτησης Q2 είναι εμφανές στην Εικόνα 3.12.

Q2

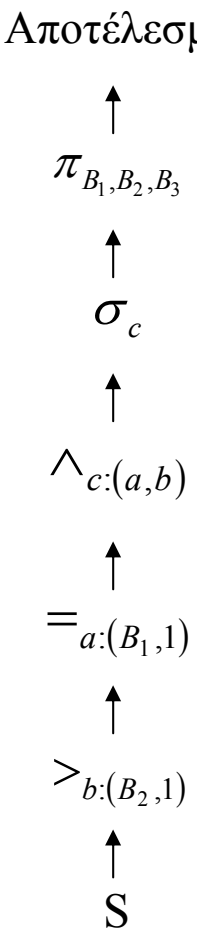
B_1	B_2	B_3
1	2	1
1	2	3

Εικόνα 3.12: Τα αποτελέσματα της επερώτησης Q1 του Τύπου 3.16 επάνω στη σχέση S.

Η επερώτηση Q2 είναι παρόμοια με την παρακάτω έκφραση της σχεσιακής άλγεβρας (Τύπος 3.17).

$$\pi_{B_1,B_2,B_3}\left(\sigma_{B_1='1' \; AND \; B_2>'1'}(S)\right) \qquad (3.17)$$

Το αλγεβρικό πλάνο της επερώτησης Q1 είναι αυτό που φαίνεται στην Εικόνα 3.13. Όπως μπορεί να διακρίνει ο αναγνώστης η <*συνθήκη*> χωρίζεται σε δύο ξεχωριστές πράξης όπου πρώτα εκτελείται η σύγκριση του μεγαλύτερου τελεστή «>» και κατόπιν αυτού ο τελεστής της ισότητας «=». Αυτή η ακολουθία των βημάτων, για την παραγωγή των δεδομένων από τη βάση δεδομένων, ακολουθείται από όλα τα σχεσιακά σχήματα μιας και είναι ο μοναδικός συμβατικός τρόπος εκτέλεσης των εντολών στα ΣΣΒΔ. Στην επόμενη ενότητα ο αναγνώστης θα δει πως αυτός ο χρόνος μπορεί να περιοριστή ακόμη περισσότερο χάριν της τεχνολογίας των συσκευών FPGA (αλλά και γενικός των συσκευών PLD) και της γλώσσας περιγραφής υλικού (HDL, VHDL).



Εικόνα 3.13: Το αλγεβρικό πλάνο της επερώτησης Q2 του Τύπου 3.16 επάνω στη σχέση S

3.2.4 Ερώτημα Q3

Η τρίτη επερώτηση **Q3** που θα υλοποιηθεί είναι αυτή που φαίνεται στον Τύπο 3.18.

Q3: SELECT	$A_1, A_2, A_3, B_1, B_2, B_3$	
FROM	R, S	
WHERE	$A_1 > '2'$ AND $A_2 = B_2$	(3.18)

Ως συνέχεια των παραδειγμάτων, επιλέγονται οι σχέσεις **R** (Εικόνα 3.1) και **S** (Εικόνα 3.4) για την εφαρμογή της τρίτης επερώτησης **Q3**. Η <λίστα γνωρισμάτων> περιλαμβάνει την εμφάνιση των γνωρισμάτων A_1, A_2, A_3, B_1, B_2 και B_3 των σχέσεων R και S. Η <λίστα πινάκων> περιλαμβάνει τις σχέσεις R και S. Τέλος η <*συνθήκη*> που θα εφαρμοστεί είναι αυτή της ταύτισης των γνωρισμάτων A_1, A_2, και B_2 των σχέσεων R και S, να είναι το μεν A_1 μεγαλύτερο από το '2' και τα A_2 και B_2 να είναι ίσα μεταξύ τους. Το αποτέλεσμα της επερώτησης Q3 είναι εμφανές στην Εικόνα 3.14.

Q3

A_1	A_2	A_3	B_1	B_2	B_3
5	3	5	5	3	1
4	2	1	1	2	1
4	2	1	1	2	3

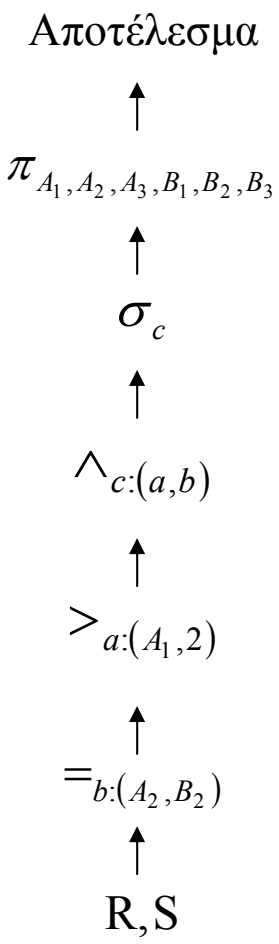
Εικόνα 3.14: Τα αποτελέσματα της επερώτησης Q3 του Τύπου 3.18 επάνω στις σχέσεις R και S.

Να σημειωθεί ότι στη λίστα γνωρισμάτων υπάρχει η δυνατότητα μετονομασίας των γνωρισμάτων και η εμφάνιση λιγότερων (γνωρισμάτων) από αυτών που υπάρχουν μέσα σε στις σχέσεις. Υπάρχουν πολλές επιπλέον δυνατότητες και ιδιότητες στη σύνταξη επερωτήσεων SQL αλλά αυτές δεν απασχόλησαν την έρευνα που υλοποιήθηκε στην παρούσα μεταπτυχιακή διατριβή. Για περισσότερη εμβάθυνση στη σύνταξη επερωτήσεων SQL παρακαλείται ο αναγνώστης να δει το ενδεικτικό βιβλίο, *«Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων»* [19] και *«Βάσεις Δεδομένων»* [39] που παραθέτεται στη βιβλιογραφία, αλλά και όποιο άλλο βιβλίο βάσεων δεδομένων βρει διαθέσιμο.

Η επερώτηση Q3 είναι παρόμοια με την έκφραση της σχεσιακής άλγεβρας του Τύπος 3.19. Είναι παρόμοια με μια σειρά πράξεων **επιλογής** – **προβολής** – **συνένωσης** της σχεσιακής άλγεβρας. Τέτοιες επερωτήσεις λέγονται επερωτήσεις *επιλογής – προβολής – συνένωσης* (*select – project – join queries*). Στην πρόταση WHERE της Q3, η συνθήκη $A_1 >$ ’2’ είναι συνθήκη επιλογής και αντιστοιχεί στην πράξη της επιλογής της σχεσιακής άλγεβρας. Η συνθήκη $A_2 = B_2$ είναι συνθήκη συνένωσης και αντιστοιχεί στη συνθήκη εκτέλεσης της πράξης συνένωσης της σχεσιακής άλγεβρας. Γενικά, οποιοδήποτε πλήθος συνθηκών επιλογής και συνένωσης μπορεί να προσδιορίζεται σε μία επερώτηση SQL. Ο διαχωρισμός και η σειρά εκτέλεσης των συνθηκών σε κάθε επερώτηση καθορίζεται από το ίδιο το ΣΣΒΔ.

$$\pi_{A_1, A_2, A_3, B_1, B_2, B_3}\left(\sigma_{A_1 > '2'}\left(R \rhd\lhd_{A_2 = B_2} S\right)\right) \qquad (3.19)$$

Το αλγεβρικό πλάνο της επερώτησης Q3 είναι αυτό που φαίνεται στην Εικόνα 3.15. Όπως μπορεί να διακρίνει ο αναγνώστης η <*συνθήκη*> χωρίζεται σε δύο ξεχωριστές πράξης όπου πρώτα εκτελείται η σύγκριση της ισότητας «=» μεταξύ των γνωρισμάτων A_2 και B_2 των σχέσεων R και S, εκτελείται δηλαδή πρώτα η πράξη της συνένωσης. Κατόπιν αυτού εκτελείται ο τελεστής του μεγαλύτερου «>» δηλαδή η πράξη της επιλογής.



Εικόνα 3.15: Το αλγεβρικό πλάνο της επερώτησης Q3 του Τύπου 3.18 επάνω στις σχέσεις R και S.

3.2.5 Ερώτημα Q4

Η τέταρτη και τελευταία επερώτηση **Q4** που θα υλοποιηθεί είναι αυτή που φαίνεται στον Τύπο 3.20.

Q4: SELECT	A_1, A_2, A_3, B_2, B_3	
FROM	R, S	
WHERE	$A_1 = B_1$	(3.20)

Ως συνέχεια των παραδειγμάτων, επιλέγονται οι σχέσεις **R** (Εικόνα 3.1) και **S** (Εικόνα 3.4) για την εφαρμογή της τέταρτης επερώτησης **Q4**. Η <λίστα γνωρισμάτων> περιλαμβάνει την εμφάνιση των γνωρισμάτων A_1, A_2, A_3, B_2 και B_3 των σχέσεων R και S. Η <λίστα πινάκων> περιλαμβάνει τις σχέσεις R και S. Τέλος η <*συνθήκη*> που θα εφαρμοστεί είναι αυτή της ταύτισης των γνωρισμάτων A_1, και B_1 των σχέσεων R και S, τέτοια ώστε να είναι ίσα μεταξύ τους. Το αποτέλεσμα της επερώτησης Q4 είναι εμφανές στην Εικόνα 3.16.

Q4

A_1	A_2	A_3	B_2	B_3
1	4	5	1	1
1	4	5	2	1
1	4	5	2	3
1	4	5	1	3
5	3	5	3	1
4	2	1	6	9
2	7	4	4	8
2	7	4	5	5
2	7	4	5	4
1	2	2	1	1
1	2	2	2	1
1	2	2	2	3
1	2	2	1	3

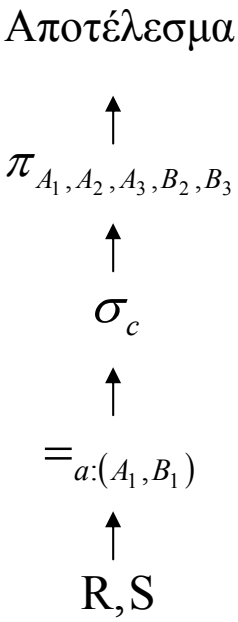
Εικόνα 3.16: Τα αποτελέσματα της επερώτησης Q4 του Τύπου 3.20 επάνω στις σχέσεις R και S.

Να σημειωθεί ότι στη λίστα γνωρισμάτων έγινε η χρήση της δυνατότητας εμφάνισης λιγότερων γνωρισμάτων (μόνο αυτών που χρειάζονται) για την εκτέλεση της επερώτησης. Όπως μπορεί να παρατηρήσει ο αναγνώστης, η συγκεκριμένη επερώτηση υλοποιεί ουσιαστικά την πράξη της φυσικής συνένωσης. Η ομοιότητα μπορεί να παρατηρηθεί βλέποντας την Εικόνα 3.9 και η διαφορά βλέποντας την Εικόνα 3.7. Σε αυτήν την επερώτηση το ΣΣΒΔ εκτελεί εσωτερικά όλες τις απαραίτητες ενέργειες ώστε να παραχθεί το επιθυμητό αποτέλεσμα. Η εκτέλεση της κάθε ενέργειας (τελεστές, προβολή κτλ) γίνεται ακολουθιακά και με τη σωστή σειρά. Αυτός ο τρόπος παραγωγής των δεδομένων έχει μια καθυστέρηση αντίστοιχη των τελεστών και των σχέσεων που υπάρχουν στην επερώτηση. Εκεί ακριβώς στηρίχθηκε και η έρευνα που υλοποιήθηκε με τα αποτελέσματα να είναι άκρως εντυπωσιακά., για περισσότερη λεπτομέρεια στην επόμενη ενότητα που ακολουθεί.

Η επερώτηση Q4 είναι παρόμοια με την έκφραση της σχεσιακής άλγεβρας του Τύπος 3.21. Είναι παρόμοια με μια σειρά πράξεων **επιλογής – προβολής –συνένωσης** της σχεσιακής άλγεβρας. Στην πρόταση WHERE της Q4, η συνθήκη $A_1 = B_1$ είναι συνθήκη συνένωσης και αντιστοιχεί στη συνθήκη εκτέλεσης της πράξης συνένωσης της σχεσιακής άλγεβρας. Από το αποτέλεσμα όμως φαίνεται ότι έχει υλοποιηθεί η αντίστοιχη πράξη της φυσικής συνένωσης. Από αυτό το παράδειγμα μπορεί να δει ο αναγνώστης την απλότητα και ευχρηστία της γλώσσας επερωτήσεων SQL.

$$\pi_{A_1, A_2, A_3, B_2, B_3}\left(\sigma\left(R \rhd\lhd_{A_1 = B_1} S\right)\right) \qquad (3.21)$$

Το αλγεβρικό πλάνο της επερώτησης Q4 είναι αυτό που φαίνεται στην Εικόνα 3.17. Όπως μπορεί να διακρίνει ο αναγνώστης στη <*συνθήκη*> εκτελείται η σύγκριση της ισότητας «=» μεταξύ των γνωρισμάτων A_1 και B_1 των σχέσεων R και S, εκτελείται δηλαδή η πράξη της συνένωσης. Κατόπιν αυτού εκτελείται η πράξη της επιλογής για την εμφάνιση των πλειάδων και τέλος η παρουσίαση των γνωρισμάτων από τις παραγόμενες πλειάδες. Το παραγόμενο αποτέλεσμα μπορεί να χαρακτηριστεί και ως φυσικής συνένωση των σχέσεων R και S.



Εικόνα 3.17: Το αλγεβρικό πλάνο της επερώτησης Q4 του Τύπου 3.20 επάνω στις σχέσεις R και S.

3.3 Επέκταση σε VHDL

Έχοντας γνώση για το πώς υλοποιούνται και εκτελούνται οι πράξεις της σχεσιακή άλγεβρα αλλά και της γλώσσας επερωτήσεων SQL, σε αυτήν την ενότητα θα αναλυθεί ο τρόπος με τον οποίο θα μπορούσε μια τέτοια λειτουργία να υλοποιηθεί και με τη γλώσσα περιγραφής υλικού VHDL. Συγκεκριμένα σε κάθε ενότητα θα γίνει η επέκταση των επερωτήσεων Q1, Q2, Q3 και Q4 στην αντίστοιχη VHDL.

Όπως αναφέρθηκε και στην Ενότητα 2.1 η γλώσσα VHDL έχει μια ποικιλία από τρόπους διευθέτησης των προβλημάτων. Όπως σε κάθε γλώσσα προγραμματισμού έτσι και στη VHDL η

λύση ενός προβλήματος έχει πολλές παραμέτρους. Η σχεδίαση και η ανάλυση κάθε προβλήματος εξαρτάται τόσο από την εμπειρία του σχεδιαστή (προγραμματιστή) όσο και από την ικανότητα του προγράμματος σχεδίασης CAD, αλλά και την αξιοπιστία των υλικών (FPGA, CPLD, PLD κτλ). Με άλλα λόγια οι ενδεικτικές λύσεις που προτείνονται παρακάτω ως επέκταση των επερωτήσεων Q1, Q2, Q3 και Q4, δεν είναι μοναδικές. Ως εκ τούτου μπορεί να υπάρχουν καλύτερες σχεδιαστικές λύσεις, από τις προτεινόμενες, με καλύτερα αποτελέσματα. Η προσπάθεια που υλοποιήθηκε (και θα συνεχίζει να υλοποιείται) είναι περισσότερο ενδεικτική, τόσο να αναδείξει όσο και να μυήσει τον αναγνώστη στη συγκεκριμένη τεχνολογία.

Μια ακόμη πτυχή που πρέπει να αναφερθεί, είναι ο χρόνος διευθέτησης των ερωτημάτων, δηλαδή ο *χρόνος εκτέλεσης* (*latency*) που απαιτείται για την επεξεργασία των δεδομένων από τη στιγμή που επιλέγονται και μπαίνουν στην επεξεργαστική μηχανή (από την πιο απλή μέχρι την πιο πολύπλοκη) μέχρι την έξοδο από αυτήν. Αν μια επεξεργαστική μηχανή έχει 3 υποκυκλώματα και το κάθε υποκύκλωμα έχει *latency* = 1, τότε η συνολική καθυστέρηση του *latency* είναι ίσο με 3. Στα παραδείγματα που θα ακολουθήσουν ο χρόνος αυτός αναφέρεται και αποτελεί μέτρο συγκρίσεις με τη συμβατική εκτέλεση των ερωτημάτων από έναν κοινό Η/Υ.

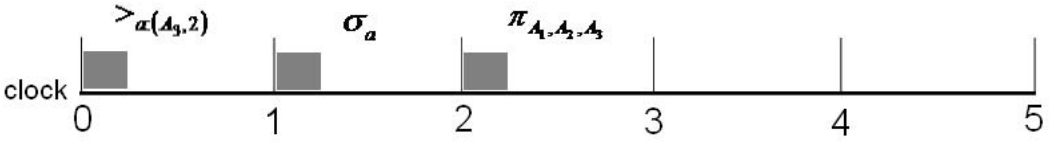
Ένα δεύτερο χρονικό μέτρο είναι το *ζητούμενο ποσοστό* (*issue rate*) το οποίο καθορίζει την μέγιστη καθυστέρηση που μπορεί να έχει μια επεξεργαστική μηχανή ή ένα υποκύκλωμα. Αν σε μια επεξεργαστική μηχανή το κάθε υποκύκλωμα του έχει issue rate =1, τότε η συνολική καθυστέρηση του issue rate είναι ίσο με 1. Δηλαδή ο χρόνος αυτός είναι άμεσα συνδεδεμένος με τον μέγιστο που υπάρχει σε κάθε υποκύκλωμα. Αυτή το δεύτερο χρονικό μέτρο επηρεάζει περισσότερο τα μεγάλα κυκλώματα όπου οι λογικές μονάδες έχουν καθυστερήσεις (θελημένες και μη) για τον καλύτερο συγχρονισμό αυτών. Στα κυκλώματα που έχουν υλοποιηθεί παρακάτω, αυτό το μέτρο είναι πολύ μικρό και δεν επηρεάζει άμεσα την ταχύτητα επεξεργασίας.

Μια ακόμη επισήμανση που πρέπει να ειπωθεί σε αυτό το σημείο είναι ότι επειδή η γλώσσα VHDL είναι γλώσσα περιγραφής υλικού όλοι οι αριθμοί και τα γράμματα πρέπει να κωδικοποιούνται σε δυαδική μορφή. Από αυτό το σημείο και παρακάτω κάθε φορά που θα αναφέρονται αριθμοί και γράμματα αυτό θα γίνεται σε δεκαδική και κανονική μορφή, αλλά στους κώδικες που θα παρουσιάζονται θα είναι σε δυαδική μορφή. Επίσης, όταν σχεδιάζεται (προγραμματίζεται) μία επερώτηση ο σχεδιαστής (προγραμματιστής) θα πρέπει να βάζει χειροκίνητα, μέσα στο πρόγραμμα, όλες τις τιμές που θα κάνουν τους ελέγχους.

3.3.1 Επέκταση Επερώτησης Q1 στη VHDL

Χρονική Εκτέλεση Επερωτήσεων

Η χρονική ακολουθία που επιτελεί η επερώτηση Q1 σε ένα ΣΒΔ, ανεξαρτήτου ταχύτητας επεξεργαστή, είναι της μορφής που φαίνεται στην Εικόνα 3.18. Και αυτό γιατί το ΣΣΔΒΔ πρέπει να διευθετήσει την επερώτηση έτσι ώστε υπάρχει συνέπεια δεδομένων και σύνταξη επερώτησης.



Εικόνα 3.18: Συμβατική χρονική ακολουθία επερώτησης Q1 από ένα ΣΒΔ, ανεξαρτήτου επεξεργαστικής ταχύτητας.

Όπως φαίνεται απαιτούνται τρεις κύκλοι ρολογιού για τη διευθέτηση της επερώτησης Q1 από το σύστημα. Αυτοί οι χρόνοι ελαχιστοποιούνται σε έναν κύκλο ρολογιού στη σχεδίαση με VHDL μιας και όπως έχει αναφερθεί η συγκεκριμένη γλώσσα περιγράφει υλικό. Η αλληλουχία των πράξεων δεν χαλάει μιας και υπάρχει συνέπεια τόσο στην εκτέλεση όσο και στην ροή των δεδομένων. Για να μπορέσει ο αναγνώστης να καταλάβει αυτή την ιδιαιτερότητα αρκεί να σκεφτεί ότι οι χρόνοι που τρέχουν στα ηλεκτρονικά κυκλώματα κυμαίνονται σε ελάχιστα ns (nanosecond, νανοδευτερόλεπτα) όπου 1 ns = 10^{-9} second. Οπότε η διευθέτηση συγκριτικών τελεστών και ιδιοτήτων, γίνεται πολύ γρήγορα μιας και σχεδιάζεται κύκλωμα για αποκλειστική αρχιτεκτονική, όπου αυτή η αρχιτεκτονική θα δώσει λύση σε ένα συγκεκριμένο πρόβλημα. Αποτέλεσμα αυτής της ιδιαιτερότητας είναι η επερώτηση Q1 να διευθετείται στη VHDL (και στο υλικό του FPGA) σε έναν κύκλο ρολογιού όπως φαίνεται στην Εικόνα 3.19. Η διαχείριση των δεδομένων μέσα στο κύκλωμα γίνεται ασύγχρονα και όταν εξαχθούν ως λύση του προβλήματος, συγχρονίζονται (γραμμή 60 του κώδικα της Εικόνας 3.20).

Εικόνα 3.19: Χρονική ακολουθία επερώτησης Q1 μετά από σχεδίαση στη γλώσσα VHDL και εκτέλεση του στη συσκευή FPGA

Σημείωση: Στις δύο εικόνες (Εικόνα 3.18 και 3.19) δεν φαίνεται η επιλογή της σχέσης στην οποία θα γίνει η επερώτηση. Να αναφερθεί ότι μια επιλογή ενός πίνακα (μιας σχέσης) μπορεί να γίνει με ένα απλό **if then else**. Στο συγκεκριμένο παράδειγμα όμως και στα άλλα που θα ακολουθήσουν, θεωρείται ότι η συσκευή FPGA λαμβάνει τα δεδομένα του πίνακα από μια εξωτερική σύνδεση. Οπότε είναι δεδομένη η επιλογή του συγκεκριμένου πίνακα.. Θα μπορούσε μέσω της χρήσης επιπλέουν κώδικα να διευθετηθεί και αυτή η παράμετρος (τις επιλογής του πίνακα), αλλά αυτό ξέφευγε από το πλαίσιο της έρευνας της μεταπτυχιακής διατριβής. Σε αυτήν την περίπτωση θα πρέπει να υλοποιούταν επιλογή μέσα από *ροή δεδομένων* (*dada stream*) και αυτό αποτελεί μελέτη - ερευνά της αναφοράς, *«Streams on Wires - A Query Compiler for FPGAs»* [20]. Στις επόμενες επερωτήσεις που περιλαμβάνουν δυο πίνακες (σχέσεις) έχει υλοποιηθεί μια λειτουργία που πρέπει να κάνει η συσκευή FPGA πριν εκτελέσει τις επερωτήσεις, περισσότερη λεπτομέρεια στη συνέχεια.

Λογική Μονάδα Q1

Ο κώδικας που υλοποιεί την επερώτηση Q1 φαίνεται στην Εικόνα 3.20. Στις γραμμές 30 – 39 γίνεται η δήλωση της οντότητα **Q1_select** και η δήλωση των μεταβλητών εισόδου εξόδου. Στις γραμμές 44 - 47 γίνεται η δήλωση των εσωτερικών σημάτων ελέγχου. Και τέλος στις γραμμές 49 – 71 υλοποιείται η λύσης του προβλήματος. Όπως μπορεί να παρατηρήσει ο αναγνώστης ο έλεγχος που γίνεται στη γραμμή 61 είναι του τελεστή «**>**» και συγκρίνει την τιμή του γνωρίσματος **A3** να είναι μεγαλύτερο της τιμής '**2**'. Σε αυτό το σημείο να διευκρινιστεί ότι ο σχεδιαστής αν για κάποιο λόγο χρειαστεί να αλλάξει τη <*συνθήκη*> του Τύπου 3.14, δε έχει από το να πάει στη γραμμή 61 και να τροποποιήσει τον τελεστή ή και το γνώρισμα. Αυτή ακριβώς είναι η ευχρηστία που μπορεί να προσφέρει η σχεδίαση επερωτήσεων με τη γλώσσα VHDL.

Η εισαγωγή των πλειάδων γίνεται σε 24bit μορφή (γραμμή 35 μεταβλητή **in_data**) όπου κάθε γνώρισμα κωδικοποιείται σε μορφή των 8bit. Ο διαχωρισμός τους γίνεται με τη διοχέτευση των κατάλληλων bit στη σωστή θέση. Συγκεκριμένα στη γραμμή 53 έχουμε την εισαγωγή του πρώτου γνωρίσματος A_1 του πίνακα R στην μεταβλητή **A1**, στέλνονται τα πρώτα 8 bit (θέσεις 23… 16). Στη γραμμή 54 έχουμε την εισαγωγή του δεύτερου γνωρίσματος A_2 του πίνακα R, στην μεταβλητή **A2** (θέσεις 15… 8). Τέλος στη γραμμή 55 έχουμε την εισαγωγή του τρίτου και τελευταίου γνωρίσματος A_3 του πίνακα R, στην μεταβλητή **A3** (θέσεις 7… 0).

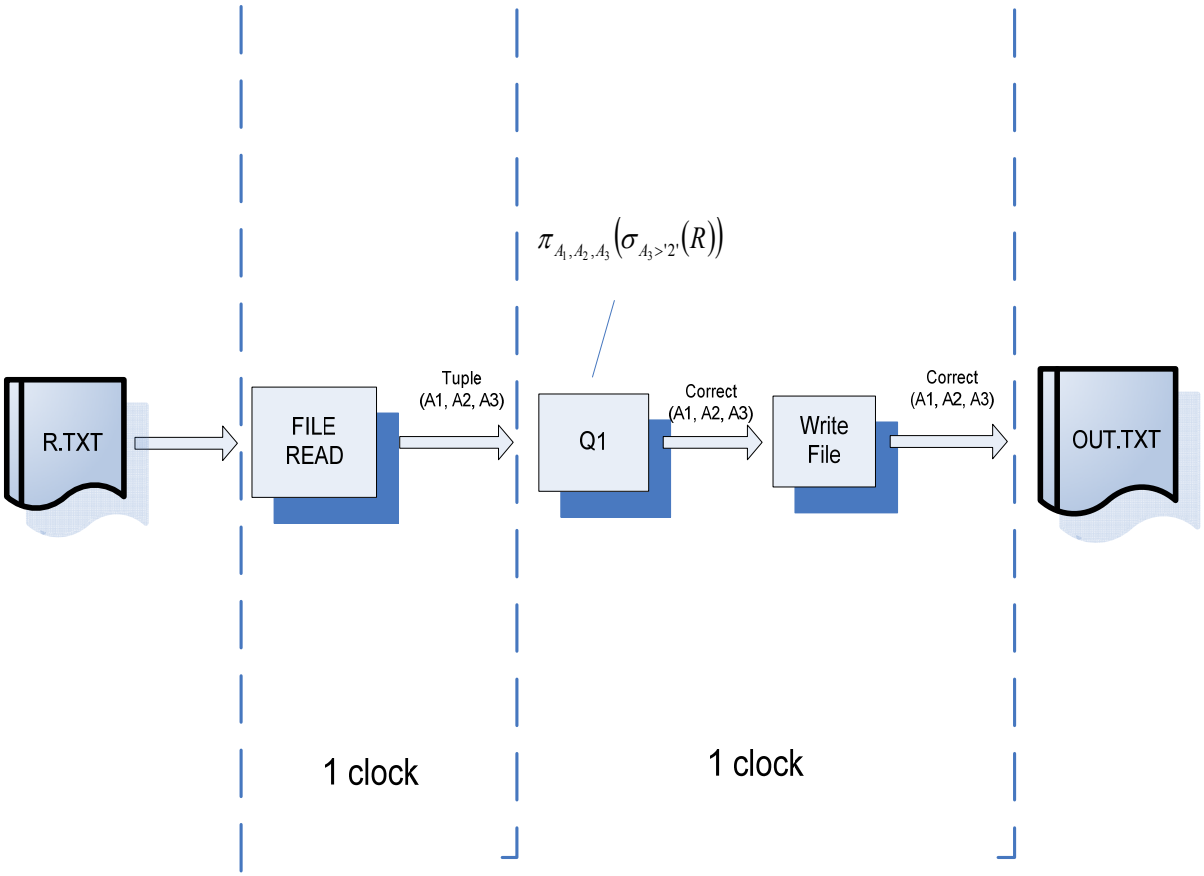
```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity Q1_select is
31
32  Port (
33          clk         : in   std_logic;                          -- clock input
34          rst         : in   std_logic;                          -- reset input
35          in_data     : in   STD_LOGIC_VECTOR (23 downto 0);     -- data input
36          valid_bit   : out   STD_LOGIC;                         -- valid bit output
37          out_data    : out   STD_LOGIC_VECTOR (23 downto 0)     -- data output
38          );
39  end Q1_select;
40
41  architecture Behavioral of Q1_select is
42
43  -- control signals
44  signal A1      :   std_logic_vector(7 downto 0);      -- the first element A1 of the tuple
45  signal A2      :   std_logic_vector(7 downto 0);      -- the second element A2 of the tuple
46  signal A3      :   std_logic_vector(7 downto 0);      -- the third element A3 of the tuple
47  signal s_in_data  :   STD_LOGIC_VECTOR (23 downto 0); -- the signal from input data
48
49  begin
50
51  -- signals  initialization
52  s_in_data <= in_data;
53  A1 <= s_in_data (23 downto 16);      -- initialize first element A1 of the tuple
54  A2 <= s_in_data (15 downto  8);      -- initialize second element A2 of the tuple
55  A3 <= s_in_data ( 7 downto  0);      -- initialize third element A3 of the tuple
56
57  ------------------------------------------- Q1 ------------------------------------------------
58  Q1: process (clk)                    -- start process Q1
59     begin
60     if clk'event and clk='1' then     -- synchronization whit clock
61        if (A3 > "00000010") then      -- the control element A3 > 2
62           out_data <= s_in_data;      -- output correct data
63           valid_bit <= '1' ;          -- valid bit = '1'
64        else
65           out_data <= (others =>'0'); -- output null data
66           valid_bit <= '0' ;          -- valid bit = '0'
67           end if;
68     end if;
69  end process Q1;                      --end process Q1
70
71  end Behavioral;
```

Εικόνα 3.20: Κώδικας VHDL που υλοποιεί την επερώτηση Q1 του Τύπου 3.14

Επιπλέον, αν χρειαστεί για κάποιο λόγο να δοθεί ως έξοδος το ένα μόνο στοιχείο από μια πλειάδα, τότε ο σχεδιαστής θα πρέπει να τροποποιήσει τις μεταβλητές εξόδου αλλά και τα σήματα ελέγχου. Αυτή η διαδικασία απαιτεί μεγαλύτερη σχεδιαστική εμπειρία μιας και αυτή η λογική μονάδα αποτελεί υποκύκλωμα μιας άλλης μεγαλύτερης. Μια τελευταία επισήμανση είναι η μορφής των τιμών εισόδου, όπως μπορεί να διακρίνει ο αναγνώστης οι τιμές είναι σε δυαδική μορφή γιατί όπως προαναφέρθηκε η όλη σχεδίαση έχει υλοποιηθεί σε μορφή υλικού.

Λογική Αρχιτεκτονική Δομή Επερώτησης Q1

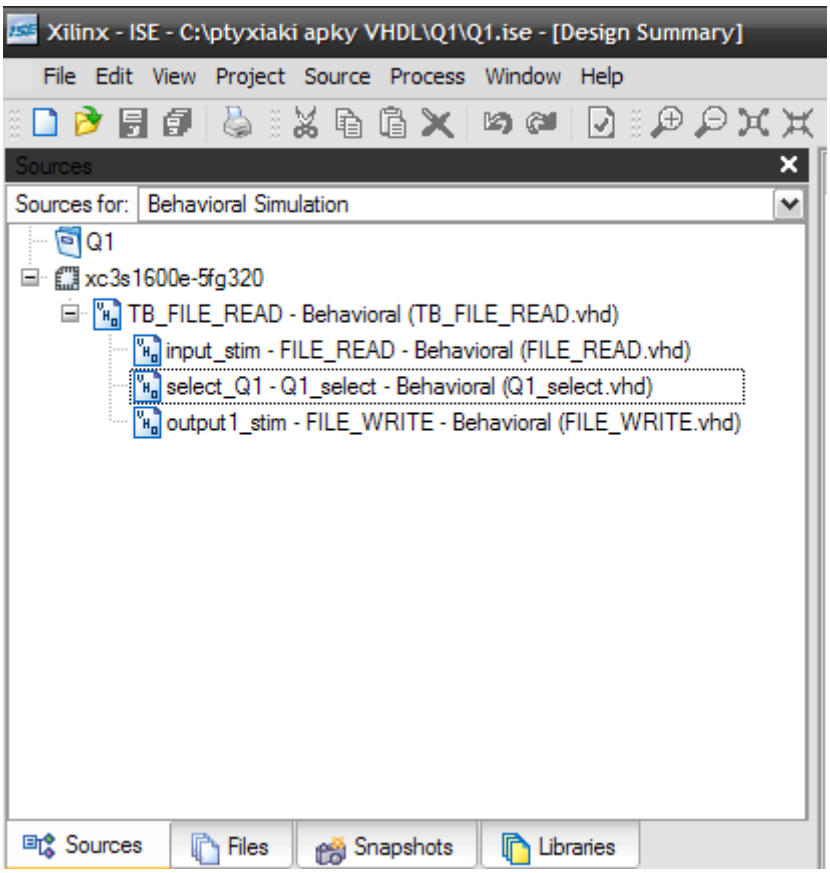
Στην Εικόνα 3.21 φαίνεται η αρχιτεκτονική δομή της διαδικασία που υλοποιεί τη σχεδίαση επίλυσης της επερώτησης Q1. Ο *χρόνος εκτέλεσης* (*latency*) της επερώτησης από τη στιγμή εισόδου των δεδομένων μέχρι τη στιγμή εξόδου τους ισούται με 2 (2 κύκλοι ρολογιού, latency = 2) και το *ζητούμενο ποσοστό* (*issue rate*) ισούται με 1. Είναι 1 κύκλος ρολογιού για κάθε λογική διεργασία άρα issue rate = 1, άρα το συνολικό ζητούμενο ποσοστό (issue rate) του κυκλώματος ισούται με 1, (γιατί παίρνουμε το μέγιστο issue rate κάθε λογικής μονάδας).



Εικόνα 3.21: Σχεδιαστική πρόταση που υλοποιεί την επίλυση της επερώτησης Q1

Σχεδίαση ISE

Οι λογικές μονάδες που χρησιμοποιήθηκαν κατά την επίλυση φαίνονται στην Εικόνα 3.22 και έχουν παρθεί από το παράθυρο *Project Navigator* (Εικόνα Δ.2 Παράρτημα Δ) του προγράμματος σχεδίασης CAD ISE.



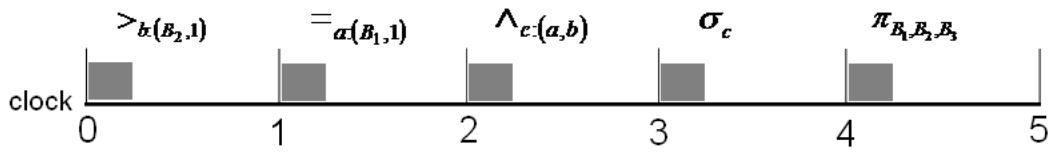
Εικόνα 3.22: Οι λογικές μονάδες της σχεδίασης της λύσης της επερώτησης Q1 από το παράθυρο Project Navigator του προγράμματος σχεδίασης CAD ISE

Στο Κεφάλαιο 5 ο αναγνώστης θα έχει την ευκαιρία να δει και σε γραφική μορφή τα αποτελέσματα αυτής της εκτέλεσης όπως και τα παραγόμενα δεδομένα της επερώτησης Q1.

3.3.2 Επέκταση Επερώτησης Q2 στη VHDL

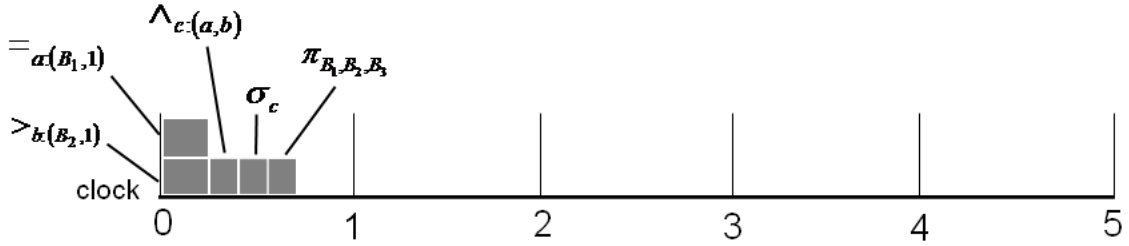
Χρονική Εκτέλεση Επερωτήσεων

Η χρονική ακολουθία που επιτελεί η επερώτηση Q2 σε ένα ΣΒΔ είναι της μορφής που φαίνεται στην Εικόνα 3.23. Όπως φαίνεται απαιτούνται πέντε κύκλοι ρολογιού για τη διευθέτηση της επερώτησης Q2 από το σύστημα.



Εικόνα 3.23: Συμβατική χρονική ακολουθία επερώτησης Q2 από ένα ΣΒΔ, ανεξαρτήτου επεξεργαστικής ταχύτητας.

Αυτοί οι χρόνοι ελαχιστοποιούνται σε <u>έναν</u> κύκλο ρολογιού στη σχεδίαση με VHDL (Εικόνα 3.24). Όπως μπορεί να παρατηρηθεί οι συγκρίσεις των τελεστών «$>$» και «$=$» εκτελούνται παράλληλα και αμέσως μετά θα μεταβούν στο λογικό τελεστή AND «$\wedge$». Σε κάθε περίπτωση, ο σχεδιαστής θα πρέπει να είναι πολύ προσεχτικός με τη σειρά των πράξεων, αυτό δεν είναι καθόλου εύκολο όταν οι πράξεις είναι πολύπλοκες και έχουν πολλά κρατούμενα.. Πρέπει να προβλέπει αυτές τις ιδιαιτερότητες και να χρησιμοποιεί γεννήτριες καθυστέρησης, ώστε τα δεδομένα προς επεξεργασία να φτάνουν στην κατάλληλη χρονική στιγμή. Στη συγκεκριμένη περίπτωση οι πράξης και τα δεδομένα ήταν απλοϊκά και δεν υπήρχαν καθυστερήσεις στα αποτελέσματα.



Εικόνα 3.24: Χρονική ακολουθία επερώτησης Q2 μετά από σχεδίαση στη γλώσσα VHDL και εκτέλεση του στη συσκευή FPGA

Λογική Μονάδα Q2

Ο κώδικας που υλοποιεί την επερώτηση Q2 φαίνεται στην Εικόνα 3.25. Σε αυτόν τον κώδικα ο αναγνώστης μπορεί να διακρίνει τις τρεις διεργασίες **Q2_A**, **Q2_B** και **Q2_C** όπου και οι τρεις εκτελούνται παράλληλα. Αυτό όμως που τα συγχρονίζει μεταξύ τους είναι το ρολόι αλλά και οι μεταβλητές **valid_bit_A** και **valid_bit_B**, όπου και τα τρία μαζί παράγουν την έξοδο της ζητούμενης πλειάδας (γραμμές 91 … 97).

Η διεργασία **Q2_A** υλοποιεί τη σύγκριση $=_{a:(B_1,1)}$, η διεργασία **Q2_B** υλοποιεί τη σύγκριση $>_{b:(B_2,1)}$ και τέλος η διεργασία **Q2_C** υλοποιεί το λογικό τελεστή $\wedge_{c:(a,b)}$. Μέσα σε έναν κύκλο ρολογιού και η τρεις διεργασίες εκτελούνται ταυτόχρονα και παράγουν το επιθυμητό αποτέλεσμα. Αυτό που συγχρονίζει τη σωστή τιμή εξόδου, πέραν των μεταβλητών valid_bit_A και valid_bit_B όπως προαναφέρθηκε, είναι και η καταχώρηση της μεταβλητής (σήματος) **s_in_data_B** στις γραμμές 67 και 80. Σε περίπτωση που δεχόταν την τιμή απευθείας από τη μεταβλητή **in_data και s_in_data,** τότε ως τιμή εξόδου θα δινόταν η τρέχουσα και όχι αυτή που

ελέγχθηκε με τις διεργασίες Q2_A και Q2_B. Όπως αναφέρθηκε (ίσως γίνεται κουραστικό αυτό αλλά είναι πολύ σημαντικό και πρέπει να το καταλάβει ο αναγνώστης), οι ταχύτητες που υπάρχουν είναι πολύ γρήγορες και για αυτό πρέπει να υπάρχει συντονισμός και συγχρονισμός των δεδομένων. Αυτή η σχεδίαση αποτελεί μια λύση, αλλά όχι τη μοναδική.

Να σημειωθεί εδώ ότι σε περίπτωση που χρειαζόταν και άλλες διεργασίες να προστεθούν, τέτοιες που να υλοποιούσαν συγκρίσεις, τότε θα μπορούσαν να χρησιμοποιηθούν όπως οι προαναφερόμενοι με τα ίδια αποτελέσματα. Στην έρευνα που έγινε δεν υλοποιήθηκε (ακόμη) επερώτηση με περισσότερους από δυο τελεστές σύγκρισης. Αν και από τα αποτελέσματα που έχουν εξαχθεί, φαίνεται ότι κάτι τέτοιο μπορεί να υλοποιηθεί. Ίσως το μόνο που χρήζει προσοχής είναι ο συγχρονισμός των αποτελεσμάτων να είναι ο σωστός.

```
54  begin
55  -- signals  initialization
56  s_in_data <= in_data;
57  B1 <= s_in_data (23 downto 16);      -- initialize first element B1 of the tuple
58  B2 <= s_in_data (15 downto  8);      -- initialize second element B2 of the tuple
59  B3 <= s_in_data ( 7 downto  0);      -- initialize third element B3 of the tuple
60
61  ------------------------------------------------ Q2_A ------------------------------------------------
62  Q2_A: process (clk)                  -- start process Q2_A
63     begin
64     if clk'event and clk='1' then     -- synchronization whit clock
65        if (B1 = "00000001") then      -- the control element B1 = '1'
66           valid_bit_A <= '1' ;        -- valid bit A = '1'
67           s_in_data_B <= in_data;     -- correct data
68        else
69           valid_bit_A <= '0' ;        -- valid bit A = '0'
70           end if;
71     end if;
72  end process Q2_A;                    -- end process Q2_A
73
74  ------------------------------------------------ Q2_B ------------------------------------------------
75  Q2_B: process (clk)                  -- start process Q2_B
76     begin
77     if clk'event and clk='1' then     -- synchronization whit clock
78        if (B2 > "00000001") then      -- the control element B2 > 1
79           valid_bit_B <= '1' ;        -- valid bit B = '1'
80           s_in_data_B <= in_data;     -- correct data
81        else
82           valid_bit_B <= '0' ;        -- valid bit B = '0'
83           end if;
84     end if;
85  end process Q2_B;                    -- end process Q2_B
86
87  ------------------------------------------------ Q2_C ------------------------------------------------
88  Q2_C:process (clk,valid_bit_A,valid_bit_B)   -- start process Q2_C
89     begin
90     -- if clk'event and clk='1' then          -- synchronization whit clock
91           if (valid_bit_A = '1' and valid_bit_B = '1') then -- if the comparisons are correct
92              out_data <= s_in_data_B;     -- output correct data
93              valid_bit <= '1' ;           -- valid bit = '1'
94           else
95              out_data <= (others =>'0');     -- output null data
96              valid_bit <= '0' ;           -- valid bit = '0'
97           end if;
98     -- end if;
99  end process Q2_C;                        -- end process Q2_C
100
101 end Behavioral;
```

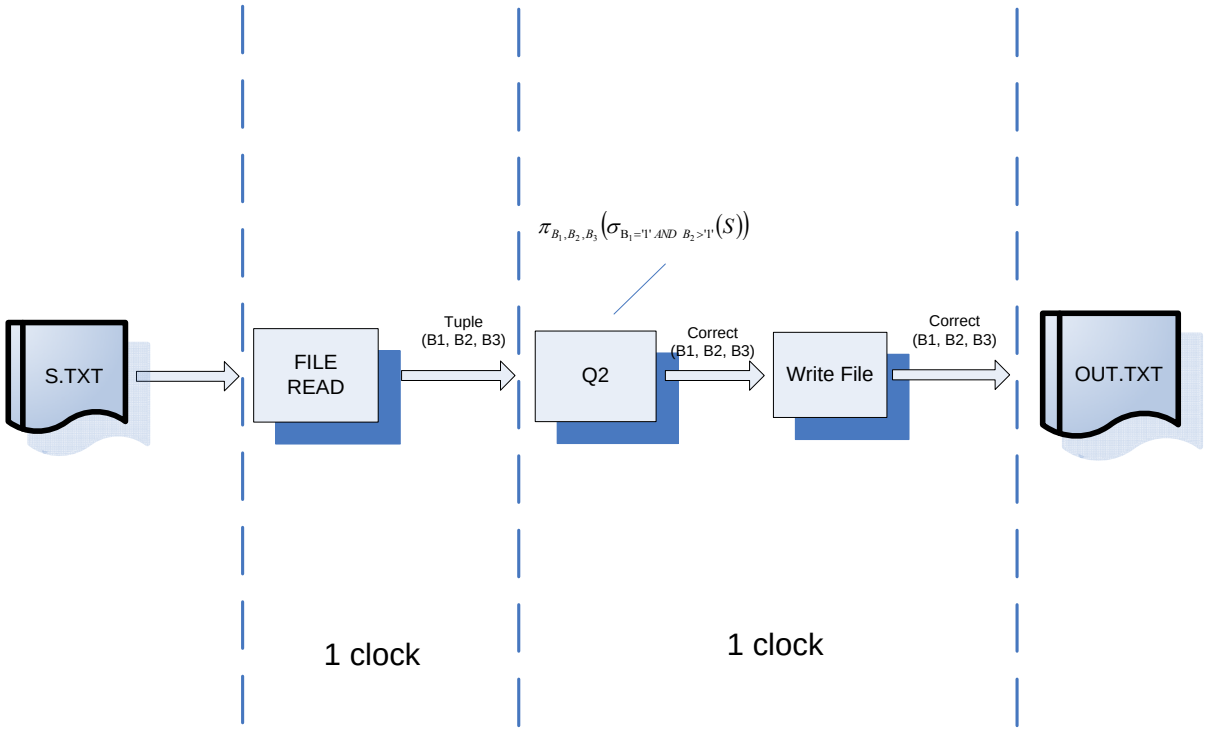
Εικόνα 3.25: Κώδικας VHDL που υλοποιεί την επερώτηση Q2 του Τύπου 3.16

Παρατήρηση: Στην Εικόνα 3.25 στις γραμμές 90 και 98 φαίνεται ότι ο κώδικας "**if clk'event and clk='1' then**" και "**end if;** " έχει απενεργοποιηθεί. Ο λόγος είναι ότι αυτό το κομμάτι κώδικα αν ενεργοποιηθεί θα δώσει μια καθυστέρηση της τάξεως ενός κύκλου ρολογιού. Δηλαδή για να εκτελεστεί η επερώτηση Q2 θα χρειάζονταν δύο κύκλους ρολογιού αντί για έναν. Η συγκεκριμένη παρέμβαση είναι εφικτή όταν η αξιοπιστία του αποτελέσματος δεν επηρεάζεται, σε αντίθετη περίπτωση δεν επιτρέπεται. Γενικά ο σχεδιαστής μπορεί να κρίνει πότε πρέπει να κάνει τέτοιες παρεμβάσεις, ώστε να πετυχαίνει πάντα το κύκλωμα βέλτιστες επιδώσεις. Στη συγκεκριμένη περίπτωση η παρέμβαση αυτή δεν αλλοιώνει το αποτέλεσμα, αντιθέτως μεγιστοποιεί την ταχύτητα εκτέλεσης.

Και σε αυτήν την περίπτωση η τροποποίηση των τελεστών σύγκρισης και των λογικών τελεστών γίνεται γρήγορα και εύκολα, για περαιτέρω αλλαγές χρειάζεται εμπειρία και προσοχή.

Λογική Αρχιτεκτονική Δομή Επερώτησης Q2

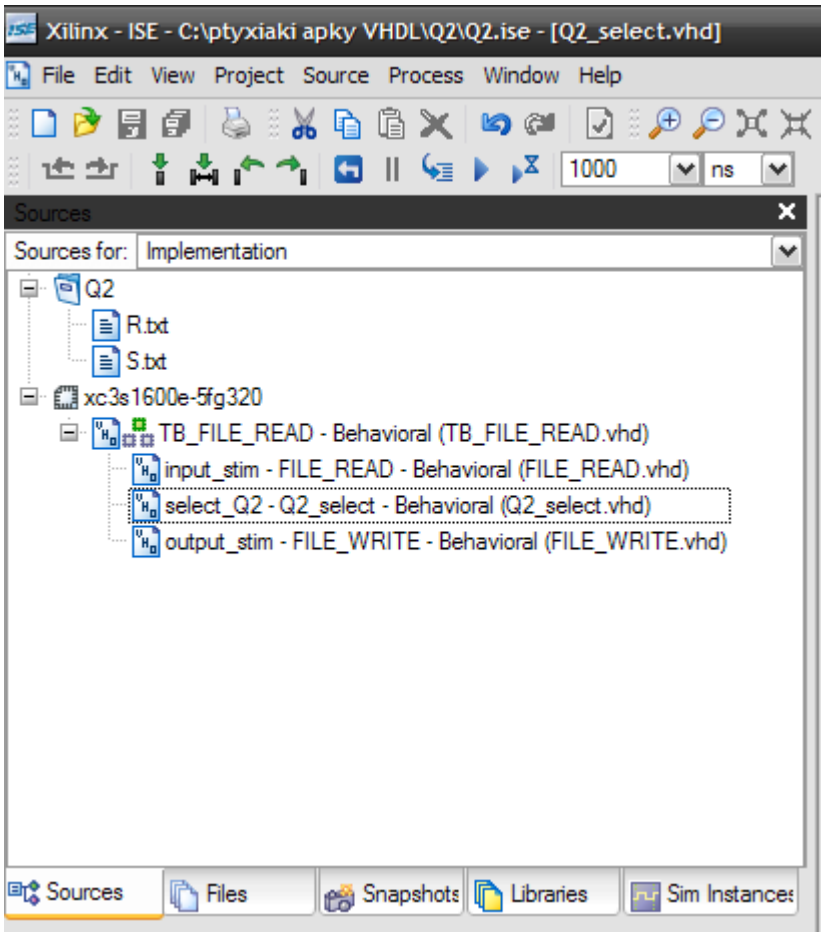
Στην Εικόνα 3.26 φαίνεται η αρχιτεκτονική δομή της διαδικασίας που υλοποιεί τη σχεδίαση επίλυσης της επερώτησης Q2. Ο *χρόνος εκτέλεσης* (*latency*) ισούται με 2 (2 κύκλοι ρολογιού, latency = 2) και το *ζητούμενο ποσοστό* (*issue rate*) ισούται με 1. Είναι 1 κύκλος ρολογιού για κάθε λογική μονάδα διεργασίας άρα issue rate = 1, άρα το συνολικό ζητούμενο ποσοστό (issue rate) του κυκλώματος ισούται με 1.



Εικόνα 3.26: Σχεδιαστική πρόταση που υλοποιεί την επίλυση της επερώτησης Q2

Σχεδίαση ISE

Οι λογικές μονάδες που χρησιμοποιήθηκαν κατά την επίλυση φαίνονται στην Εικόνα 3.27 και έχουν παρθεί από το παράθυρο *Project Navigator* (Εικόνα Δ.2 Παράρτημα Δ) του προγράμματος σχεδίασης CAD ISE.



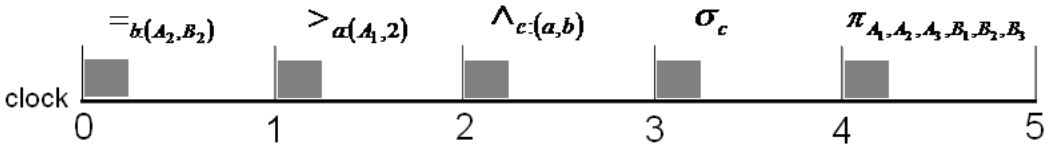
Εικόνα 3.27: Οι λογικές μονάδες της σχεδίασης της λύσης της επερώτησης Q2 από το παράθυρο Project Navigator του προγράμματος σχεδίασης CAD ISE

Στο Κεφάλαιο 5 ο αναγνώστης θα έχει την ευκαιρία να δει και σε γραφική μορφή τα αποτελέσματα αυτής της εκτέλεσης όπως και τα παραγόμενα δεδομένα της επερώτησης Q2.

3.3.3 Επέκταση Επερώτησης Q3 στη VHDL

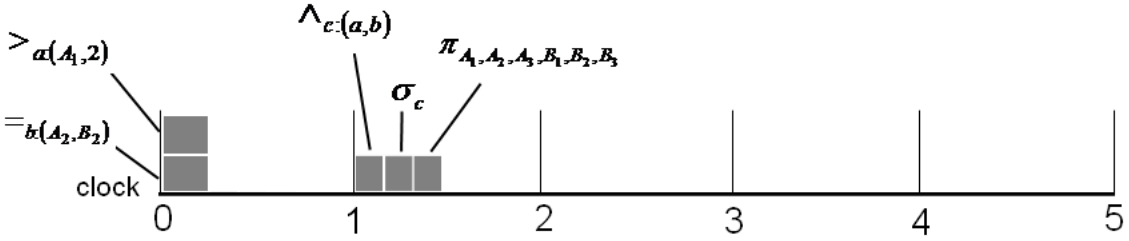
Χρονική Εκτέλεση Επερωτήσεων

Η χρονική ακολουθία που επιτελεί η επερώτηση Q3 σε ένα ΣΒΔ είναι της μορφής που φαίνεται στην Εικόνα 3.28. Όπως φαίνεται απαιτούνται πέντε κύκλοι ρολογιού για τη διευθέτηση της επερώτησης Q3 από το σύστημα.



Εικόνα 3.28: Συμβατική χρονική ακολουθία επερώτησης Q3 από ένα ΣΒΔ ανεξαρτήτου επεξεργαστικής ταχύτητας.

Αυτοί οι χρόνοι ελαχιστοποιούνται σε δυο κύκλος ρολογιού στη σχεδίαση με VHDL (Εικόνα 3.29). Όπως μπορεί να παρατηρηθεί οι συγκρίσεις των τελεστών «=» και «>» εκτελούνται παράλληλα και στον επόμενο παλμό θα μεταβούν στο λογικό τελεστή AND « ∧ », κατόπιν αυτού θα εξαχθούν τα ζητούμενα δεδομένα. Η χρονική ακολουθία μπορεί να μοιάζει με αυτή της επερώτησης Q2 αλλά εδώ η ανάμειξη της πράξης της συνένωσης σε συνδυασμό με την καθυστέρηση διάδοσης των δεδομένων, δεν επέτρεπε την καλύτερη χρονική βελτιστοποίηση της.

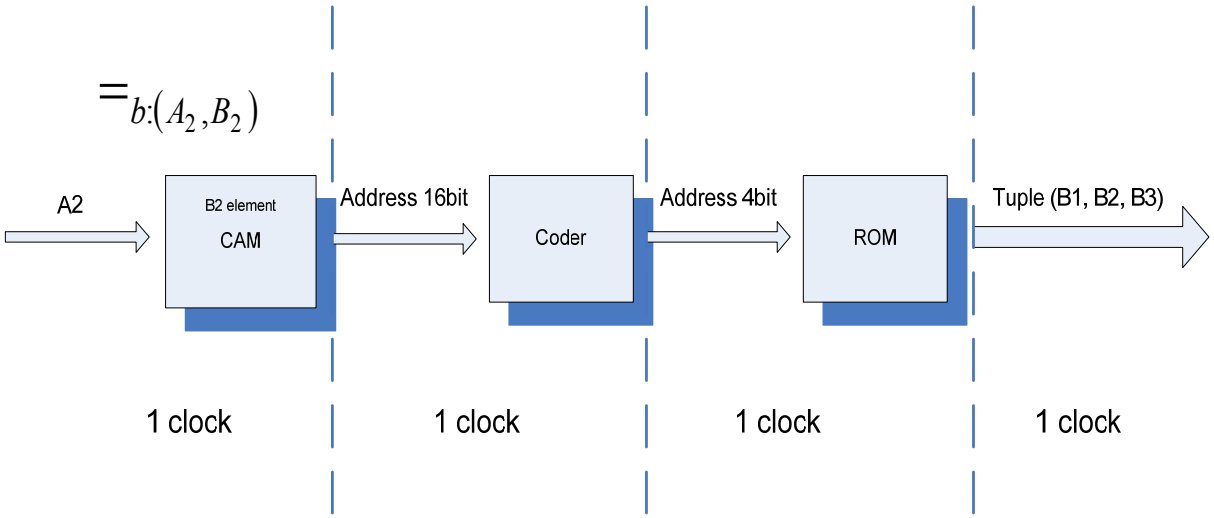


Εικόνα 3.29: Χρονική ακολουθία επερώτησης Q3 μετά από σχεδίαση στη γλώσσα VHDL και εκτέλεση του στη συσκευή FPGA

Μνήμες CAM, RAM και ROM

Για να απαντηθεί αυτή η επερώτηση με τη γλώσσα VHDL χρησιμοποιώντας τη συσκευή FPGA, θα πρέπει πρώτα να γίνει μια προεργασία. Θα πρέπει να καθορίσει ο σχεδιαστής ποια από τις δυο σχέσεις που εμπλέκονται στην επερώτηση πρέπει να "φορτωθεί" στη συσκευή. Δηλαδή για να μπορέσει η συσκευή να διευθετήσει το ζητούμενο πρέπει πρώτα να αποθηκευθεί ο ένας από τους δύο πίνακες στις μνήμες CAM και RAM (ή ROM) της συσκευής FPGA. Ο λόγος που χρειάζονται και οι δύο μνήμες είναι ότι αν χρησιμοποιούταν μόνο η μνήμη RAM - ROM τότε για να βρεθούν τα δεδομένα θα έπρεπε να γίνει προσπέλαση της μνήμης θέση θέση. Αυτό σημαίνει ότι στη χειρότερη περίπτωση θα χρειαζόμασταν **n** κύκλους ρολογιού για την εύρεση του ζητούμενου στοιχείου. Με τη χρήση όμως και της μνήμης CAM αυτός ο χρόνος περιορίζεται σε **τρεις** κύκλους ρολογιού **πάντα**! Η λειτουργία της μνήμης CAM είναι ότι, ως στοιχείο αναζήτησης

είναι τα ίδια τα δεδομένα, δηλαδή στέλνονται τα προς εύρεση δεδομένα στη μνήμη CAM και επιστρέφεται η διεύθυνση στην οποία βρίσκονται. Κατόπιν αυτού η διεύθυνση κωδικοποιείται και μέσω ενός διαύλου πάει στη μνήμη RAM - ROM και βρίσκει την πλειάδα που χρειάζεται. Η λογική λειτουργία που πρέπει να υλοποιηθεί είναι αυτή που φαίνεται στην Εικόνα 3.30.



Εικόνα 3.30: Διαδικασία εκμαιεύσεις δεδομένων από τη μνήμη της επερώτησης Q3 του Τύπου 3.18

Αυτή η εικόνα δείχνει τη διαδικασία που πρέπει να υλοποιηθεί για να εκτελεστεί η πράξη της ισότητας $=_{b:(A_2,B_2)}$ της επερώτησης Q3. Η σειρά εκτέλεσης είναι: το στοιχείο A2 στέλνεται στη μνήμη CAM για ταυτοποίηση με το στοιχείο B2, αν βρεθεί μέσα στην CAM τότε εξάγεται η διεύθυνση στην οποία βρίσκεται, η διεύθυνση αυτή (16bit) κωδικοποιείται στη μορφή που υποστηρίζεται η μνήμη RAM (ROM), δηλαδή 4bit. Τέλος μέσω της κωδικοποιημένης διεύθυνσης εξάγεται η επιθυμητή πλειάδα με τα γνωρίσματα της, κατόπιν αυτού τα αποτελέσματα θα πάνε σε μια άλλη διαδικασία για την περαιτέρω επεξεργασία τους. Υπάρχει η δυνατότητα απευθείας κωδικοποιημένης διεύθυνση, από τη μνήμη CAM, ώστε να απαλειφόταν ένας ακόμη κύκλος ρολογιού από την Εικόνα3.30, αλλά για κάποιο λόγο παρουσίαζε πρόβλημα. Σε κάθε περίπτωση, όπως προαναφέρθηκε, υπάρχει πληθώρα σχεδιαστικών προσεγγίσεων στη λύση ενός προβλήματος, η τελική λύση που υλοποιήθηκε για την επερώτηση Q3 προσεγγίζει αυτή της εικόνας Εικόνα3.30.

Αλγόριθμός CAM_ARRAY

Μια ακόμη επισήμανση για τη μνήμη CAM είναι ότι αν περιέχει περισσότερα από ένα ίδια στοιχεία μέσα στη μνήμη, τότε ο χρήστης έχει τη δυνατότητα να λάβει την πρώτη ή την τελευταία θέση της. Αν υπάρχουν περισσότερες από μία *επιτυχίες εύρεσης* (*hit*) μπορεί να

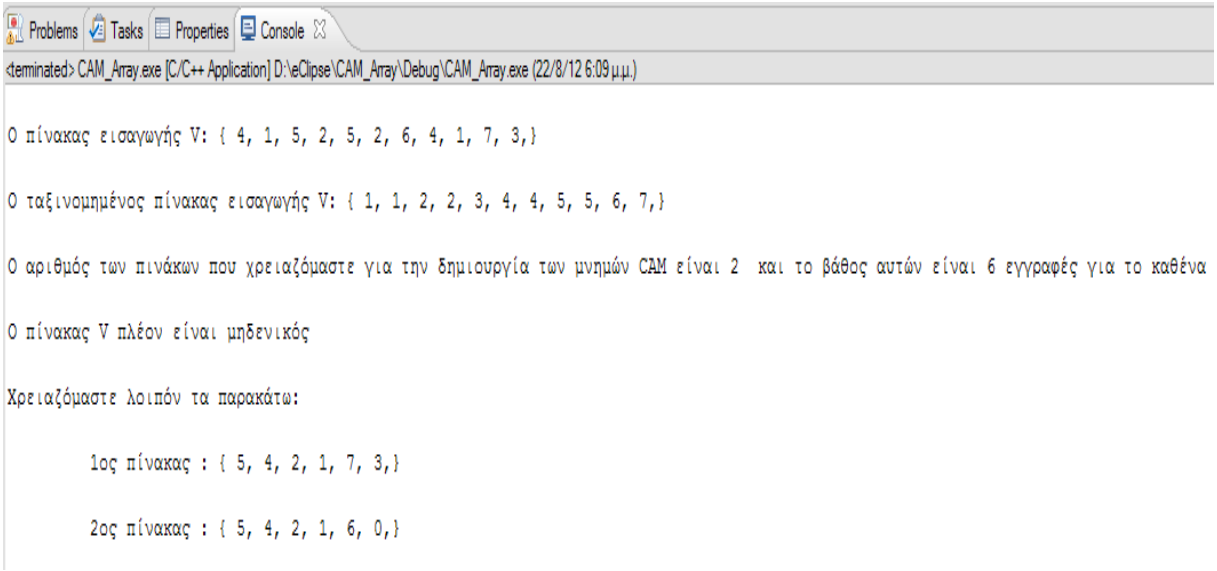
επιλέξει να εμφανίζονται όλες αυτές. Η εμφάνιση της επιτυχής εύρεσης (hit) έχει να κάνει με τον αριθμό των 1 που βρίσκονται στην παραγόμενη διεύθυνση (address 16bit). Αυτό όμως θα ήταν πολύ δύσχρηστο στη μετέπειτα εύρεση του στη μνήμη RAM (ROM). Η προσέγγιση που υλοποιήθηκε ήταν η δημιουργία διαφορετικών μνημών CAM και RAM - ROM όπου η κάθε μια θα περιέχει μοναδικά δεδομένα. Όπως προαναφέρθηκε, οι συσκευές FPGA έχουν τη δυνατότητα να κάνουν παράλληλη επεξεργασία δεδομένων, οπότε αυτήν την ιδιαιτερότητα εκμεταλλεύεται η παρούσα λύση. Για περισσότερη λεπτομέρεια γύρω από τις ιδιότητες των μνήμών CAM και RAM - ROM παρακαλείται ο αναγνώστης να μεταβεί στις Ενότητες 4.2.1 και 4.3.1.

Για να μπορέσει κάποιος όμως να διαχωρίσει το περιεχόμενο των μνημών CAM και RAM- ROM θα πρέπει να είναι ιδιαίτερα προσεχτικός με τα περιεχόμενα. Όταν τα προς διαχωρισμού περιεχόμενα είναι ελάχιστα, η διαδικασία μπορεί να γίνει εμπειρικά και χειρονακτικά, τι γίνεται όμως όταν περιέχουν εκατοντάδες (ίσως και χιλιάδες) στοιχεία; Την απάντηση την δίνει ο αλγόριθμός CAM_ARRAY (Ενότητα 4.2.3) ο οποίος διαχωρίζει τα περιεχόμενα ενός μονοδιάστατου πίνακα σε υποδεέστερους υποπίνακες, όπου ο καθένας από αυτούς έχει μοναδικά στοιχεία στα περιεχόμενα του. Συγκεκριμένα ελέγχει ποιο στοιχείο έχει τις περισσότερες εμφανίσεις μέσα στον «πατρικό» πίνακα, το καταγράφει και στο τέλος με βάση τον αριθμό των εμφανίσεων του, δημιουργεί δυναμικά τους αντίστοιχους υποπίνακες. Κατόπιν αυτού κάνει ξανά διαπέραση τον πατρικό πίνακα για να βρει το αμέσως επόμενο στοιχείο με τις περισσότερες εμφανίσεις, όταν τον βρει τον εισάγει και αυτόν στους δημιουργηθέντες πίνακες. Αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου ο πατρικός πίνακας να μηδενιστεί. Τα στοιχεία που πρέπει να εισαχθούν στη μνήμη CAM είναι αυτά για τα οποία θα γίνει ο έλεγχος της ισότητας, ουσιαστικά η χρήση της μνήμης CAM βοηθάει στη δημιουργία της Συνένωσης (JOIN) μεταξύ των εμπλεκομένων πινάκων. Για περισσότεροι λεπτομέρεια γύρω από τον αλγόριθμο CAM_ARRAY ο αναγνώστης καλείται να δει την Ενότητα 4.2.3.

Περιεχόμενα Μνήμων CAM και ROM

Στην περίπτωση της επερώτησης Q3 επιλέχθηκε στη μνήμη ROM να βρίσκεται ο πίνακας S (Εικόνα 3.4) και στη μνήμη CAM να βρίσκεται το γνώρισμα (στήλη) B_2. Όπως μπορεί κανείς να παρατηρήσει το γνώρισμα B_2 έχει περισσότερα από ένα ίδια στοιχεία, για να γίνει λοιπόν η υλοποίηση της επερώτησης Q3 σύμφωνα με όσα έχουν ειπωθεί, θα πρέπει να γίνει χρήση του αλγόριθμου CAM_ARRAY. Να σημειωθεί ότι στον αλγόριθμο δεν πρέπει να εισαχθεί ο αριθμός '0' στα στοιχεία του πίνακα εισαγωγής και αυτό γιατί ο αριθμός μηδέν αποτελεί το στοιχείο ελέγχου.

Το γνώρισμα B_2 έχει την ακολουθία των αριθμών {4,1,5,2,5,2,6,4,1,7,3}, το παραγόμενο αποτέλεσμα μετά την εκτέλεση του αλγόριθμου φαίνεται στην Εικόνα 3.31. Όπως μπορεί κανείς να παρατηρήσει ο αλγόριθμος κάνει πρώτα ταξινόμηση και κατόπιν υπολογίζει το πλήθος των μνημών CAM καθώς και τα περιεχόμενα αυτών.



Εικόνα 3.31: Εκτέλεση του αλγόριθμου CAM_ARRAY στο γνώρισμα B_2 του πίνακα S, της επερώτησης Q3

Από το αποτέλεσμα της εκτέλεσης του αλγόριθμου CAM_ARRAY στην ακολουθία αριθμών του γνωρίσματος B2 της σχέσης S, προκύπτουν ότι πρέπει να χρησιμοποιηθούν δύο μνήμες CAM και ως εκ τούτου δυο μνήμες ROM. Στην Εικόνα 3.32 φαίνεται ποιο στοιχείο θα πάει σε ποια μνήμη.

B_1	B_2	B_3	ROM
2	4	8	romA
1	1	1	romA
2	5	5	romA
1	2	1	romA
2	5	4	romB
1	2	3	romB
4	6	9	romB
8	4	2	romB
1	1	3	romB
3	7	2	romA
5	3	1	romA

B_2	CAM
4	camA
1	camA
5	camA
2	camA
5	camB
2	camB
6	camB
4	camB
1	camB
7	camA
3	camA

Εικόνα 3.32: Κατανομή στοιχείων πίνακα S στις αντίστοιχες μνήμες CAM και ROM, της επερώτησης Q3

Στην Εικόνα 3.33 φαίνονται τα περιεχόμενα των μνημών που θα χρησιμοποιηθούν για την επίλυση της επερώτησης Q3. Κάθε θέση μνήμης των μνημών ROM είναι 24bit, ο διαχωρισμός των στοιχείων μεταξύ τους θα γίνει μετά (όταν υποστούν την επεξεργασία). Η θέση μνήμης κάθε CAM είναι 8bit. Ο αναγνώστης μπορεί ακόμη να παρατηρήσει ότι τα ζεύγη μνημών CAM και ROM έχουν το γνώρισμα B_2 κοινό. Το αποτέλεσμα αυτής της ομοιότητας είναι όταν το στοιχείο προς εξέταση βρεθεί στην CAM, θα δείχνει στη σωστή διεύθυνση μνήμης στη μνήμη ROM.

CAM A

B_2
4
1
5
2
7
3

ROM A

B_1	B_2	B_3
2	4	8
1	1	1
2	5	5
1	2	1
3	7	2
5	3	1

CAM B

B_2
5
2
6
4
1

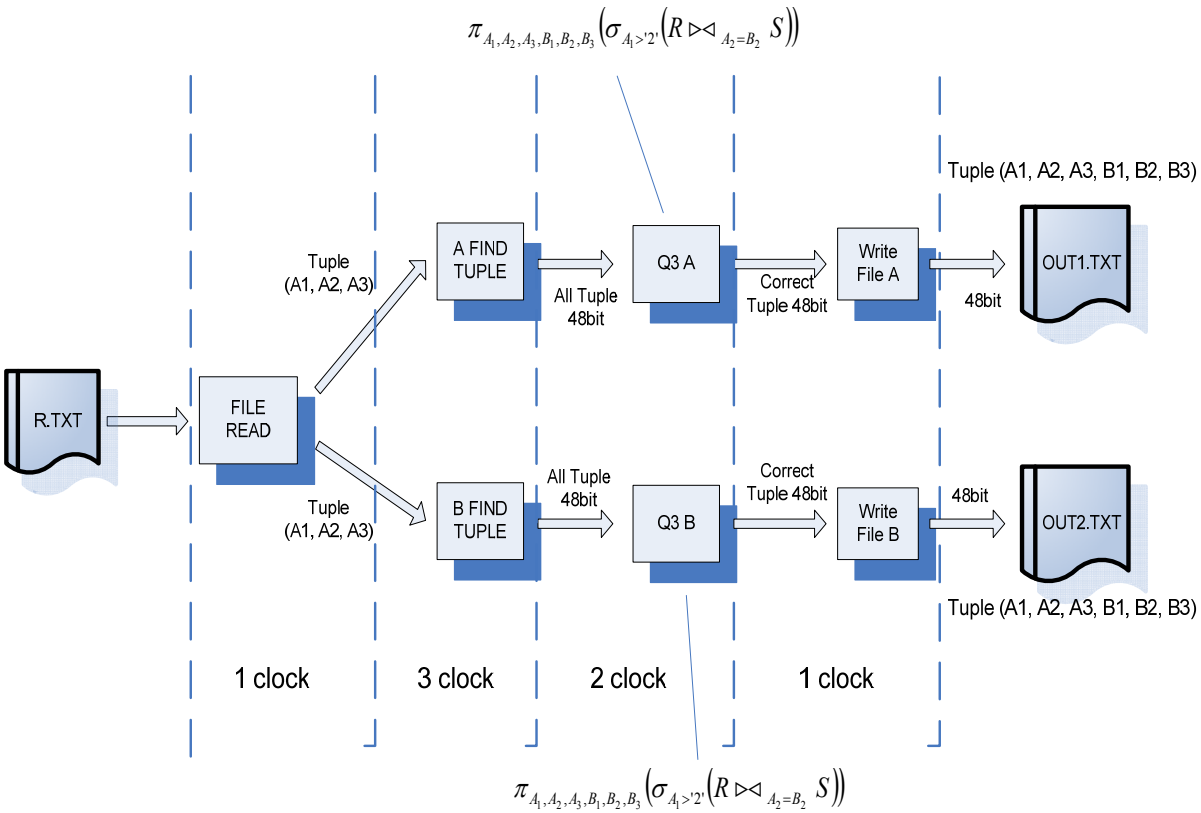
ROM B

B_1	B_2	B_3
2	5	4
1	2	3
4	6	9
8	4	2
1	1	3

Εικόνα 3.33: Τα περιεχόμενα κάθε μνήμης της επερώτησης Q3

Λογική Αρχιτεκτονική Δομή Επερώτησης Q3

Κατόπιν όλων αυτών, στη σχεδίαση για τη λύση της επερώτησης Q3 θα πρέπει να υπάρχουν δύο λογικά κυκλώματα με ανεξάρτητες λειτουργίες και ανεξάρτητα σήματα ελέγχου. Η σχεδίαση και των δυο λογικών κυκλωμάτων θα είναι η ίδια και το μόνο κοινό στοιχείο θα είναι το λογικό κύκλωμα εισαγωγής στοιχείων. Η αρχιτεκτονική δομή του λογικού κυκλώματος που δίνει λύση στην επίλυση ερωτημάτων της επερώτησης Q3 είναι αυτό που φαίνεται στην Εικόνα 3.34.



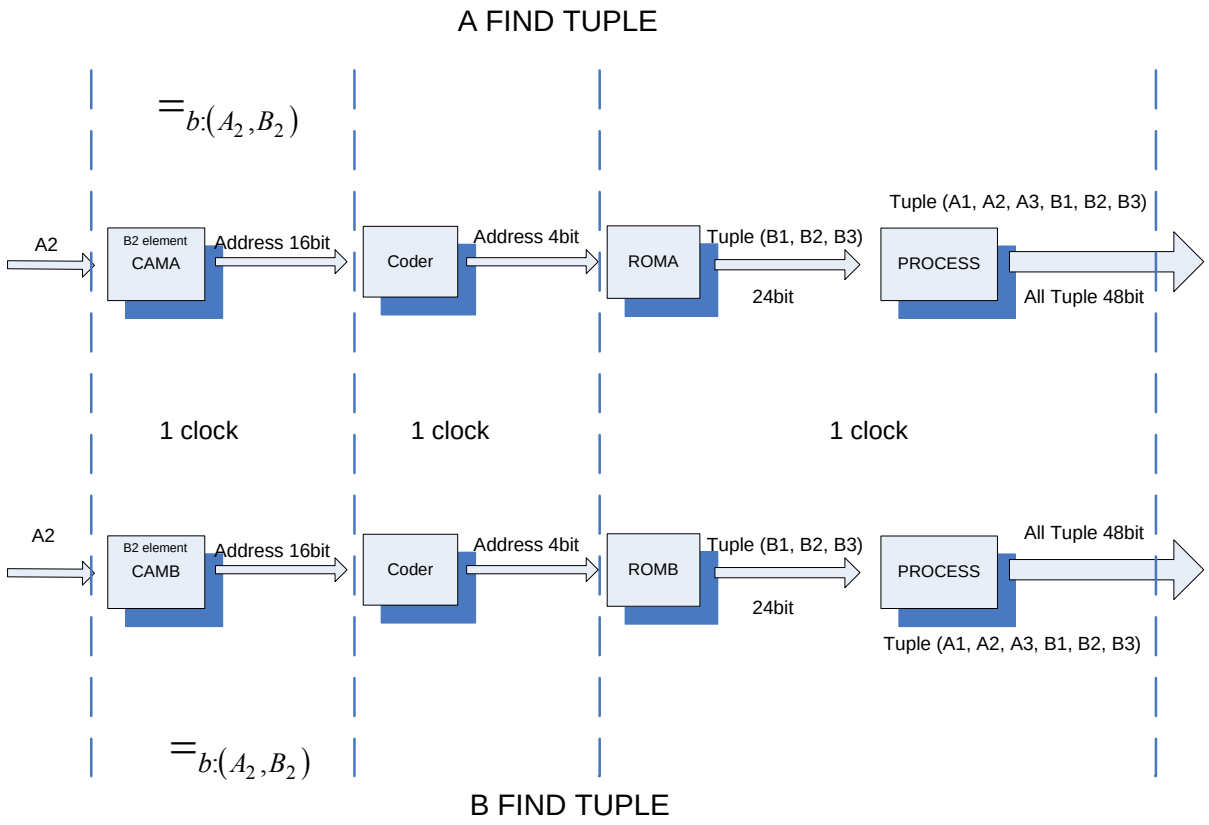
Εικόνα 3.34: Αρχιτεκτονική δομή λογικού κυκλώματος που δίνει λύση στην επίλυση ερωτημάτων της επερώτησης Q3

Όπως μπορεί να φανεί στην Εικόνα 3.34 οι διεργασίες που θα επιτελέσουν τα δυο κυκλώματα είναι ανεξάρτητα. Το καθένα από αυτά θα ενεργεί αυτόνομα για την παραγωγή των ζητούμενων. Μόνο το λογικό κύκλωμα **FILE READ,** που κάνει την ανάγνωση των στοιχείων από το αρχείο **R.txt,** είναι ίδιο και τα αποτελέσματα που παράγει μοιράζονται από κοινού στα δύο κυκλώματα. Η λογικές μονάδες **A FIND TUPLE** και **B FIND TUPLE** βρίσκουν ανεξάρτητα τη ζητούμενη πλειάδα και τη μεταφέρουν στις λογικές μονάδες **Q3A** και **Q3B,** όπου και θα εκτελέσουν την επερώτηση. Τέλος οι λογικές μονάδες **WRITE FILE A** και **WRITE FILE B** θα εξάγουν τα αποτελέσματα σε δύο ανεξάρτητα αρχεία txt τα **OUT1.TXT** και **OUT2.TXT**. Στην αρχιτεκτονική αυτή δεν φαίνονται τα σήματα ελέγχου που χρησιμοποιούνται για το συγχρονισμό και τον έλεγχο που υφίσταται, όλη η λογική μονάδα. Θεωρήθηκε σκόπιμο να μην εμφανιστούν, διότι θα περιέπλεκαν την εικόνα.

Ο *χρόνος εκτέλεσης* (*latency*) ισούται με 7 (7 κύκλοι ρολογιού, latency = 7) και το *ζητούμενο ποσοστό* (*issue rate*) ισούται με 3. Είναι, ένας κύκλος ρολογιού για τις λογικές μονάδες FILE READ και WRITE FILE (άρα issue rate = 1), δύο κύκλοι ρολογιού για τη μονάδα Q3 (άρα issue rate = 2) και 3 κύκλος ρολογιού για τη λογική μονάδα FIND TUPLE, άρα το συνολικό ζητούμενο ποσοστό (issue rate) του κυκλώματος ισούται με 3.

Λογική Μονάδα FIND TUPLE

Οι λογικές μονάδες **FIND TUPLE** περιέχουν τη δομή που φαίνεται στην Εικόνα 3.35. Είναι παρόμοια με αυτή της Εικόνας 3.30 απλά σε αυτήν προστέθηκε μια επιπλέον λογική μονάδα η *Process* η οποία επιστρέφει και τις δύο πλειάδες μαζί (τις σχέσης R και S), το μέγεθος αυτής **48bit**. Επιστρέφει τις πλειάδες οι οποίες ταυτοποιήθηκαν (έγινε δηλ *matching*) στις μνήμες CAM και ROM. Στο επόμενο στάδιο, αυτό της εκτέλεσης των συγκρίσεων, θα γίνει περαιτέρω επεξεργασία της πλειάδας αυτής με απώτερο στόχο την παραγωγή της λύσης. Η λογική αυτή υιοθετήθηκε από το σχεδιαστή για να δώσει πιο εύκολα τη ζητούμενη απάντηση.



Εικόνα 3.35: Οι λογικές μονάδες FIND TUPLE και η λογική δομή τους, της επερώτησης Q3

Λογική Μονάδα Q3A και Q3B

Η λογική μονάδα που υλοποιήθηκε με τη VHDL για να επιλύει την επερώτηση Q3 (Q3A και Q3B) φαίνεται στην Εικόνα 3.36. Έχει παρόμοια σχεδίαση με αυτήν της επερώτησης Q2 (Εικόνα 3.25) με τη μόνη διαφορά, πέραν των τελεστών και των μεταβλητών που χρησιμοποιούνται για την επίλυση της επερώτησης, τη χρήση του κώδικα συγχρονισμού με το ρολόι "**if clk'event and clk='1' then**" και "**end if;** ", γραμμές 111 και 119. Σε αυτήν την περίπτωση έγινε χρήση του συγκεκριμένου κώδικα γιατί υπήρχε πρόβλημα στο συγχρονισμό των δεδομένων που βγαίνουν

προς τα έξω. Θεωρήθηκε σκόπιμο από το σχεδιαστή η χρήση του κώδικα για να συγχρονιστούν και να παραχθούν τα σωστά αποτελέσματα. Το ίδιο συμβαίνει με τη λογική μονάδα **delay47** (γραμμές 77 … 83) η οποία χρησιμοποιείται για να συγχρονιστούν τα δεδομένα.

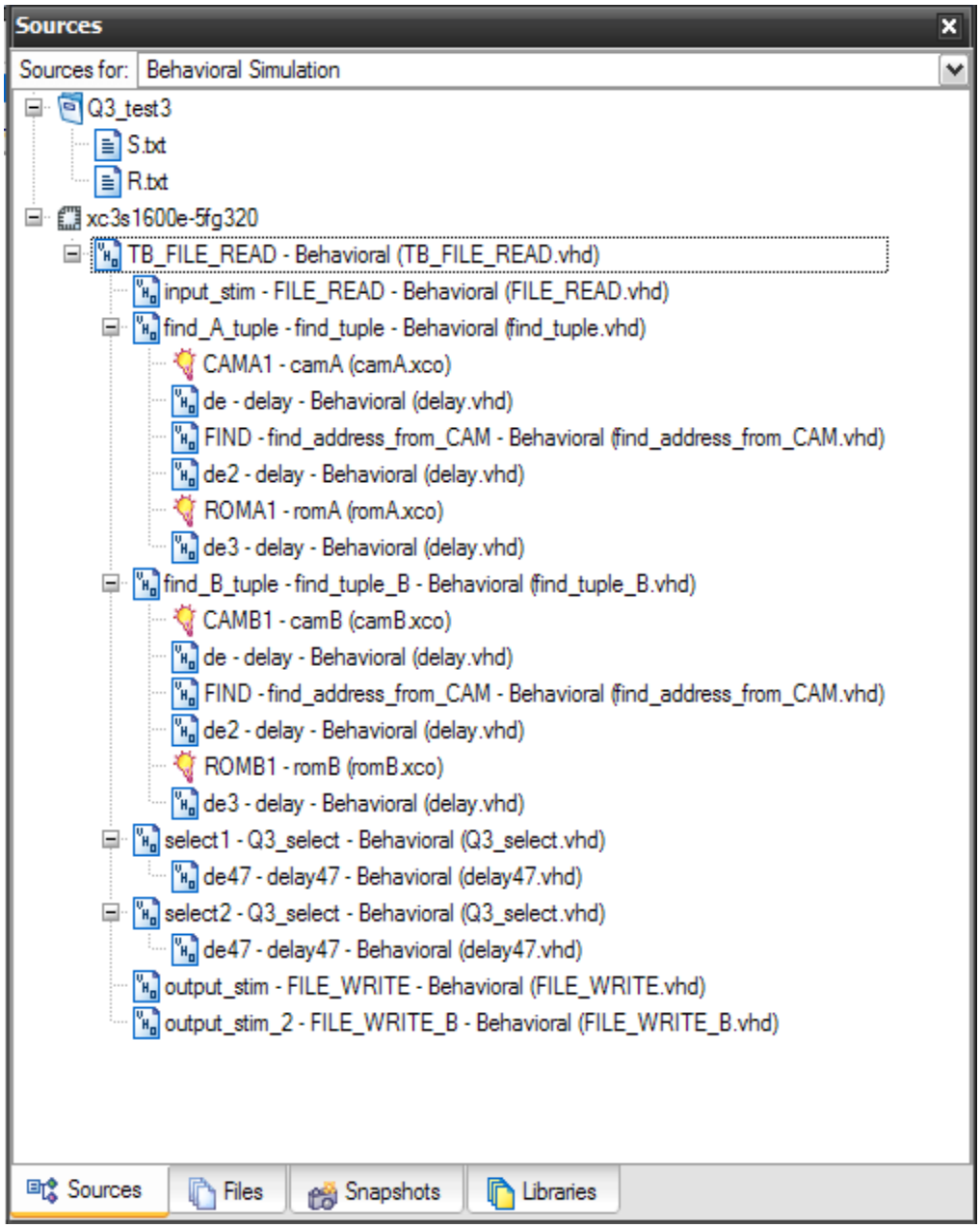
```
67  begin
68  -- signals  initialization
69  s_in_data <= tuple_A_and_B;
70  A1 <= s_in_data (47 downto 40);      -- initialize first element A1 of the tuple
71  A2 <= s_in_data (39 downto 32);      -- initialize second element A2 of the tuple
72  A3 <= s_in_data (31 downto 24);      -- initialize third element A3 of the tuple
73  B1 <= s_in_data (23 downto 16);      -- initialize first element B1 of the tuple
74  B2 <= s_in_data (15 downto  8);      -- initialize second element B2 of the tuple
75  B3 <= s_in_data ( 7 downto  0);      -- initialize third element B3 of the tuple
76  ------------------------------------------- delay47 ------------------------------------------
77  de47: delay47                        -- delay for synchronization
78      port map (
79                CLK   => clk,
80                RST   => rst,
81             data_in => s_in_data,
82            data_out => data_in2
83            );
84  ------------------------------------------- Q3_A ---------------------------------------------
85  Q3_A: process (clk)                  -- start process Q3_A
86     begin
87     if clk'event and clk='1' then     -- synchronization whit clock
88        if (A1 > "00000010") then      -- the control element A2 > 2
89           valid_bit_A <= '1' ;        -- valid bit A = '1'
90        else
91           valid_bit_A <= '0' ;        -- valid bit = '0'
92           end if;
93     end if;
94  end process Q3_A;                    -- end process Q3_A
95
96  ------------------------------------------- Q3_B ---------------------------------------------
97  Q3_B: process (clk)                  -- start process Q3_B
98     begin
99     if (clk'event and clk='1') then   -- synchronization whit clock
100       if (A2 = B2) then               -- the control element A2 = B2
101          valid_bit_B <= '1' ;         -- valid bit B = '1'
102       else
103          valid_bit_B <= '0' ;         -- valid bit B = '0'
104          end if;
105    end if;
106 end process Q3_B;                    -- end process Q3_B
107
108 ------------------------------------------- Q3_C ---------------------------------------------
109 Q3_C:process (clk,valid_bit_A,valid_bit_B)    -- start process Q3_C
110    begin
111       if clk'event and clk='1' then                          -- synchronization whit clock
112          if (valid_bit_A = '1' and valid_bit_B = '1') then   -- if the comparisons are correct
113             out_data <= data_in2;           -- output correct data
114             valid_bit <= '1' ;              -- valid bit = '1'
115          else
116             out_data <= (others =>'0');     -- output null data
117             valid_bit <= '0' ;              -- valid bit = '0'
118          end if;
119       end if;
120 end process Q3_C;                          -- end process Q3_C
121
122 end Behavioral;
```

Εικόνα 3.36: Κώδικας VHDL που υλοποιεί την επερώτηση Q3 του Τύπου 3.18

Και σε αυτήν την περίπτωση η τροποποίηση των τελεστών σύγκρισης και των λογικών τελεστών γίνεται γρήγορα και εύκολα, για περαιτέρω αλλαγές χρειάζεται εμπειρία και προσοχή.

Σχεδίαση ISE

Οι λογικές μονάδες που χρησιμοποιήθηκαν κατά την επίλυση φαίνονται στην Εικόνα 3.37 και έχουν παρθεί από το παράθυρο *Project Navigator* (Εικόνα Δ.2 Παράρτημα Δ) του προγράμματος σχεδίασης CAD ISE.



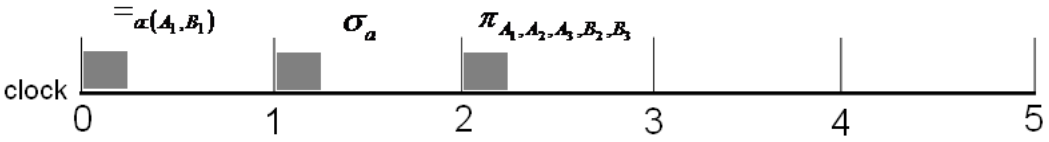
Εικόνα 3.37: Οι λογικές μονάδες της σχεδίασης της λύσης της επερώτησης Q3 από το παράθυρο Project Navigator του προγράμματος σχεδίασης CAD ISE

Στο Κεφάλαιο 5 ο αναγνώστης θα έχει την ευκαιρία να δει και σε γραφική μορφή τα αποτελέσματα αυτής της εκτέλεσης όπως και τα παραγόμενα δεδομένα της επερώτησης Q3.

3.3.4 Επέκταση Επερώτησης Q4 στη VHDL

Χρονική Εκτέλεση Επερωτήσεων

Η χρονική ακολουθία που επιτελεί η επερώτηση Q4 σε ένα ΣΒΔ είναι της μορφής που φαίνεται στην Εικόνα 3.38. Όπως φαίνεται απαιτούνται τρεις κύκλοι ρολογιού για τη διευθέτηση της επερώτησης Q4 από το σύστημα.



Εικόνα 3.38: Συμβατική χρονική ακολουθία επερώτησης Q4 από ένα ΣΒΔ ανεξαρτήτου επεξεργαστικής ταχύτητας.

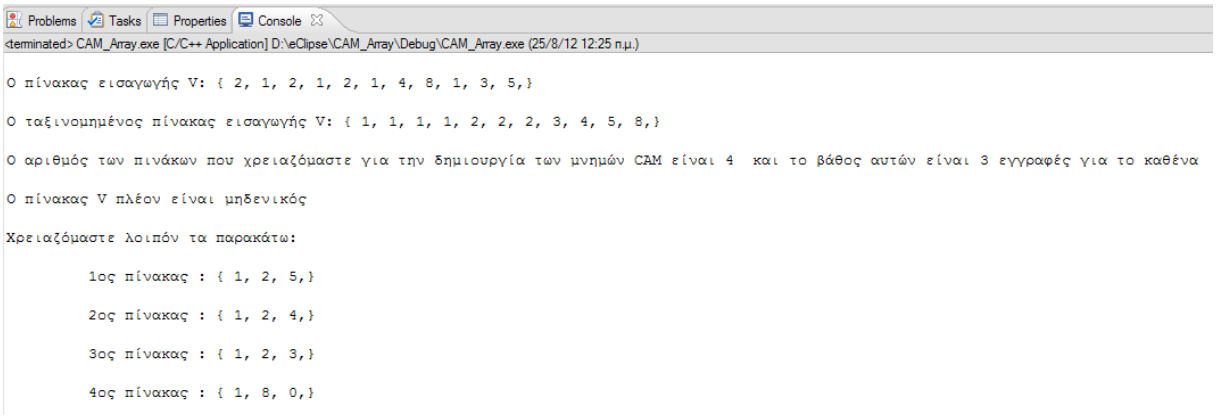
Αυτοί οι χρόνοι ελαχιστοποιούνται σε έναν κύκλο ρολογιού στη σχεδίαση με VHDL (Εικόνα 3.39). Όπως μπορεί να παρατηρηθεί η σύγκριση του τελεστή «=» εκτελείται και αμέσως μετά θα εξαχθούν τα ζητούμενα δεδομένα στο ίδιο κύκλο ρολογιού. Εδώ, εν αντιθέσει με την Εικόνα 3.29, φαίνεται καθαρά ότι η μη χρήση πολλών τελεστών εξυπηρετεί πολύ την παραγωγή γρήγορων κυκλωμάτων. Παρόλο αυτά όμως, η χρήση πολλών τελεστών και μεταβλητών συμβάλουν στη δημιουργία έξυπνων λογικών κυκλωμάτων.

Εικόνα 3.39: Χρονική ακολουθία επερώτησης Q4 μετά από σχεδίαση στη γλώσσα VHDL και εκτέλεση του στη συσκευή FPGA

Περιεχόμενα Μνήμων CAM και ROM

Στην περίπτωση της επερώτησης Q4 επιλέχθηκε στη μνήμη ROM να βρίσκεται πάλι ο πίνακας S (Εικόνα 3.4) και στη μνήμη CAM να βρίσκεται το γνώρισμα (στήλη) B_1. Όπως μπορεί κανείς να παρατηρήσει το γνώρισμα B_1 έχει περισσότερα από ένα ίδια στοιχεία, για να γίνει λοιπόν η υλοποίηση της επερώτησης Q4 σύμφωνα με όσα έχουν ειπωθεί, θα πρέπει να γίνει χρήση του αλγόριθμου CAM_ARRAY για άλλη μια φορά.

Το γνώρισμα B_1 έχει την ακολουθία των αριθμών {2,1,2,1,2,1,4,8,1,3,5}, το παραγόμενο αποτέλεσμα μετά την εκτέλεση του αλγόριθμου φαίνεται στην Εικόνα 3.40.



Εικόνα 3.40: Εκτέλεση του αλγόριθμου CAM_ARRAY στο γνώρισμα B_1του πίνακα S, της επερώτησης Q4

Από το αποτέλεσμα της εκτέλεσης του αλγόριθμου CAM_ARRAY στην ακολουθία αριθμών του γνωρίσματος B1 της σχέσης S, προκύπτουν ότι πρέπει να χρησιμοποιηθούν τέσσερις μνήμες CAM και τέσσερις μνήμες ROM.

Συμβουλή: Σε περίπτωση που όλα τα στοιχεία ή ένα μεγάλο ποσοστό των στοιχείων σε έναν πίνακα είναι ίδια, τότε θα πρέπει να μελετηθεί μια διαφορετική λύση από την προτεινόμενη.

Στην Εικόνα 3.41 φαίνεται ο καταμερισμός των στοιχείων του πίνακα S στις μνήμες CAM και ROM.

B_1	B_2	B_3	ROM
2	4	8	romA
1	1	1	romA
2	5	5	romB
1	2	1	romB
2	5	4	romC
1	2	3	romC
4	6	9	romB
8	4	2	romD
1	1	3	romD
3	7	2	romC
5	3	1	romA

B_1	CAM
4	romA
1	romA
5	romB
2	romB
5	romC
2	romC
6	romB
4	romD
1	romD
7	romC
3	romA

Εικόνα 3.41: Κατανομή στοιχείων πίνακα S στις αντίστοιχες μνήμες CAM και ROM

Στην Εικόνα 3.42 φαίνονται τα περιεχόμενα των μνημών που θα χρησιμοποιηθούν για την επίλυση της επερώτησης Q4. Κάθε θέση μνήμης των μνημών ROM είναι 24bit, ο διαχωρισμός των στοιχείων μεταξύ τους θα γίνει μετά (όταν υποστούν την επεξεργασία). Η θέση μνήμης κάθε CAM είναι 8bit. Ο αναγνώστης μπορεί ακόμη να παρατηρήσει ότι τα ζεύγη μνημών CAM και ROM έχουν το γνώρισμα B_1 κοινό. Το αποτέλεσμα αυτής της ομοιότητας είναι όταν το στοιχείο προς εξέταση βρεθεί στην CAM, θα δείχνει στη σωστή διεύθυνση μνήμης στη μνήμη ROM.

CAM A

B_1
2
1
5

ROM A

B_1	B_2	B_3
2	4	8
1	1	1
5	3	1

CAM B

B_1
2
1
4

ROM B

B_1	B_2	B_3
2	5	5
1	2	1
4	6	9

CAM C

B_1
2
1
3

ROM C

B_1	B_2	B_3
2	5	4
1	2	3
3	7	2

CAM D

B_1
8
1

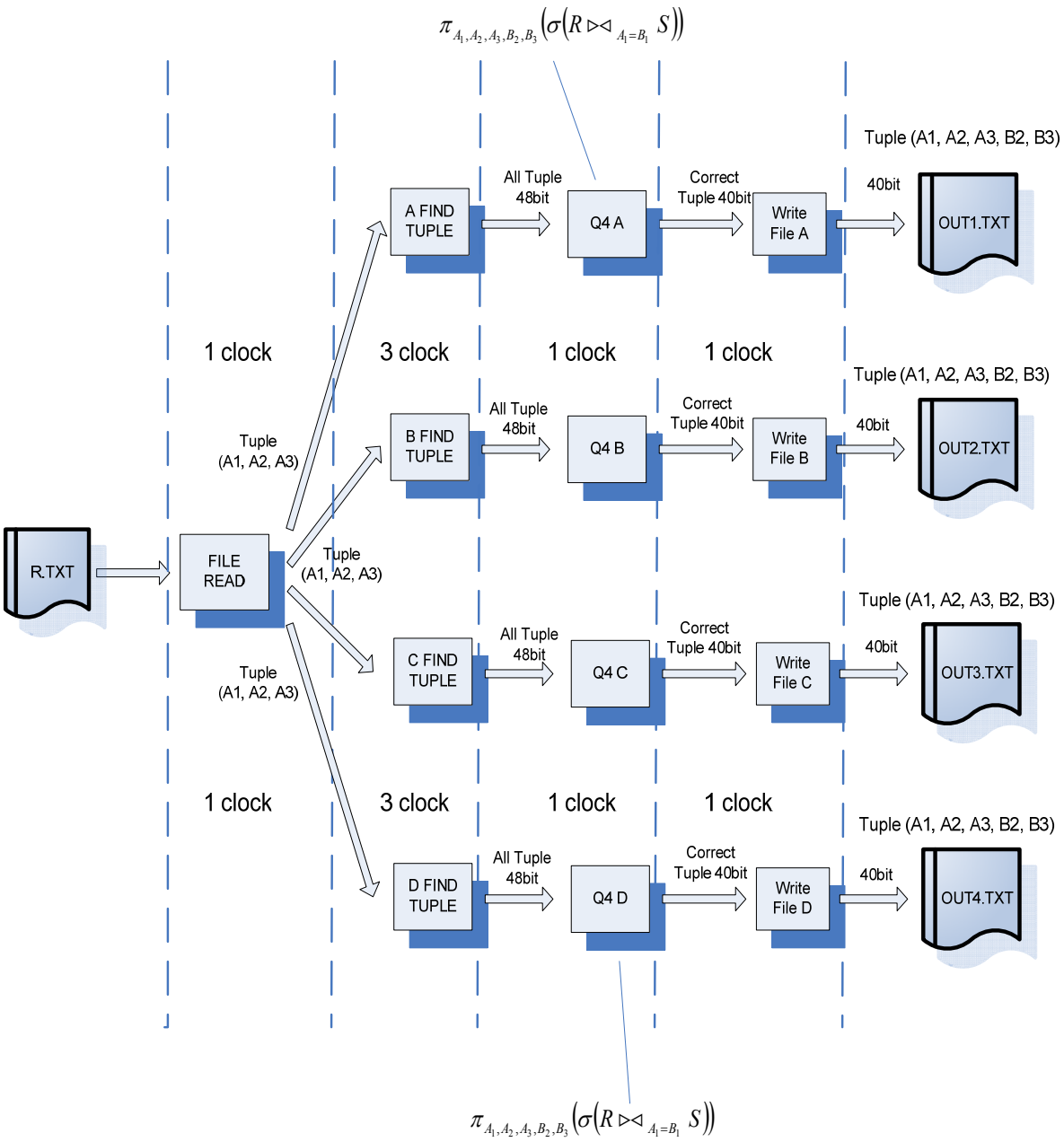
ROM D

B_1	B_2	B_3
8	4	2
1	1	3

Εικόνα 3.42: Τα περιεχόμενα κάθε μνήμης της επερώτησης Q4

Λογική Αρχιτεκτονική Δομή Επερώτησης Q4

Κατόπιν όλων αυτών, στη σχεδίαση για τη λύση της επερώτησης Q4 θα πρέπει να υπάρχουν τέσσερα λογικά κυκλώματα με ανεξάρτητες λειτουργίες και ανεξάρτητα σήματα ελέγχου. Η σχεδίαση και των τεσσάρων λογικών κυκλωμάτων θα είναι η ίδια και το μόνο κοινό στοιχείο θα είναι το λογικό κύκλωμα εισαγωγής στοιχείων. Η αρχιτεκτονική δομή του λογικού κυκλώματος που δίνει λύση στην επίλυση ερωτημάτων της επερώτησης Q4 είναι αυτό που φαίνεται στην Εικόνα 3.43.



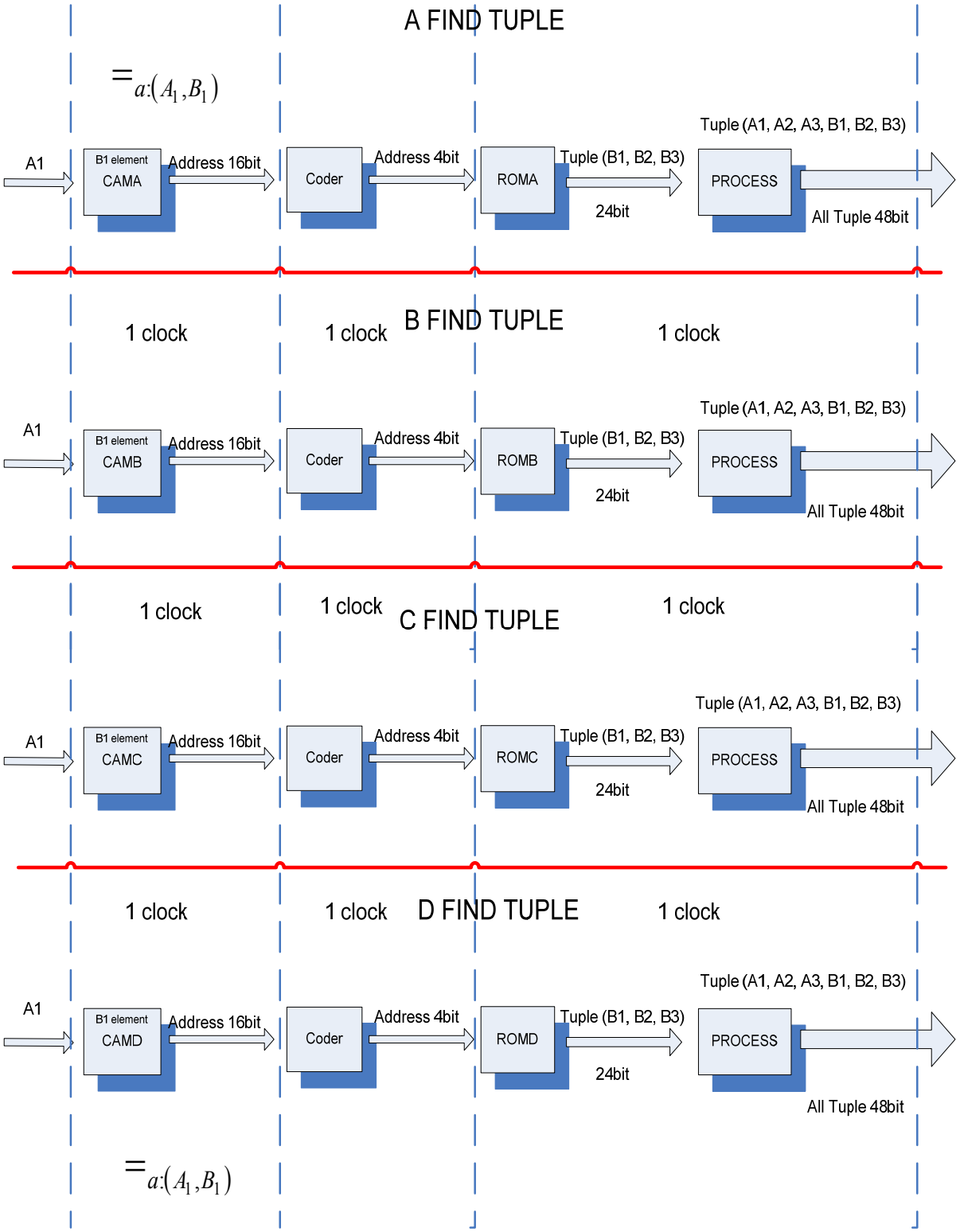
Εικόνα 3.43: Αρχιτεκτονική δομή λογικού κυκλώματος που δίνει λύση στην επίλυση ερωτημάτων της επερώτησης Q4

Όπως μπορεί να φανεί στην Εικόνα 3.43 οι διεργασίες που θα επιτελέσουν τα τέσσερα κυκλώματα είναι ανεξάρτητα. Το καθένα από αυτά θα ενεργεί αυτόνομα για την παραγωγή των ζητούμενων. Και εδώ μόνο το λογικό κύκλωμα **FILE READ,** που κάνει την ανάγνωση των στοιχείων από το αρχείο **R.txt,** είναι ίδιο και τα αποτελέσματα που παράγει μοιράζονται από κοινού στα τέσσερα κυκλώματα. Η λογικές μονάδες **FIND TUPLE** βρίσκουν ανεξάρτητα η κάθε μια, τη ζητούμενη πλειάδα και τη μεταφέρουν στις υπόλοιπες λογικές μονάδες όπου και θα εκτελέσουν την επερώτηση. Τέλος οι λογικές μονάδες **WRITE FILE** θα εξάγουν τα αποτελέσματα σε τέσσερα ανεξάρτητα αρχεία txt τα **OUT1.TXT, OUT2.TXT, OUT3.TXT**. και **OUT4.TXT**.

Ο *χρόνος εκτέλεσης* (*latency*) ισούται με 6 (6 κύκλοι ρολογιού, latency = 6) και το *ζητούμενο ποσοστό* (*issue rate*) ισούται με 3. Είναι, ένας κύκλος ρολογιού για τις λογικές μονάδες FILE READ, Q4 και WRITE FILE (άρα issue rate = 1), και 3 κύκλος ρολογιού για τη λογική μονάδα FIND TUPLE, άρα το συνολικό ζητούμενο ποσοστό (issue rate) του κυκλώματος ισούται με 3.

Λογική Μονάδα FIND TUPLE

Οι λογικές μονάδες **FIND TUPLE** της επερώτησης Q4 φαίνονται στην Εικόνα 3.44 και περιέχουν την ίδια δομή που μελετήθηκε και στην Εικόνα 3.35, επιτελούν την ίδια ακριβώς λειτουργία. Απλά σε αυτή την περίπτωση έχουμε τέσσερα λογικά κυκλώματα τα A FIND TUPLE, B FIND TUPLE, C FIND TUPLE και D FIND TUPLE και όχι δύο. Το μόνο που χρειάσθηκε αλλαγή είναι τα περιεχόμενα των μνημών CAM και ROM. Και τα τέσσερα κυκλώματα λειτουργούν ακριβώς παράλληλα μεταξύ τους χωρίς το ένα να υστερεί πόρους και ενέργεια από το άλλο. Άλλωστε αυτή είναι η εξερετική δυνατότητα λειτουργίας των συσκευών FPGA, την οποία και εκμεταλλεύεται αυτή η σχεδίαση.



Εικόνα 3.44: Οι λογικές μονάδες FIND TUPLE και η λογική δομή τους, της επερώτησης Q4

Λογική Μονάδα Q4A, Q4B, Q4C, Q4D

Η λογική μονάδα που υλοποιήθηκε με τη VHDL για να επιλύει την επερώτηση Q4 (Q4A, Q4B, Q4C και Q3D) φαίνεται στην Εικόνα 3.45. Στη γραμμή 67 φαίνεται η τοποθέτηση των γνωρισμάτων ώστε να παράγουν την πλειάδα των 40bit. Στις γραμμές 70 ... 81 εκτελείται η επερώτηση Q4

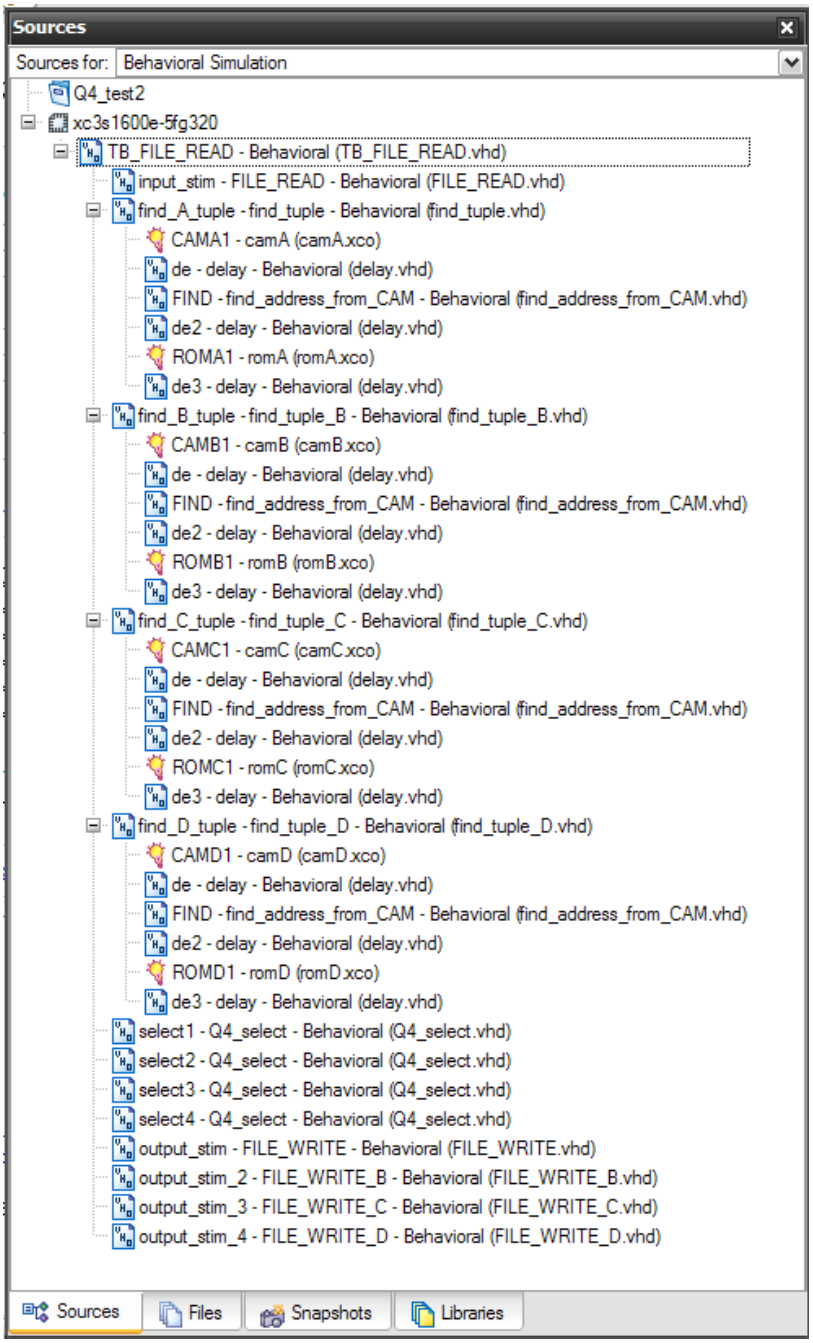
επάνω στις πλειάδες εισόδου. Στη γραμμή 73 φαίνεται η σύγκριση των γνωρισμάτων A_1 και B_1, όπως και με τις άλλες επερωτήσεις έτσι και με αυτή, αν ο σχεδιαστής θέλει να τροποποιήσει τους τελεστές και τις μεταβλητές, το κάνει πολύ εύκολα.

```
41
42  architecture Behavioral of Q4_select is
43
44  -- control signals
45  signal A1   :  std_logic_vector(7 downto 0);          -- the first element A1 of the tuple
46  signal A2   :  std_logic_vector(7 downto 0);          -- the second element A2 of the tuple
47  signal A3   :   std_logic_vector(7 downto 0);         -- the third element A3 of the tuple
48
49  signal B1   :  std_logic_vector(7 downto 0);          -- the first element B1 of the tuple
50  signal B2   :  std_logic_vector(7 downto 0);          -- the second element B2 of the tuple
51  signal B3   :   std_logic_vector(7 downto 0);         -- the third element B3 of the tuple
52
53  signal s_in_data  :  STD_LOGIC_VECTOR (47 downto 0);  -- the signal from input data
54  signal data_out_A1_A2_A3_B2_B3   :  std_logic_vector(39 downto 0) := (others => '0');
55
56  begin
57  -- signals  initialization
58  s_in_data <= tuple_A_and_B;
59  A1 <= s_in_data (47 downto 40);     -- initialize first element A1 of the tuple
60  A2 <= s_in_data (39 downto  32);    -- initialize second element A2 of the tuple
61  A3 <= s_in_data (31 downto  24);    -- initialize third element A3 of the tuple
62  B1 <= s_in_data (23 downto 16);     -- initialize first element B1 of the tuple
63  B2 <= s_in_data (15 downto  8);     -- initialize second element B2 of the tuple
64  B3 <= s_in_data ( 7 downto  0);     -- initialize third element B3 of the tuple
65
66  -- initialize the correct data A1, A2, A3, B2, B3
67  data_out_A1_A2_A3_B2_B3 <= A1 & A2 & A3 & B2 & B3;
68
69  ------------------------------------------- Q4-----------------------------------------------
70  Q4: process (clk)                       -- start process Q4
71     begin
72     if (clk'event and clk='1') then      -- synchronization whit clock
73        if (A1 = B1) then                 -- the control element A1 = B1
74           out_data <= data_out_A1_A2_A3_B2_B3;  -- output correct data
75              valid_bit <= '1' ;          -- valid bit = '1'
76        else
77           out_data <= (others =>'0');    -- output null data
78              valid_bit <= '0' ;          -- valid bit = '0'
79           end if;
80     end if;
81  end process Q4;                         -- end process Q4
82
83  end Behavioral;
84
```

Εικόνα 3.45: Κώδικας VHDL που υλοποιεί την επερώτηση Q4 του Τύπου 3.20

Σχεδίαση ISE

Οι λογικές μονάδες που χρησιμοποιήθηκαν κατά την επίλυση φαίνονται στην Εικόνα 3.46 και έχουν παρθεί από το παράθυρο *Project Navigator* (Εικόνα Δ.2 Παράρτημα Δ) του προγράμματος σχεδίασης CAD ISE.



Εικόνα 3.46: Οι λογικές μονάδες της σχεδίασης της λύσης της επερώτησης Q4 από το παράθυρο Project Navigator του προγράμματος σχεδίασης CAD ISE

Στο Κεφάλαιο 5 ο αναγνώστης θα έχει την ευκαιρία να δει και σε γραφική μορφή τα αποτελέσματα αυτής της εκτέλεσης όπως και τα παραγόμενα δεδομένα της επερώτησης Q4.

3.4 Συμπεράσματα

Σε αυτό το κεφάλαιο ο αναγνώστης είχε την ευκαιρία να δει πως οι επερωτήσεις SQL σχηματίζονται στη σχεσιακή άλγεβρα αλλά και τις επεκτάσεις τους στη γλώσσα VHDL. Ο συγκεκριμένος τρόπος υλοποίησης επερωτήσεων στη γλώσσα VHDL δεν είναι ο μοναδικός και δεν αποτελεί τη βέλτιστη λύση. Η συγγραφή και σχεδίαση κυκλωμάτων έγκειται περισσότερο στην εμπειρία αλλά και την προσωπικότητα του κάθε σχεδιαστή. Οπότε, τα παραγόμενα κυκλώματα και η συγκεκριμένη προσπάθεια αποτελεί περισσότερο εκκολαπτόμενη προσέγγιση παρά ώριμη και εξαντλητικά ελεγχόμενη. Η συγκεκριμένη προσπάθεια έγινε περισσότερο για να μυηθεί ο αναγνώστης στην τεχνολογία των συσκευών FPGA και να πάρει μια γεύση για το πώς αυτά αντιμετωπίζουν θέματα που απασχολούν τις βάσεις δεδομένων.

Τα παραγόμενα λογικά κυκλώματα είχαν πολύ καλές χρονικές επιδώσεις σε σύγκριση με αυτά των παραδοσιακών χρονικών εκτελέσεων. Παράλληλα, η δυνατότητα τους ως προς την παράλληλη επεξεργασία έδειξαν την ικανότητα τους στη διεκπεραίωση μεγάλου όγκου πληροφοριών σε ελάχιστο χρόνο. Η ελάττωση των κύκλων εκτέλεσης σε σύγκριση και με τις χαμηλές επιδώσεις ενέργειας [21], καθιστά τις συσκευές FPGA αρκετά ελκυστικές ως προς τη χρήση τους σε συστήματα βάσεων δεδομένων.

Τα συγκεκριμένα παραδείγματα πιστοποιούν τη δυνατότητα χρήσης των συσκευών FPGA και της γλώσσας VHDL στη συγγραφή και παραγωγή κυκλωμάτων που υλοποιούν επερωτήσεις γλώσσας SQL αλλά και πράξεων της σχεσιακής άλγεβρας. Το μόνο που μένει είναι η καταγραφή και αρχειοθέτηση σε βιβλιοθήκη αυτών των λογικών μονάδων για μελλοντική χρήση. Παράλληλα, ο αναγνώστης διαπίστωσε πόσο εύχρηστο και βολικό είναι η χρήση έτοιμων λογικών μονάδων (κυκλωμάτων) σε μελλοντικά έργα.

Από τα παραγόμενα κυκλώματα, ως τελικό συμπέρασμα, ο αναγνώστης διαπιστώνει την ικανότητα των συσκευών FPGA για χρήση τους στην επεξεργασία δεδομένων. Για του λόγου το αληθές, στο Κεφάλαιο 5 ο αναγνώστης θα έχει την ευκαιρία να δει και τις παραγόμενες κυματομορφές που καταδεικνύουν, αυτή τη θεωρητική προσέγγιση.

3.5 Αναφορές

[19] R. Elmasri, S.B. Navathe. «Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων». Εκδόσεις Δίαυλος, ISBN 978-960-531-219-0, Αθήνα 2001.

[20] Rene Mueller, Jens Teubner, Gustavo Alonso. «Streams on Wires — A Query Compiler for FPGAs». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2009.

[21] Rene Mueller, Jens Teubner, Gustavo Alonso. «Data Processing on FPGAs». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2009.

[39] Μιχάλης Ξένος, Δημήτρης Χρηστοδουλάκης. «Βάσεις Δεδομένων». Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα 2000.

Κεφάλαιο 4

Εισαγωγή Στοιχείων

Σε αυτήν την ενότητα παρουσιάζεται ο τρόπος εισαγωγής των δεδομένων που είναι για επεξεργασία τόσο κατά την προσομοίωση όσο και κατά την πραγματική υλοποίηση στο FPGA. Επίσης, παρουσιάζεται η μνήμη CAM (Contact Addressable Memory) και η μνήμη RAM (Block RAM ή ROM), οι δύο αυτές μνήμες χρησιμεύουν στη διευθέτηση των λύσεων των επερωτήσεων. Θα αναλυθεί ο τρόπος της εισαγωγής των δεδομένων που χρησιμεύουν στη λύση των ερωτημάτων, δηλαδή των τρόπο αρχικοποίησης των μνημών CAM και RAM(ROM). Επίσης παρουσιάζεται ο αλγόριθμος *CAM_ARRAY,* ο οποίος υπολογίζει τον ακριβή αριθμό μνημών CAM που πρέπει να χρησιμοποιηθούν στη λύση μιας επερώτησης, όταν τα δεδομένα αρχικοποίησης έχουν περισσότερα από ένα ίδια στοιχεία. Τέλος παρουσιάζεται ο τρόπος επικοινωνίας της συσκευής FPGA με τον Η/Υ μέσου του πρότυπου επικοινωνίας RS232, όπως και μέσου του πρωτοκόλλου επικοινωνίας UART, το οποίο υλοποιεί το πρότυπο RS232.

Αναλυτικά στην Ενότητα 4.1 παρουσιάζεται ο τρόπος ανάγνωσης των αρχείων *.txt κατά την προσομοίωση, στην Ενότητα 4.2 παρουσιάζεται η μνήμη CAM (Contact Addressable Memory) με όλες τις ιδιότητες της και τον τρόπο αρχικοποίησης της (Ενότητα 4.2.1), και αναφέρεται ο αλγόριθμος *CAM_ARRAY* (Ενότητα 4.2.3). Στην Ενότητα 4.3 παρουσιάζεται η μνήμη RAM (Block RAM, ROM) με τις ιδιότητες της αλλά και τον τρόπο αρχικοποίησης της. Τέλος στην Ενότητα 4.4

παρουσιάζεται το πρότυπο RS232 (Ενότητα 4.3.1) και το πρωτόκολλο επικοινωνίας UART (Ενότητα 4.4.2).

4.1 Ανάγνωση και Εγγραφή Αρχείων *.txt

Σε αυτήν την ενότητα θα παρουσιαστεί ο τρόπος ανάγνωσης των αρχείων που περιέχουν τις πληροφορίες προς επεξεργασία (αυτά που αποτελούν τη ΒΔ), τόσο κατά την προσομοίωση όσο και κατά την εκτέλεση στο υλικό. Και στις δύο περιπτώσεις θα γίνεται η ανάγνωση των αρχείων **R.txt** και **S.txt** που φαίνονται στην Εικόνα 5.4 και 5.7 αντίστοιχα. Τα περιεχόμενα των αρχείων είναι σε δυαδική μορφή μιας και όπως έχει αναφερθεί, η γλώσσα περιγραφής υλικού VHDL προσομοιώνει υλικό. Οπότε και τα δεδομένα που θα διαβάζονται, θα επεξεργάζονται και θα εγγράφονται, θα είναι σε δυαδική μορφή.

4.1.1 Ανάγνωση και Εγγραφή Αρχείου Κατά την Προσομοίωση

Αρχείο txt_util.vhd

Για την ανάγνωση και εγγραφή των αρχείων *.txt κατά την προσομοίωση χρησιμοποιήθηκε κώδικας VHDL, ο **txt_util.vhd** (Ενότητα Ζ.1.1 Παράρτημα Ζ), ο οποίος χρησιμοποιεί μια βιβλιοθήκη, την std.textio.all, για τη μετατροπή των χαρακτήρων από *ASCII* (***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, Αμερικανικός Πρότυπος Κώδικας για Ανταλλαγή Πληροφοριών,* Εικόνες Α.1 και Α.2 Παράρτημα Α) στους τύπους μεταβλητών και σημάτων που αναπτύχθηκαν στην Ενότητα 2.1.2. Αυτός το αρχείο λειτουργεί ως βοήθημα παρέχοντας συναρτήσεις που υλοποιούν όλες τις απαιτούμενες μετατροπές.

Αρχείο FILE_READ.vhd

Το αρχείο που χρειάζεται για την ανάγνωση των αρχείων *.txt είναι το **FILE_READ.vhd** (Ενότητα Ζ.1.2 Παράρτημα Ζ), στο οποίο ορίζεται το όνομα του αρχείου που πρέπει να διαβαστεί, ο ορισμός της φαίνεται στην Εικόνα 4.1. Όπως μπορεί να παρατηρήσει κανείς, στη γραμμή 36 ορίστηκε και το αρχείο το οποίο θα διαβαστεί, το οποίο είναι το S.txt.

```
34  entity FILE_READ is
35  generic (
36            stim_file:       string  := "S.txt"
37           );
38    port(
39         CLK              : in  std_logic;
40         RST              : in  std_logic;
41         Y                : out std_logic_vector(23 downto 0) := (others => '0');
42         EOG              : out std_logic
43        );
44  end FILE_READ;
```

Εικόνα 4.1: Ορισμός οντότητα ς FILE_READ

Αρχείο FILE_WRITE.vhd

Το αρχείο που χρειάζεται για την εγγραφή των αρχείων *.txt είναι το **FILE_WRITE.vhd** (Ενότητα Ζ.1.3 Παράρτημα Ζ), στο οποίο ορίζεται το όνομα του αρχείου που πρέπει να διαβαστεί, ο ορισμός της φαίνεται στην Εικόνα 4.2. Όπως μπορεί να παρατηρήσει κανείς στη γραμμή 34 ορίστηκε και το αρχείο στο οποίο θα γίνει η εγγραφή, το οποίο είναι το OUT.TXT.

```
32  entity FILE_WRITE is
33  generic (
34             write_file:       string  := "OUT.TXT"
35            );
36    port(
37           CLK    : in std_logic;
38           RST    : in std_logic;
39           X      : in std_logic_vector(23 downto 0);
40           EOG    : in std_logic;
41      valid_bit   : in std_logic
42           );
43  end FILE_WRITE;
```

Εικόνα 4.2: Ορισμός οντότητα ς FILE_WRITE

Αρχείο TB_FILE_READ.vhd

Το αμέσως επόμενο επίπεδο δημιουργίας για την ανάγνωση και εγγραφή ενός αρχείου κατά την προσομοίωση είναι το κύριο αρχείο (*main*), αυτό το οποίο θα συντονίσει όλες τις επιμέρους λειτουργίες για να μπορέσει να λειτουργήσει το λογικό κύκλωμα. Αυτό το αρχείο είναι το **TB_FILE_READ.vhd** (Ενότητα Ζ.1.4 Παράρτημα Ζ), σε αυτό το αρχείο οι οντότητες FILE_READ και FILE_WRITE θα πρέπει να οριστούν όπως δημιουργήθηκαν και αυτό για να μπορέσουν να συνδεθούν τα κυκλώματα και να παραχθεί η ανάγνωση και εγγραφή του αρχείου. Στην Εικόνα 4.3 φαίνεται στις γραμμές 39 – 47 ο ορισμός της οντότητας FILE_READ και στις γραμμές 49 – 60 ο ορισμός της οντότητας FILE_ WRITE. Επίσης για να μπορέσει ο κώδικας να λειτουργήσει θα πρέπει να είναι απενεργοποιημένες οι μεταβλητές εισόδου και εξόδου του κύριου αρχείου

TB_FILE_READ.vhd. Στην ίδια εικόνα ας παρατηρήσει ο αναγνώστης τις γραμμές 31 – 35 που έχουν απενεργοποιηθεί.

```
30  entity TB_FILE_READ is
31  ---port(                                                -- not used for simulation only for execute
32          --clk             : in  std_logic;              -- not used for simulation only for execute
33          --rst             : in  std_logic;              -- not used for simulation only for execute
34          --s_i_dieythinsi  :std_logic_vector(3 downto 0) -- not used for simulation only for execute
35          --);                                            -- not used for simulation only for execute
36  end TB_FILE_READ;
37
38  architecture Behavioral of TB_FILE_READ is
39  component FILE_READ
40     generic (stim_file:       string  := "S.txt");
41     port(
42           CLK              : in  std_logic;
43           RST              : in  std_logic;
44           Y                : out std_logic_vector(23 downto 0);
45           EOG              : out std_logic
46           );
47  end component;
48
49  component FILE_WRITE
50     generic (
51                 write_file:       string  := "OUT.TXT"
52               );
53     port(
54           CLK   : in std_logic;
55           RST   : in std_logic;
56           X     : in std_logic_vector(23 downto 0);
57           EOG   : in std_logic;
58        valid_bit : in std_logic
59           );
60  end component;
```

Εικόνα 4.3: Ορισμός οντότητας FILE_READ και FILE_ WRITE στο αρχείο TB_FILE_READ.vhd

Με τους παραπάνω κώδικες και τις διαδικασίες ορισμού τους, ορίστηκε ο τρόπος ανάγνωσης και εγγραφής ενός αρχείου *.txt κατά την προσομοίωση. Να σημειωθεί ότι οι παραγόμενη κώδικές είναι από την υλοποίηση της επερώτησης Q1. Επιπλέον αυτοί οι κώδικες χρησιμοποιήθηκαν και τροποποιήθηκαν από τους ενδεικτικούς που βρίσκονται στις ηλεκτρονικές σελίδες [24,25].

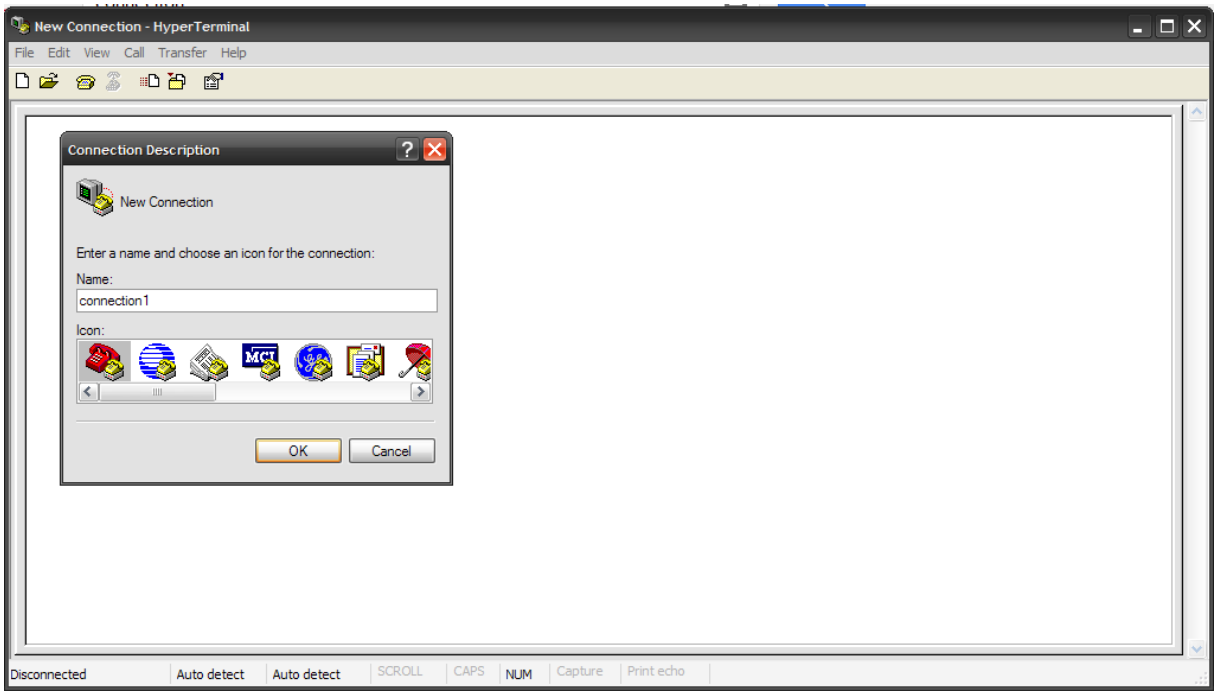
4.1.2 Ανάγνωση Εγγραφή Αρχείου Κατά την Πραγματική Εκτέλεση στο Υλικό FPGA

Η ανάγνωση και η εγγραφή αρχείων *.txt κατά την πραγματική εκτέλεση του υλικού FPGA, διαφέρει κατά πολύ μιας και τα αρχεία που αναπτύχθηκαν στην Ενότητα 4.1.1 δεν χρησιμοποιούνται. Το μόνο αρχείο που χρησιμοποιείται είναι μια τροποποιημένη έκδοση του TB_FILE_READ.vhd, αφού απαλείφονται οι οντότητες FILE_READ και FILE_ WRITE. Αυτό που πρέπει όμως να αναπτυχθεί είναι ο τρόπος εισαγωγής και εξαγωγής των αρχείων ανάγνωσης και εγγραφής.

Για να μπορέσει η συσκευή FPGA να λαμβάνει και να στέλνει δεδομένα από έναν Η/Υ θα πρέπει να διευθετηθεί ο τρόπος της επικοινωνίας τους (*connection*). Η διαδικασία που χρησιμοποιήθηκε στην παρούσα μεταπτυχιακή διατριβή είναι μέσω της θύρας COM1, όπου υποστηρίζει το πρότυπο επικοινωνίας RS232 (Ενότητα 4.4.1). Για να μπορέσει κάποιος να στείλει και να λάβει δεδομένα μέσω της θύρας COM1 παρέχεται ως βοήθεια ένα *τερματικό* (*terminal*), το Hyper Terminal του λειτουργικού συστήματος Windows.

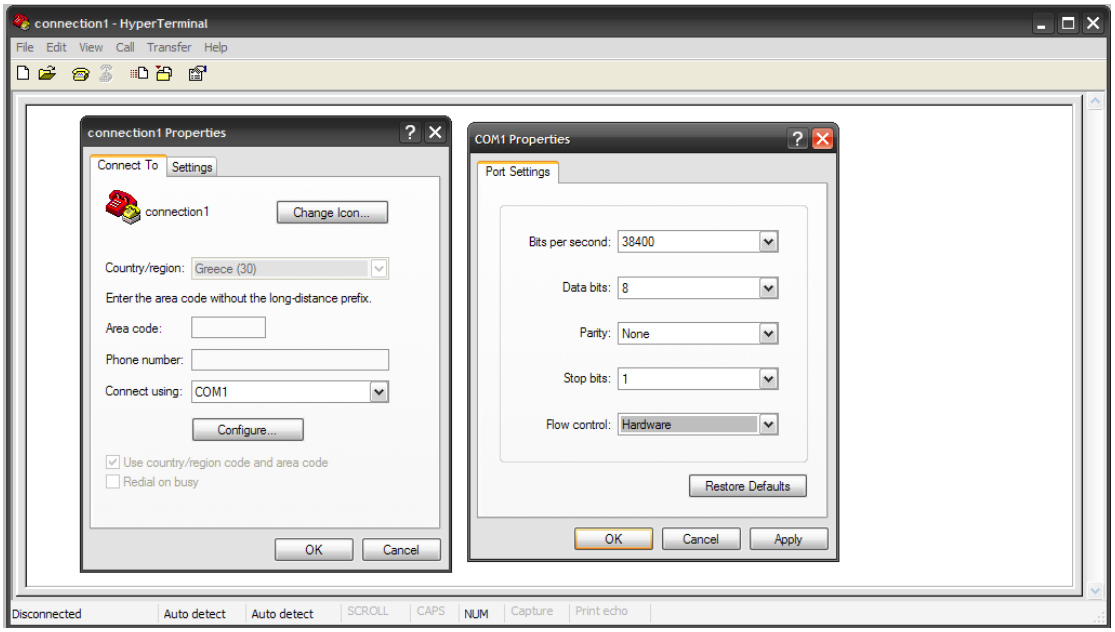
Διευθέτηση Παραμέτρων Επικοινωνίας Hyper Terminal με Συσκευή FPGA.

Για να υπάρξει επικοινωνία της συσκευής FPGA με τον Η/Υ θα πρέπει πρώτα να διευθετηθούν κάποιες παράμετροι με το Hyper Terminal το οποίο θα καθορίσει της ιδιότητες της επικοινωνίας μέσου του προτύπου RS232. Στην Εικόνα 4.4 φαίνεται η δημιουργία νέας σύνδεσης με όνομα **connection1**.



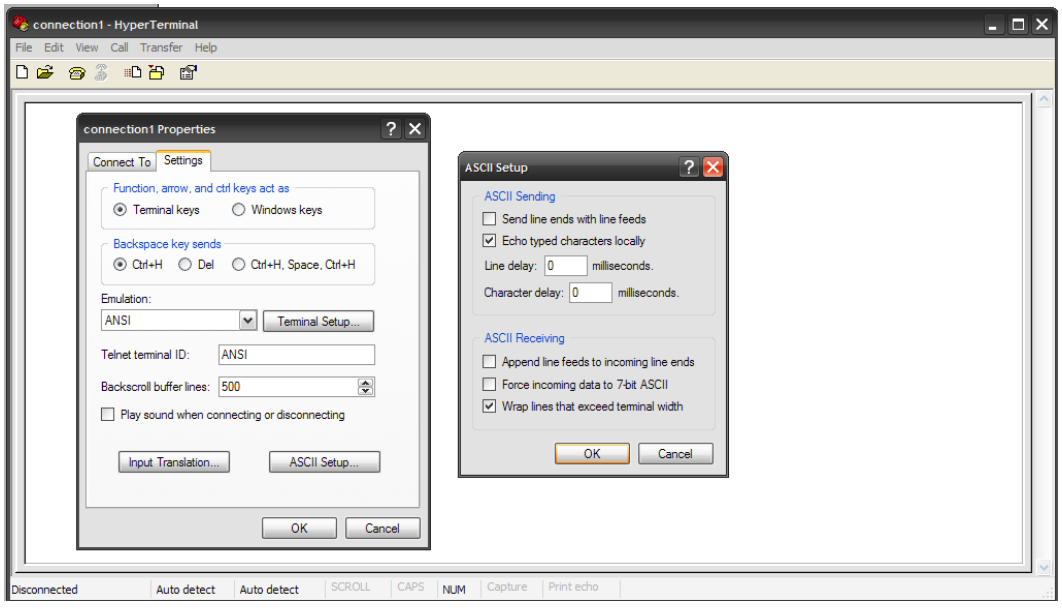
Εικόνα 4.4: Δημιουργία νέας σύνδεσης ονόματι connection1

Στην Εικόνα 4.5 φαίνονται οι ιδιότητες της επικοινωνίας που θα λάβει χώρα. Όπως μπορεί κανείς να παρατήρηση υπάρχει η δυνατότητα τροποποίησης των ιδιοτήτων. Μια σημαντική ιδιότητα είναι η αποστολή και λήψη των *bit ανά δευτερόλεπτο* (*bit per second*). Αυτή η παράμετρος πρέπει να καθοριστεί και στο πρωτόκολλο επικοινωνίας UART (Ενότητα 4.4.2), για να μπορέσει η συσκευή FPGA να συγχρονιστεί, ώστε να λαμβάνει και να στέλνει στο σωστό χρόνο τα bit.



Εικόνα 4.5: Ιδιότητες επικοινωνίας connection1.

Παράλληλα θα πρέπει να καθοριστεί ο κώδικας χαρακτήρων που χρειάζεται για να γίνει η επικοινωνία., στην Εικόνα 4.6 φαίνεται ότι έχει επιλεγεί ANSI ASCII κώδικας. Αυτός ο κώδικας καθορίζει τη *μορφή* (*format*) των χαρακτήρων που θα στέλνονται. Είναι πολύ σημαντική και αυτή η παράμετρος μιας και θα πρέπει στον κώδικα που θα υλοποιηθεί η επερώτηση, στις συγκρίσεις, να υπολογίζεται αυτός ο κωδικοποιημένος αριθμός. Συγκεκριμένα στον κώδικα της γραμμής 61 της επερώτησης Q1 (Εικόνα 3.20), αντί να γραφεί ο αριθμός 2 ως "00000010" θα πρέπει να γραφεί ως "00110010" το οποίο είναι και η πρότυπη μορφή του αριθμού 2 στον ASCII κώδικα (Εικόνα Α.1 Παράρτημα Α), θα πρέπει δηλαδή να γίνει όπως φαίνεται στη γραμμή 61 της Εικόνας 4.7.



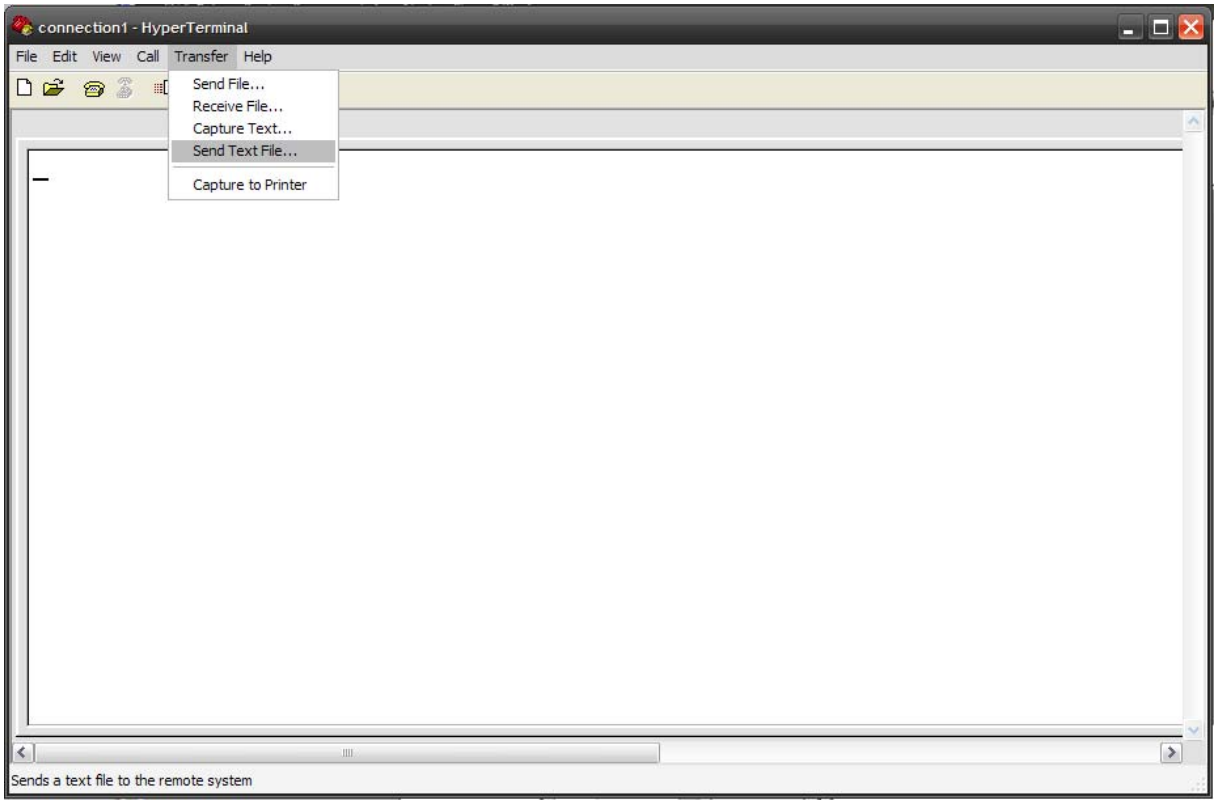
Εικόνα 4.6: Επιλογή ANSI ASCII κώδικα για την επικοινωνία

```
57  ------------------------------------------- Q1 ----------------
58  Q1: process (clk)                    -- start process Q1
59     begin
60     if clk'event and clk='1' then     -- synchronization whit clock
61        if (A3 > "00110010") then      -- the control element A3 > 2
62           out_data <= s_in_data;         -- output correct data
63           valid_bit <= '1' ;             -- valid bit = '1'
64        else
65           out_data <= (others =>'0'); -- output null data
66           valid_bit <= '0' ;             -- valid bit = '0'
67           end if;
68     end if;
69  end process Q1;                         --end process Q1
```

Εικόνα 4.7: Τροποποιημένη σύγκριση του αριθμού 2 με βάση τη μορφή του στον ASCII κώδικα

Αποστολή Αρχείου *.txt με το Hyper Terminal

Αφού διευθετηθούν όλες οι λεπτομέρειες θα πρέπει τώρα να σταλεί το αρχείο *.txt από τη σύνδεση connection1 του Hyper Terminal. Στην Εικόνα 4.8 φαίνεται η επιλογή *Transfer → Send Text File* με την οποία μπορεί κανείς να στείλει ένα αρχείο *.txt στη θύρα COM1. Κατόπιν στο νέο παράθυρο διαλόγου που θα εμφανιστεί θα επιλεγεί το αρχείο αποστολής, έστω το R.txt.

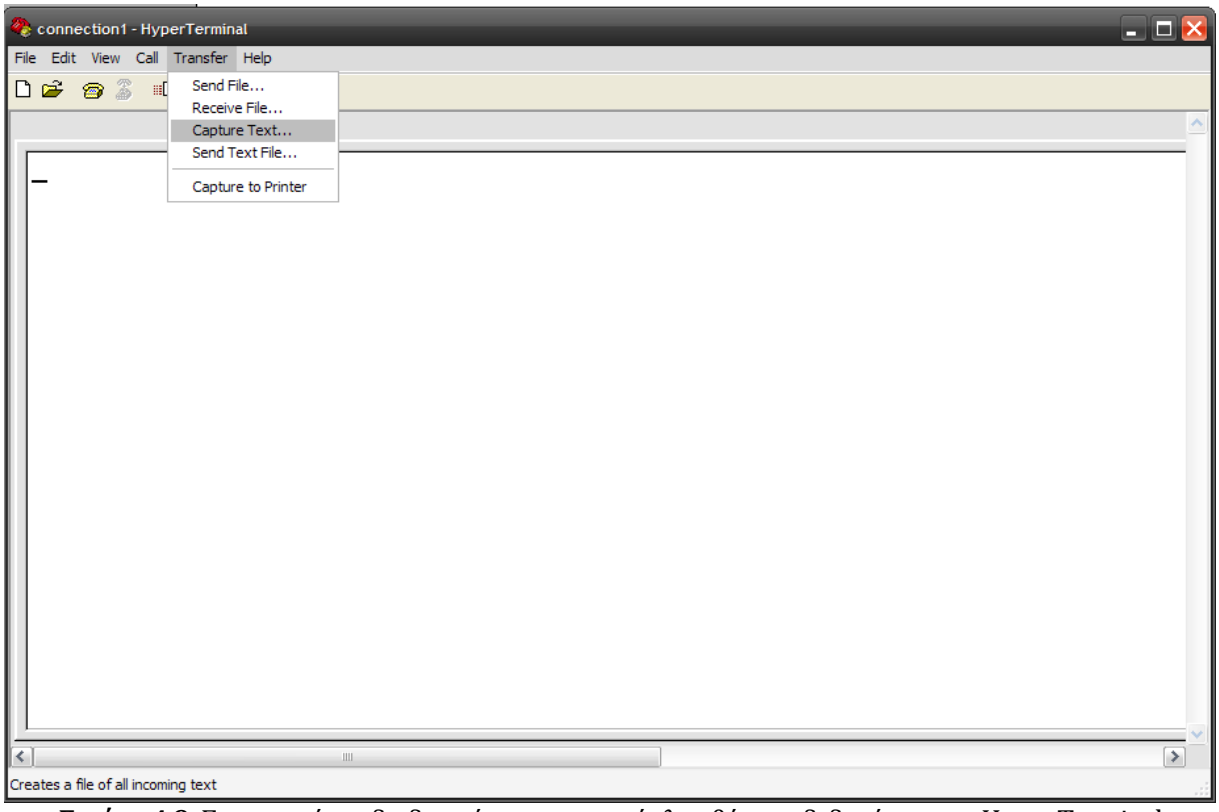


Εικόνα 4.8: Επιλογή Transfer → Send Text File για αποστολή αρχείου *.txt στη θύρα COM1

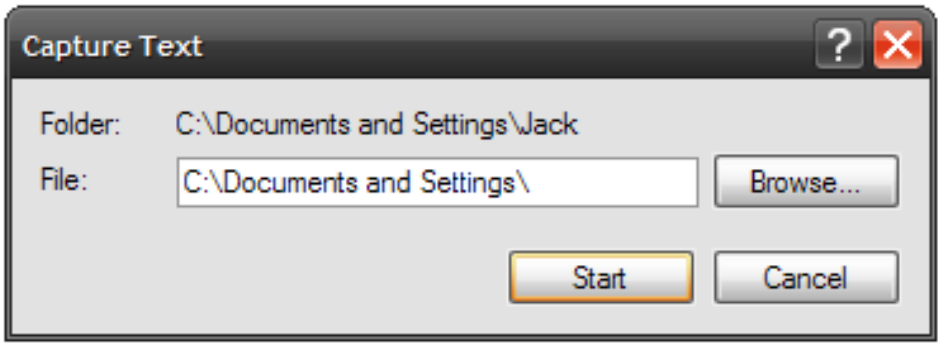
Λήψη Αρχείου *.txt με το Hyper Terminal

Για να λάβει κανείς τα δεδομένα που φτάνουν από τη θύρα COM1 αρκεί να ενεργοποιήσει τη λήψη δεδομένων από το Hyper Terminal. Τα δεδομένα αυτά αποθηκεύονται σε ένα αρχείο *.txt το οποίο και καθορίζεται. Στην Εικόνα 4.9 φαίνεται πως ενεργοποιείται αυτή η διαδικασία, γίνεται η επιλογή *Transfer* → *Capture Text,* με την οποία μπορεί κανείς να στείλει σε ένα αρχείο *.txt τα αποτελέσματα που λαμβάνονται από τη θύρα COM1. Και στην Εικόνα 4.10 καθορίζεται το μονοπάτι και το όνομα του αρχείου *.txt αυτού.

Σημειώνεται ότι το αρχείο *.txt το οποίο θα λαμβάνει τα δεδομένα από την θύρα COM1 θα πρέπει πρώτα να έχει δημιουργηθεί (ανεξάρτητα), δεν δημιουργείται από το Hyper Terminal και τη διαδικασία που προαναφέρθηκε.



Εικόνα 4.9: Ενεργοποίηση διαδικασίας καταγραφής ληφθέντων δεδομένων στο Hyper Terminal



Εικόνα 4.10: Διαδικασία καθορισμού μονοπατιού και ονόματος αρχείου *.txt για τη λήψη δεδομένων από τη θύρα COM1

Με τις παραπάνω ενέργειες, ορίστηκε ο τρόπος ανάγνωσης και εγγραφής ενός αρχείου *.txt και κατά την πραγματική εκτέλεση στο υλικού FPGA.

4.2 Memory CAM (Contact Addressable Memory)

Η *διευθυνσιοδοτούμενη μνήμη περιεχομένου CAM* (*Content Addressable Memory*) αποτελεί μια ιδιαίτερη μνήμη διότι επιτελεί την αντίστροφη λειτουργία από αυτή που επιτελούν οι παραδοσιακές μνήμες. Συγκεκριμένα το στοιχείο αναζήτησης μέσα στη μνήμη δεν είναι η διεύθυνση αλλά το περιεχόμενο της, δηλαδή στέλνονται τα δεδομένα στη μνήμη CAM και επιστρέφεται η διεύθυνση στην οποία βρίσκονται. Ο χρόνος απόκρισης της είναι της τάξεως του **ενός** κύκλου ρολογιού (!) για κάθε προσπέλαση. Αυτή η ιδιότητα έχει κάνει τις μνήμες CAM ιδιαίτερα δημοφιλείς διότι σε συνδυασμό με μια μνήμη RAM – ROM, ο χρόνος απόκρισης για την εύρεση δεδομένων μέσα στη μνήμη RAM – ROM μπορεί να φτάνει τους **δυο** με **τρεις** κύκλους ρολογιού. Στην παραδοσιακή αναζήτηση περιεχομένων μέσα σε μια μνήμη RAM – ROM, η χειρότερη περίπτωση έφτανε τους **n** κύκλους ρολογιού, όσο ουσιαστικά ήταν το μέγεθος μιας μνήμης RAM – ROM. Οπότε αυτός ο συνδυασμός, μνήμης CAM με μία μνήμη RAM – ROM, δίνει το επιθυμητό αποτέλεσμα πολύ γρήγορα. Αυτήν ακριβώς την ιδιότητα εκμεταλλεύεται και η σχεδίαση των λογικών κυκλωμάτων που υλοποιούν τις επερωτήσεις Q3 (Ενότητα 5.2.3) και Q4 (Ενότητα 5.2.4).

Στην παρούσα ενότητα παρουσιάζονται και κάποια άλλα χαρακτηριστικά και γνωρίσματα που πρέπει να ξέρει ο αναγνώστης ώστε να κατανοήσει καλύτερα τη σχεδίαση των λογικών κυκλωμάτων που υλοποιούν τις επερωτήσεις Q3 και Q4. Συγκεκριμένα στην Ενότητα 4.2.1 θα παρουσιαστούν μερικές ιδιότητες της μνήμης και ο τρόπος αρχικοποίησης της μνήμης. Και στην

Ενότητα 4.2.2 θα παρουσιαστεί ο αλγόριθμος CAM_ARRAY ο οποίος υλοποιήθηκε από την ερευνητική ομάδα για την απόδοση καλύτερης λύσης στη σωστή κατανομή στοιχείων σε ανεξάρτητες μνήμες CAM.

Οι μνήμη CAM που θα παρουσιαστεί είναι από τη βιβλιοθήκη του εργαλείου σχεδίασης CAD ISE [32,33] και αποτελεί στοιχείο και της συσκευής FPGA Spartan - 3E [35,36].

4.2.1 Ιδιότητες Μνήμη CAM και ο Τρόπος Αρχικοποίησης της

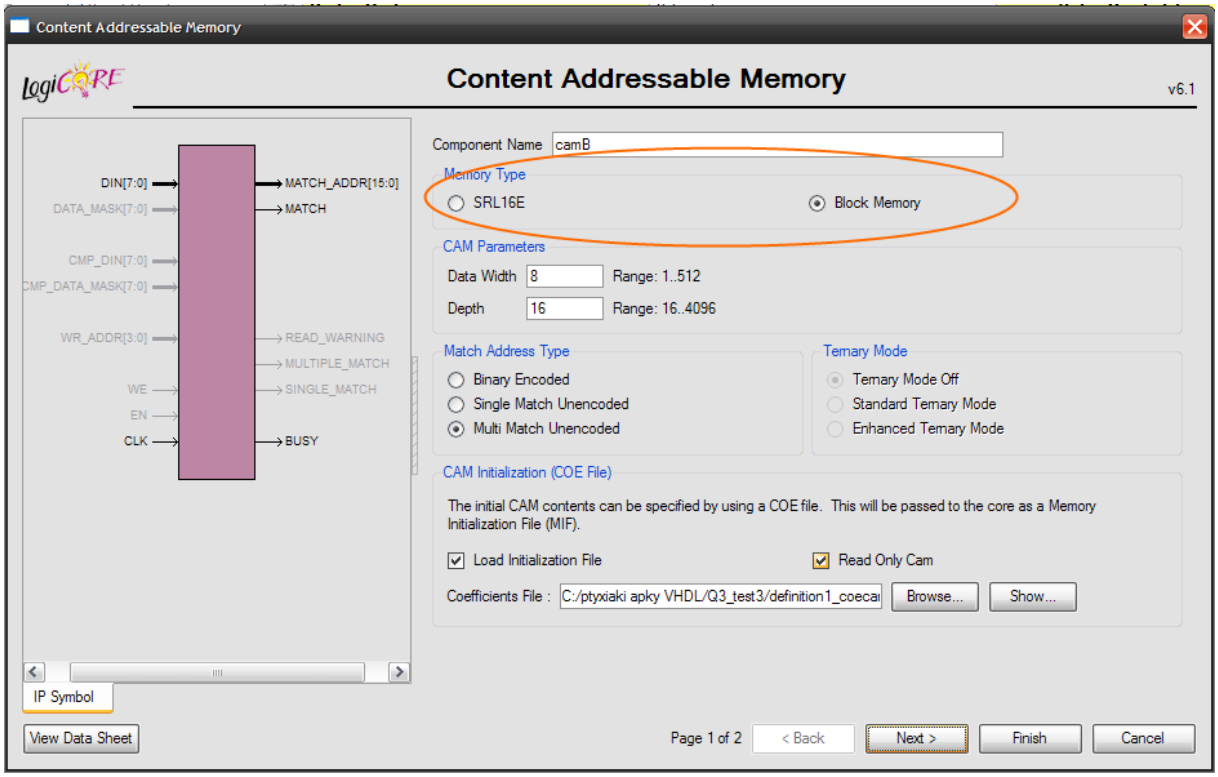
Τύπος Μνήμης SLR16E και Block Memory

Μία από τις ιδιότητες της μνήμης CAM είναι ότι μπορεί να υλοποιηθεί με δύο διαφορετικούς τρόπους στις συσκευές FPGA (άλλες αρχιτεκτονική περιλαμβάνουν μόνο μία ή και καμία). Συγκεκριμένα μπορεί να υλοποιηθεί μια μνήμη CAM:

1. είτε μέσω του τύπου *SLR16E*, η οποία χρειάζεται 16 κύκλους ρολογιού για να κάνει εγγραφή και έναν κύκλο για να κάνει ανάγνωση,

2. είτε μέσω του τύπου *Block Memory*, η οποία χρειάζεται δύο κύκλους για να κάνει εγγραφή και έναν κύκλο για να κάνει ανάγνωση.

Και στις δύο περιπτώσεις δίνεται η δυνατότητα ταυτόχρονης προσπέλασης (εγγραφής – ανάγνωσης). Αν έχει ενεργοποιηθεί αυτή η δυνατότητα τότε η ανάγνωση έπεται της εγγραφής, δηλαδή γίνεται πρώτα εγγραφή και μετά η ανάγνωση της. Αυτό το χαρακτηριστικό πρέπει να υπολογίζεται από το σχεδιαστή ώστε να προσαρμόσει την κατάλληλη μνήμη. Στις υλοποιήσεις που έγιναν για τις δύο επερωτήσεις Q3 και Q4, χρησιμοποιήθηκε μόνο η ανάγνωση των μνημών και ο τύπος Block Memory. Στην Εικόνα 4.11 στο πεδίο *Memory Type* φαίνεται ο τύπος μνήμης που επιλέχθηκε.



Εικόνα 4.11: Τύπος μνήμης Block Memory επερωτήσεων Q3 και Q4, στο εργαλειο σχεδιασης Xilinx ISE

Μέγεθος Μνήμης

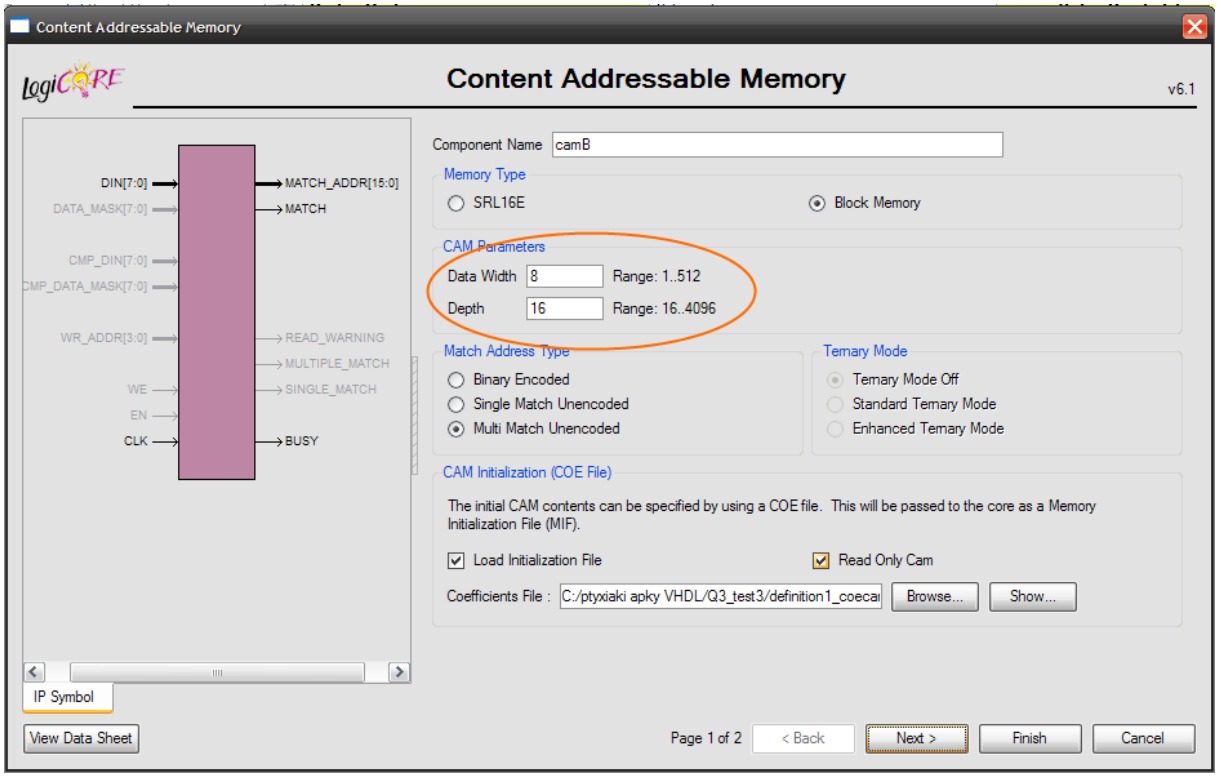
Μία δεύτερη ιδιότητα είναι το μέγεθος της μνήμης, το οποίο εξαρτάται από τη συγκεκριμένη αρχιτεκτονική δομή της συσκευής FPGA.

Συγκεκριμένα πρέπει να αρχικοποιηθούν οι παράμετροι :

1. *Data Width*, το οποίο είναι το μέγεθος των δεδομένων μέσα στη μνήμη και μπορεί να λαμβάνει τιμές από 1 bit μέχρι 512 bit.

2. *Depth*, το οποίο είναι το μέγεθος των διευθύνσεων στη μνήμη, δηλαδή το βάθος της μνήμης. Οι τιμές που μπορεί να λάβει είναι από 16 έως 4096.

Στις υλοποιήσεις που έγιναν για τις δύο επερωτήσεις Q3 και Q4, χρησιμοποιήθηκε το μέγεθος των δεδομένων *Data Width* ίσο με 8 και το μέγεθος των διευθύνσεων *Depth* ίσο με 16. Στην Εικόνα 4.12 στο πεδίο *Cam Parameters* φαίνονται οι δηλώσεις που επιλέχθηκαν.



Εικόνα 4.12: Διευθέτηση μεγέθους μνήμης CAM στα πεδία Data Width και Depth, στο εργαλειο σχεδιασης Xilinx ISE

Τύπος Παραγόμενης Διεύθυνσης

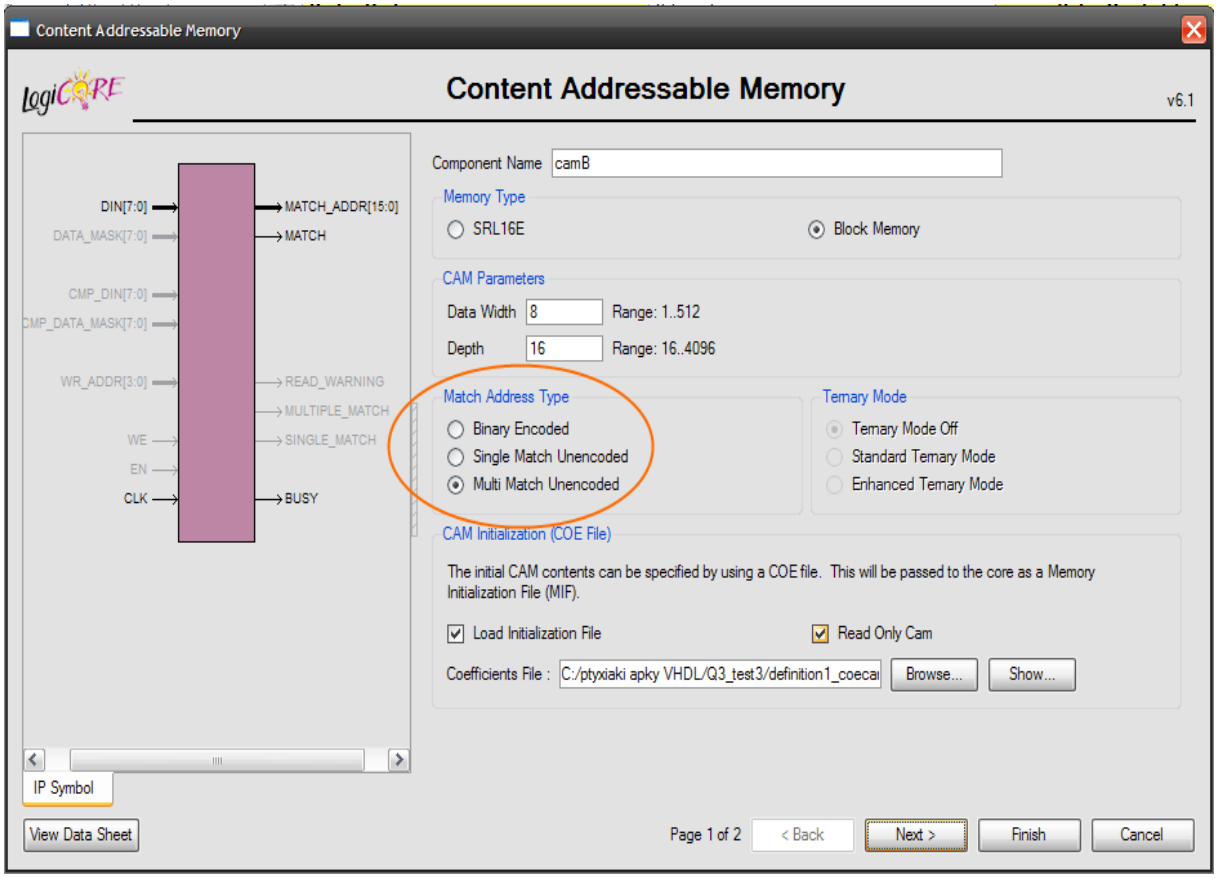
Η διεύθυνση που παράγεται αφού βρεθούν τα δεδομένα θα πρέπει να είναι κάποιας μορφής, στη δήλωση της μνήμης CAM υπάρχουν τρεις επιλογές, αυτές είναι :

1. *Binary Encoded*, στην οποία η παραγόμενη διεύθυνση κωδικοποιείται απευθείας και έχει κωδικοποιημένο μέγεθος το οποίο εξαρτάται από το μέγεθος που δηλώθηκε στο πεδίο *Depth*. Στην περίπτωση των 16 θέσεων η κωδικοποιημένη διεύθυνση είναι των 4 bit.

2. *Single Match Encoded*, και εδώ το μέγεθος της διεύθυνση εξαρτάται από το μέγεθος που δηλώθηκε στο πεδίο *Depth*. Στην περίπτωση των 16 θέσεων η διεύθυνση είναι των 16 bit. Όταν δηλωθεί αυτή η παράμετρος τότε θα επιστραφεί μια διεύθυνση που θα περιλαμβάνει το ένα bit ενεργοποιημένο από τα 16, από αυτά που έχουν ενεργοποιηθεί ως *επιτυχής εύρεση* (hit), ενώ τα υπόλοιπα θα είναι 0. Αν σε περίπτωση δεν βρεθούν τα δεδομένα στη μνήμη τότε αυτή η επιλογή θα επιστρέψει 16 μηδενικά. Ουσιαστικά επιστρέφει την μια από τις ευρέσεις που βρέθηκαν.

3. *Multi Match Encoded*, κάνει ακριβώς την ίδια διαδικασία με την Single Match Encoded αλλά αυτή επιστρέφει στη διεύθυνση, όλες τις *επιτυχής ευρέσεις* (hit) που έχουν βρεθεί και όχι την μία από αυτές.

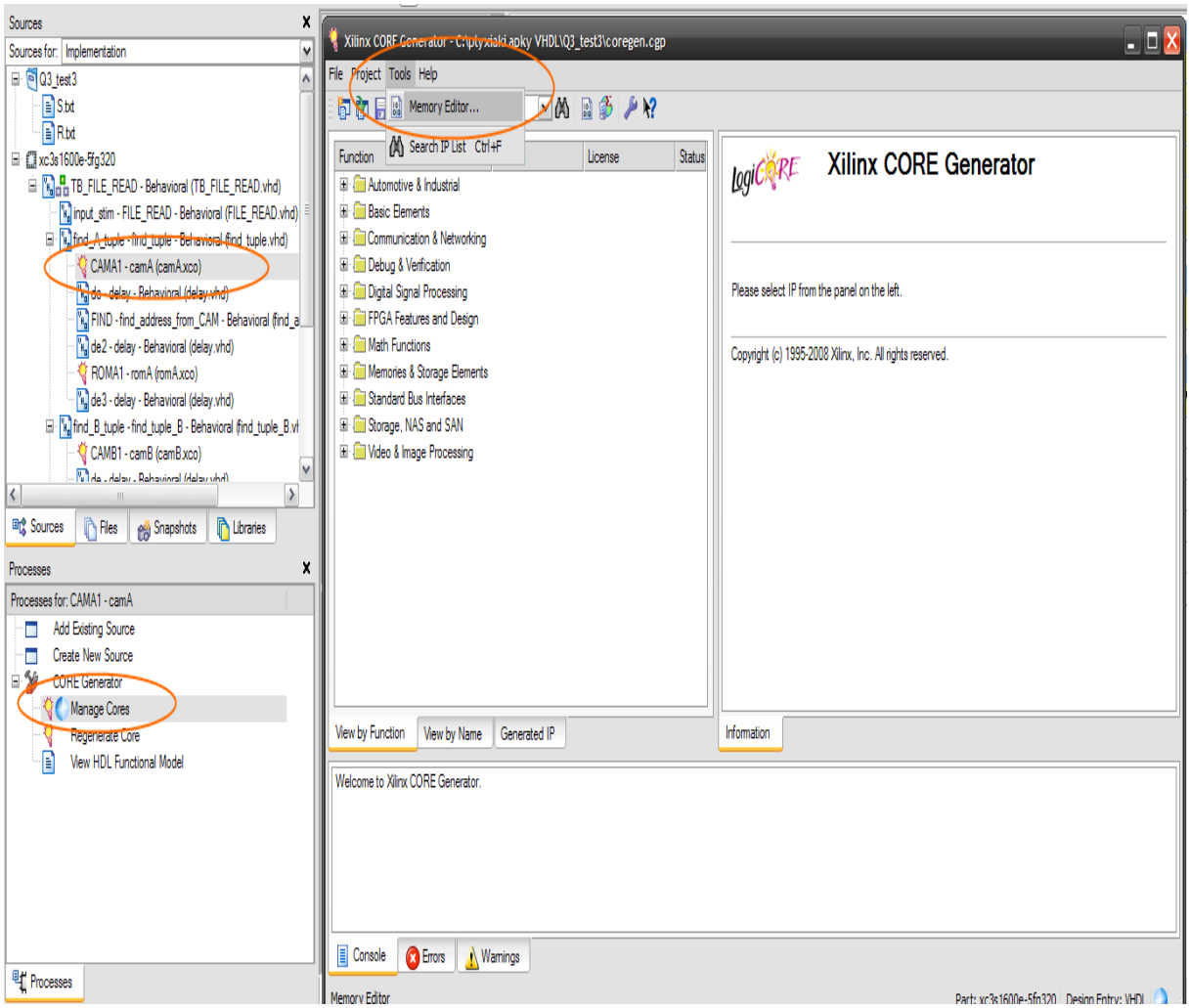
Στις υλοποιήσεις που έγιναν για τις δύο επερωτήσεις Q3 και Q4, χρησιμοποιήθηκε η τρίτη επιλογή. Μπορεί κανείς να αναρωτηθεί με πιο κριτήριο έγινε η επιλογή της πολλαπλής εμφάνισης της διεύθυνσης, η απάντηση βρίσκεται στην ίδια τη λύση της. Όπως έγινε η σχεδίαση δεν επηρεάζει στο αποτέλεσμα η πολλαπλή εμφάνιση. Θα μπορούσε να χρησιμοποιηθεί και η δεύτερη επιλογή αλλά επιλέχθηκε η τρίτη για να προσέξει ο αναγνώστης ότι δεν παίζει ρόλο μόνο το τι προσφέρει το εργαλείο σχεδίασης ISE, αλλά να δει περισσότερο και την ικανότητα του σχεδιαστή να χρησιμοποιεί τη σωστή δομή μέσα στο λογικό κύκλωμα που υλοποιεί. Στην Εικόνα 4.13 στο πεδίο *Mach Address Type* φαίνεται η επιλογή που έγινε.



Εικόνα 4.13: Επιλογή τύπος παραγόμενης διεύθυνσης από το πεδίο Mach Address Type, στο εργαλειο σχεδιασης Xilinx ISE

Αρχικοποίηση Μνήμης CAM

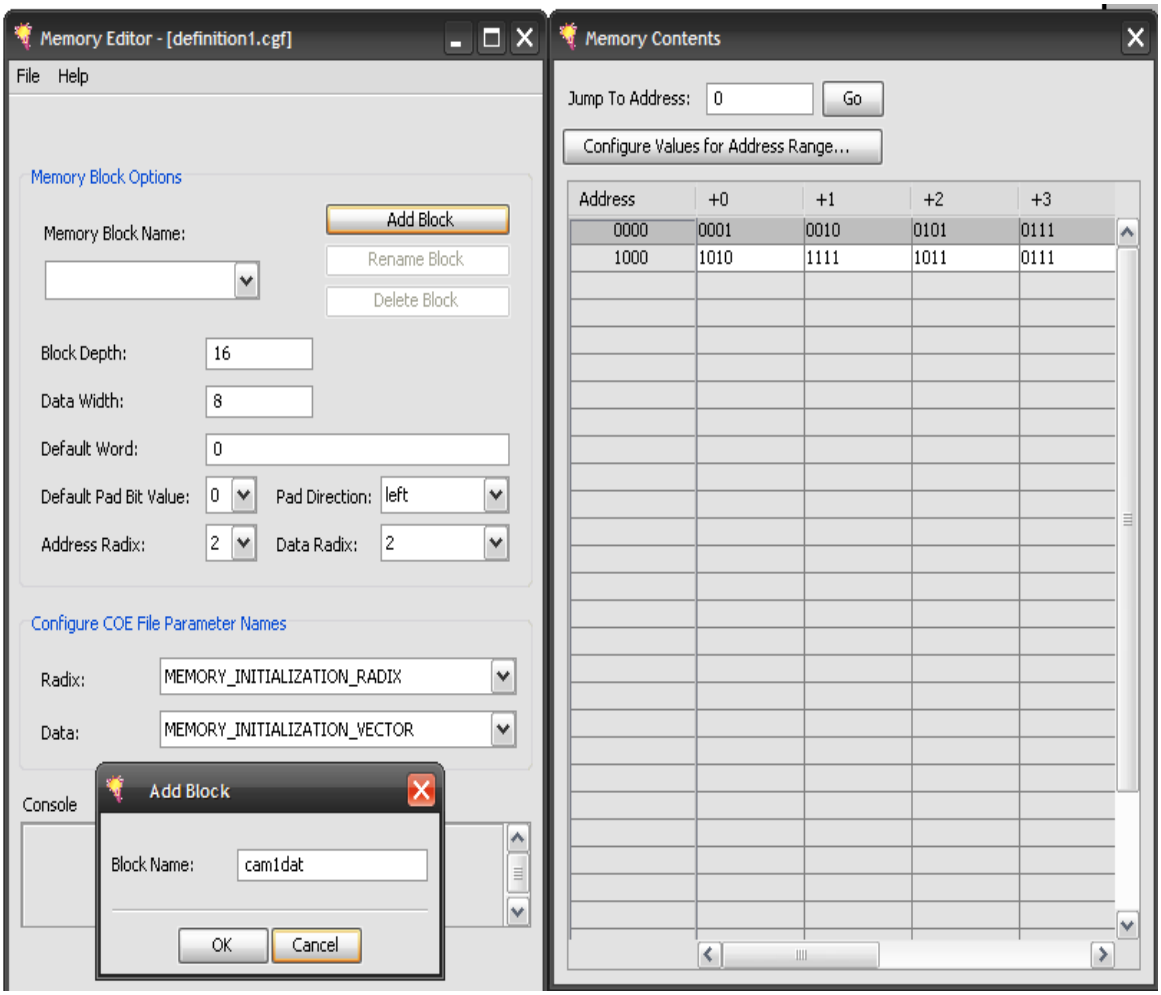
Η διαδικασία αρχικοποίησης της μνήμης CAM προαπαιτεί κάποιες αρχικές ενέργειες που πρέπει να γίνουν. Αρχικά πρέπει να παταχθεί το αρχείο ***.coe** με το οποίο μπορεί κανείς να αρχικοποιήσει μνήμες CAM, RAM, και ROM μεγέθους παρόμοιο με αυτή που σχεδιάστηκε. Για να γίνει αυτό πρέπει πρώτα να επιλεγεί και να φορτωθεί μια βοηθητική λειτουργία η *Xilinx Core Generator* με την οποία ο σχεδιαστής μπορεί να δημιουργήσει έτοιμα λογικά κυκλώματα που υποστηρίζονται όμως μόνο από στο εργαλείο ISE και όχι από κάποιο άλλο σχεδιαστικό εργαλείο CAD. Με αυτήν τη λειτουργία ο σχεδιαστής μπορεί να δημιουργήσει τα αρχεία *.coe με τα οποία θα γίνει η αρχικοποίηση των μνημών CAM. Για να γίνει αυτό πρέπει πρώτα να επιλεγεί μια μνήμη CAM από το παράθυρο *Sources* του *Project Navigator* (Εικόνα Δ.2 Παράρτημα Δ) του ISE. Κατόπιν από το παράθυρο *Processes* διπλό click στο *Manage Cores*, θα ανοίξει ένα νέο παράθυρο διαλόγου το *Xilinx Core Generator* όπου ο σχεδιαστής θα επιλέξει το *Tools* → *Memory Editor*. Στην Εικόνα 4.14 φαίνεται ο τρόπος ανοίγματος της λειτουργίας Xilinx Core Generator.



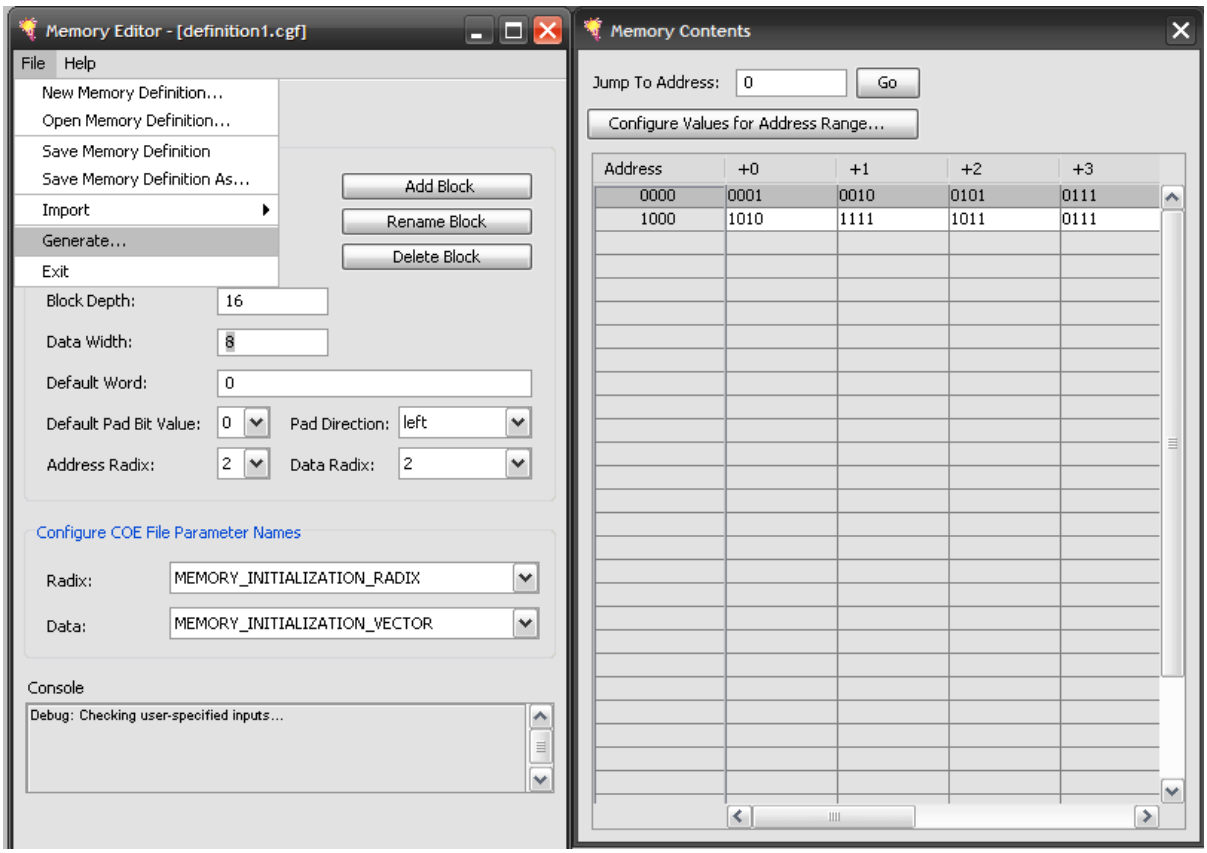
Εικόνα 4.14: Άνοιγμα λειτουργίας Core Generator στο εργαλείο σχεδιασης Xilinx ISE.

Μετά θα ανοίξουν δύο νέα παράθυρα διαλόγου του Memory Editor, στο ένα θα μπορεί ο σχεδιαστής να καθορίσει τις παραμέτρους όπως βάθος μνήμης (*Block Depth*), μέγεθος δεδομένων (*Data Width*), όνομα αρχείου (*Add Block*), εμφάνιση μορφής διεύθυνσης μνήμης (*Address Radix*) και μορφή δεδομένων (*Data Radix*) κτλ. Στο άλλο παράθυρο θα δημιουργήσει τα περιεχόμενα της μνήμης (*Memory Contents*), στην Εικόνα 4.15 φαίνεται αυτή η διαδικασία.

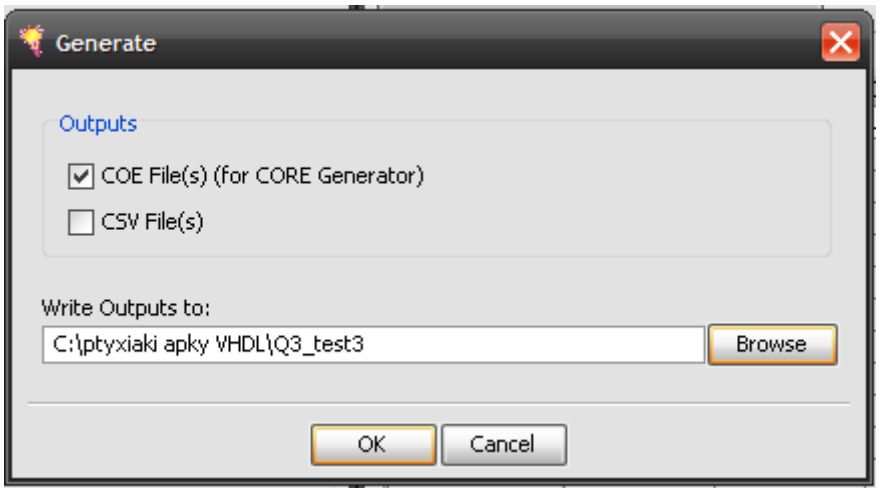
Αφού γίνει η δήλωση θα πρέπει να γίνει η επιλογή της δημιουργίας από το μενού *File* → *Generate* (Εικόνα 4.16). Όταν γίνει αυτό θα ανοίξει ένα νέο παράθυρο διαλόγου όπου και θα καθοριστεί ο τύπος αλλά και ο φάκελος στον οποίο θα δημιουργηθεί το αρχείο *coe (Εικόνα 4.17).



Εικόνα 4.15: Εισαγωγή τιμών στη μνήμη και διευθέτηση παραμέτρων για τη δημιουργία αρχείου αρχικοποίησης μνημών *.coe από το Memory Editor, (οι τιμές και το όνομα είναι ενδεικτικά), από το εργαλειο σχεδιασης Xilinx ISE



Εικόνα 4.16: Δημιουργία αρχείου *.coe από το Memory Editor, από το εργαλειο σχεδιασης Xilinx ISE



Εικόνα 4.17: Καθορισμός τελευταίων παραμέτρων πριν τη δημιουργία του αρχείου *.coe από το Memory Editor, από το εργαλειο σχεδιασης Xilinx ISE

Ένα αρχείο *.coe έχει τη μορφή που φαίνεται στην Εικόνα 4.18, το συγκεκριμένο αρχείο *definition1_coecama.coe* χρησίμευσε στην αρχικοποίηση της μνήμης camA (Ενότητα 5.2.3) του λογικού κυκλώματος που δίνει λύση στην επερώτηση Q3. Όπως μπορεί κανείς να παρατηρήσει τα δεδομένα είναι σε δυαδική μορφή.

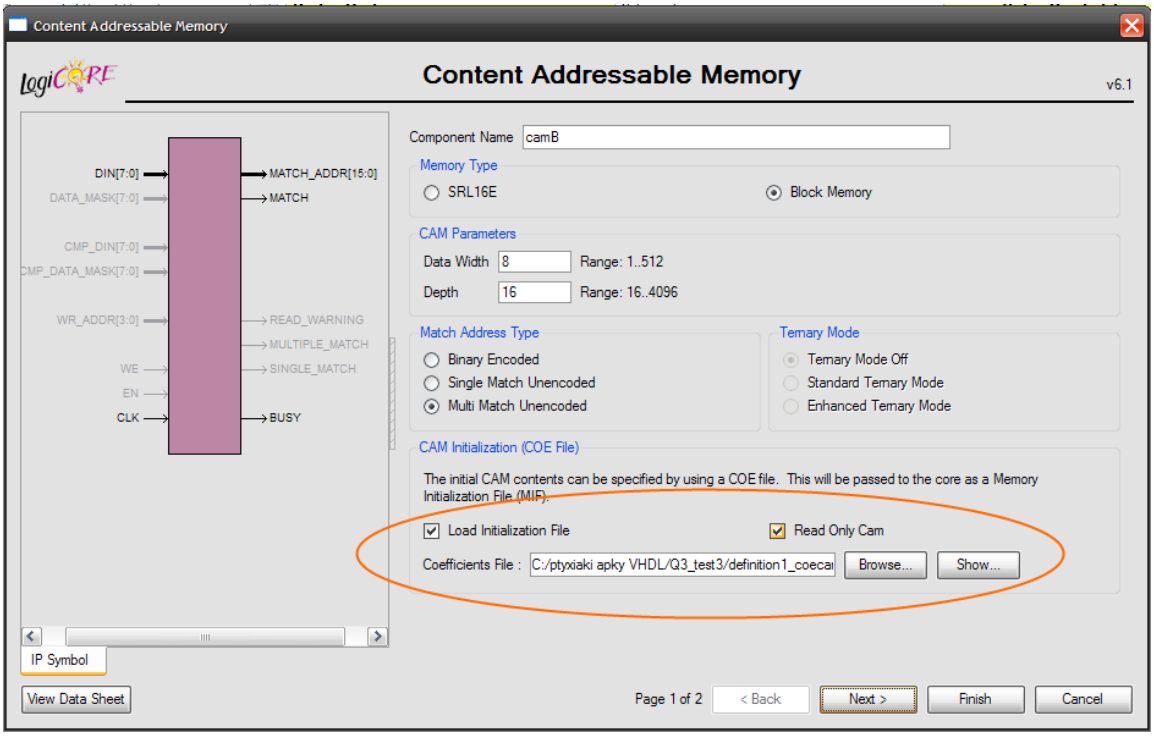
definition1_coecama.coe

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
00000100,
00000001,
00000101,
00000010,
00000111,
00000011,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000,
00000000;
```

Εικόνα 4.18: Περιεχόμενα αρχείου αρχικοποίησης definition1_coecama.coe της μνήμης camA της επερώτησης Q3

Η διαδικασία που παρουσιάστηκε παραπάνω είναι για να δημιουργηθεί το αρχείο αρχικοποίησης μνημών CAM και όχι για να γίνει η αρχικοποίηση της μνήμης CAM.

Για να αρχικοποιηθεί μια μνήμη CAM θα πρέπει να ενεργοποιηθεί η επιλογή *Load Initialization File* και αμέσως μετά να βρεθεί το αρχείο *.coe με το οποίο θα αρχικοποιηθεί η μνήμη με τα δεδομένα αρχικοποίησης. Στην Εικόνα 4.19 φαίνεται αυτή η διαδικασία επιλογής, συγκεκριμένα γίνεται αρχικοποίηση της μνήμης camA από το αρχείο αρχικοποίησης definition1_coecama.coe.



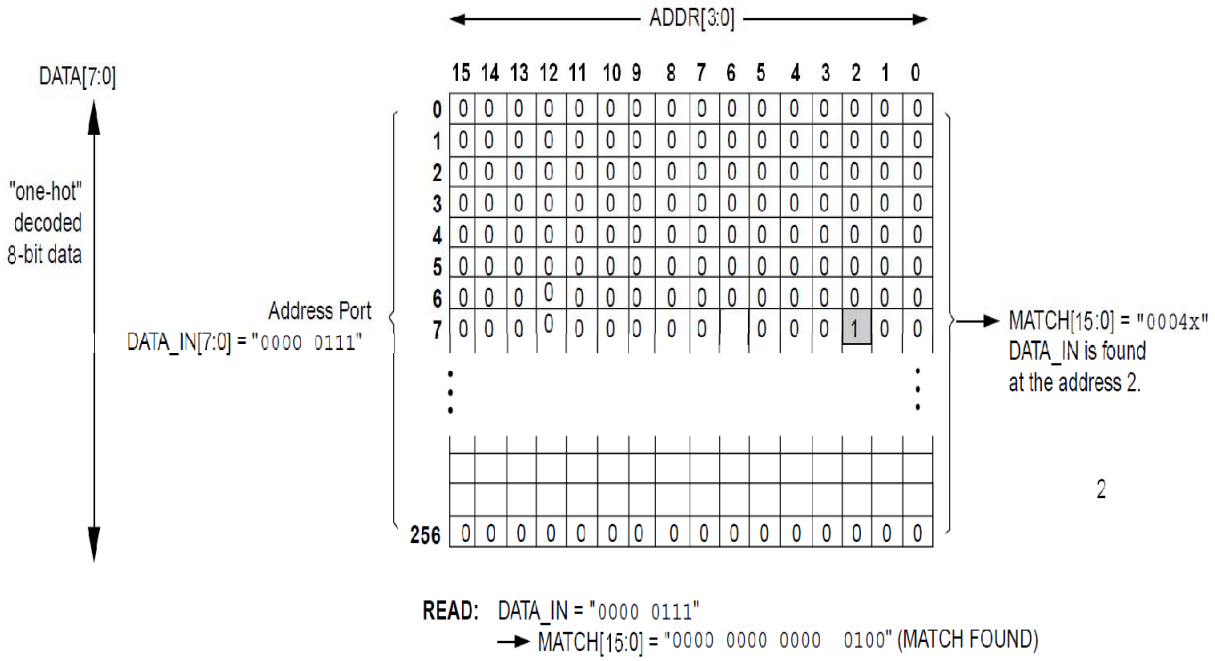
Εικόνα 4.19: Διαδικασία επιλογής για την αρχικοποίηση της μνήμης camA από το αρχείο αρχικοποίησης definition1_coecama.coe, από το εργαλειο σχεδιασης Xilinx ISE

Όπως μπορεί να διακρίνει κανείς, πέραν της αρχικοποίησης έγινε και επιλογή της μνήμης CAM να είναι και *μόνο ανάγνωσης* (*Read only CAM*).

Εύρεσης Δεδομένων

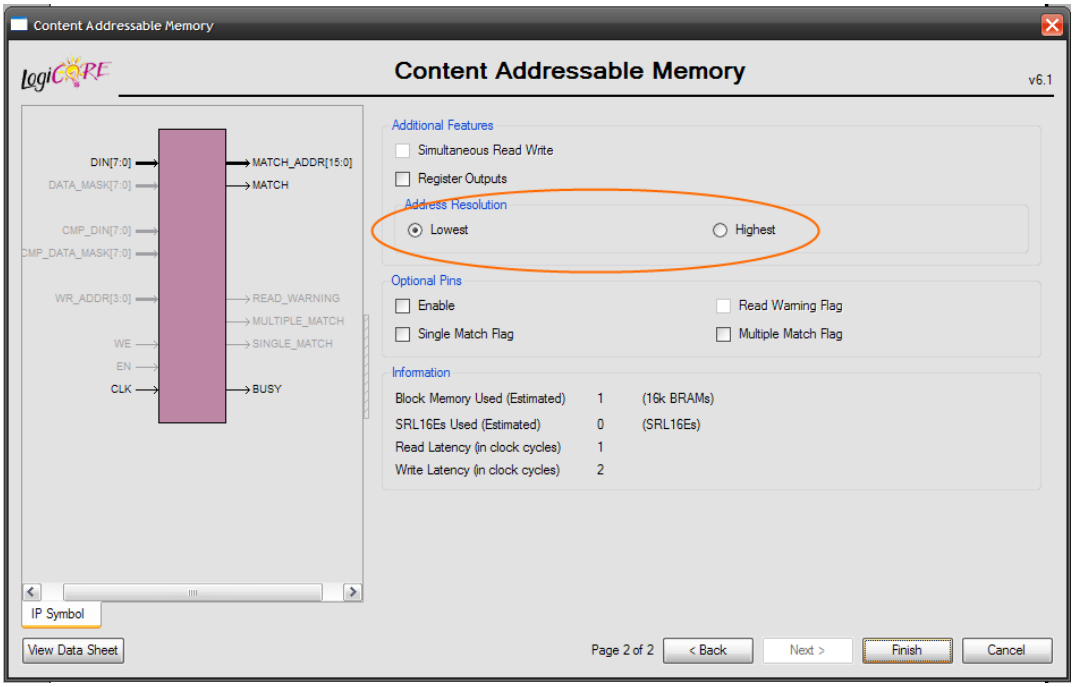
Μια τελευταία ιδιότητα της μνήμης CAM που πρέπει να αναφερθεί είναι αυτή της εμφάνισης, της *διεύθυνσης επιτυχής εύρεσης* (*Match_Address*). Όπως προαναφέρθηκε υπάρχει η δυνατότητα της πολλαπλής παρουσίασης των πετυχημένων διευθύνσεων(*Mach Address Type*), αλλά και της μονής παρουσίασης πετυχημένων διευθύνσεων (*Single Match Encoded*). Για να μπορέσει ο αναγνώστης να καταλάβει πως γίνεται η εύρεση αλλά και η κωδικοποίηση της διεύθυνσης θα παρουσιαστεί ένα παράδειγμα. Για να γίνει αυτό θα αναφερθούν λίγο τα χαρακτηριστικά της μνήμης.

Έστω ότι υπάρχει μνήμη CAM με μέγεθος δεδομένων (*Data Width*) ίσο με 8 bit και μέγεθος της διεύθυνση (*Depth*) ίσο με 16. Τα 8 bit των δεδομένων μπορούν να μεταφραστούν και ως 256 πιθανοί συνδυασμοί, δηλαδή σε μια ακολουθία 256 bit μπορεί να καλυφθούν όλοι οι συνδυασμοί ενός 8bit αριθμού. Πχ ο αριθμός 7 μπορεί να μεταφραστεί σε δυαδική μορφή του 8bit καταχωρητή ως “00000100”, στη μορφή όμως των 256bit θα ήταν ο “0000……0010000000” . Όλοι οι αριθμοί θα είναι μηδέν και μόνο στη σωστή θέση θα είναι ο μοναδικός άσσος. Ως διεύθυνση θα βγει αυτή η κωδικοποιημένη σειρά (ή η 16bit) που θα εμφανίζει σε ποια θέση βρέθηκε. Η πραγματική δομή των περιεχομένων της μνήμης CAM φαίνεται στην Εικόνα 4.20.



Εικόνα 4.20: Δομή περιεχομένων μνήμης CAM και ο τρόπος εύρεσης στοιχείου

Αν υπάρχουν περισσότερες εμφανίσεις από μία, τότε έχει τη δυνατότητα ο σχεδιαστής να επιλέξει την υψηλότερη (*Highest*) ή τη χαμηλότερη (*Lowest*) διεύθυνση. Αυτή η δυνατότητα φαίνεται στην Εικόνα 4.21. Στην περίπτωση όμως που επιλεγεί η πολλαπλή εμφάνιση, η ποσότητα των άσσων θα είναι μεγαλύτερη.



Εικόνα 4.21: Επιλογή εμφάνισης υψηλότερης (*Highest*) η χαμηλότερης (*Lowest*) διεύθυνση στη μνήμη CAM, από το εργαλειο σχεδιασης Xilinx ISE

Για περισσότερη εμβάθυνση γύρω από τις μνήμες CAM καλείται ο αναγνώστης να ανατρέξει στην προτεινόμενη βιβλιογραφία [13,29,34].

4.2.2 Αλγόριθμος *CAM_ARRAY*

Ο αλγόριθμος αυτός είναι γραμμένεος σε γλώσσα προγραμματισμού ANSI C και υπολογίζει τον αριθμό των μνημών CAM που μπορούν να χρησιμοποιηθούν σε μια επερώτηση, όταν κάποια από τα γνώρισμα του πίνακα περιέχει περισσότερο από ένα ίδια στοιχεία. Συγκεκριμένα ο αλγόριθμος δέχεται έναν μονοδιάστατο πίνακα και παράγει αριθμό άλλων μονοδιάστατων πινάκων σε δυναμική μορφή. Η ποσότητα των πινάκων που μπορεί να παραχθούν εξαρτάται από τον αριθμών των εμφανίσεων ενός στοιχείο μέσα στον «πατρικό» πίνακα. Αν δηλαδή ο πατρικός πίνακας έχει ένα στοιχείο το οποίο εμφανίζεται 25 φορές τότε θα παράγει 25 διαφορετικούς πίνακες. Στη συνέχεια γεμίζει τους πίνακες «παιδία» με τα υπόλοιπα στοιχεία του πατρικού πίνακα. Η διαδικασία αυτή επαναλαμβάνεται έως ότου μηδενιστεί ο πατρικός πίνακας.

Ο αλγόριθμος αυτός διασφαλίζει την μη ύπαρξη ίδιου στοιχείου μέσα στους πίνακες παιδία. Δηλαδή σε κάθε παραγόμενο πίνακα δεν θα βρεθούν δύο η περισσότερα ίδια στοιχεία. Ο λόγος της δημιουργίας του αλγόριθμου, ήταν για να μπορέσει ο σχεδιαστής να έχει έναν ασφαλή και πιστοποιημένο τρόπο να παράγει τόσες μνήμες CAM (για αυτό και η ονομασία του) όσες πραγματικά χρειάζεται. Η χειροκίνητη και εμπειρική διαλογή στοιχείων βοηθάει όταν ο πίνακας είναι μικρός, τι γίνεται όταν ο έχει εκατοντάδες ή χιλιάδες στοιχεία. Τη λύση την δίνει ο αλγόριθμος CAM_ARRAY ο οποίος θα διασφαλίσει στο σχεδιαστή ότι δεν θα έχει η κάθε μνήμη δυο η περισσότερα ίδια στοιχεία. Ο αλγόριθμος αυτός δουλεύει τόσο για πολύ μικρούς πίνακες όσο και για πολύ μεγάλους, για του λόγου το αληθές μπορεί ο αναγνώστης να δει στην Ενότητα Α.2.2 στο Παράρτημα Α τα αποτελέσματα που παρήχθησαν σε μια δοκιμή που έγινε σε έναν πίνακα 1225 θέσεων.

Πάρα ταύτα ο αλγόριθμος μπορεί να δουλέψει και για άλλες εφαρμογές οι οποίες χρειάζονται διαχωρισμό στοιχείων ενός πίνακα και εισαγωγή τους σε διαφορετικούς άλλους.

Βήματα Αλγόριθμου

Τα βήματα και η δομή του αλγόριθμου (το διάγραμμα ροής του) φαίνεται στην Εικόνα 4.22.

Όπως μπορεί να παρατηρήσει κανείς, ο αλγόριθμος επιτελεί τις παρακάτω ενέργειες (βήματα):

1. *Αρχή*, αρχικοποίηση πίνακα εισαγωγής ο οποίος έχει ονομαστεί μέσα στον αλγόριθμο ως “V”.

2. *Ταξινόμηση πίνακα (Merge Short)* , σε αυτό το σημείο ο αλγόριθμος κάνει ταξινόμηση του πίνακα εισαγωγής V.

3. *Εμφάνιση ταξινομημένου πίνακα*, εμφανίζει με μια επανάληψη τα περιεχόμενα του ταξινομημένου πίνακα εισαγωγής V.

4. *Εύρεση του στοιχείου του πίνακα με τις περισσότερες εμφανίσεις*, βρίσκει αυτό το στοιχείο του (ταξινομημένο) πίνακα εισαγωγής V το οποίο έχει τις περισσότερες εμφανίσεις. Για παράδειγμα το στοιχείο 671 είχε τις περισσότερες εμφανίσεις, έξι στο σύνολο, στον (ταξινομημένο) πίνακα εισαγωγής V της εκτέλεσης του αλγόριθμου της Ενότητας Α.2.2 στο Παράρτημα Α.

Σημείωση: Ο αλγόριθμος στη διαπεράσει που κάνει κάθε φορά, συγκρατεί το μεγαλύτερο στοιχείο από αυτά που βρίσκει, και ελέγχει αν ο (ταξινομημένος) πίνακα εισαγωγής V είναι μηδενικός (αυτός ο έλεγχος χρησιμεύει στα παρακάτω βήματα).

5. *Υπολογισμός αριθμού πινάκων και το μέγεθος τους*, σε αυτό το σημείο ο αλγόριθμος υπολογίζει, από τον αριθμό των εμφανίσεων αυτού του στοιχείου που είχε τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα εισαγωγής V, τον αριθμώ των πινάκων αλλά και το μέγεθος των στοιχείων του κάθε πίνακα. Για παράδειγμα στην Ενότητας Α.2.2 στο Παράρτημα Α υπολογίσθηκε ότι χρειάζονται 6 πίνακες (έξι εμφανίσεις του στοιχείου 671) όπου ο κάθε πίνακας θα περιέχει 205 εγγραφές (στοιχεία).

6. *Εμφάνιση υπολογιζόμενων στοιχείων*, γίνεται η εμφάνιση των παραγόμενων στοιχείων του βήματος 5.

7. *Δημιουργία πινάκων και αρχικοποίηση τους* με τον αριθμό Ο (μηδέν), με βάση τα παραγόμενα στοιχεία του βήματος 5 γίνεται η δημιουργία των πινάκων και η αρχικοποίηση τους με τον αριθμό Ο (μηδέν). Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, δημιουργήθηκαν 6 πίνακες.

8. *Κάνει επανάληψη όσος είναι ο αριθμός των πινάκων n*, με βάση τον αριθμό των εμφανίσεων του πρώτου στοιχείου που βρέθηκε, κάνει επανάληψη. Αυτή η επανάληψη χρειάζεται για να αρχικοποιηθούν οι παραγόμενοι πίνακες. Αμέσως μετά η διαδικασία μεταβαίνει στο βήμα 9. Αν τελειώσει η επανάληψη τότε μεταβαίνει στο βήμα 10.

9. *Αρχικοποίηση Πινάκων με το στοιχείο που είχε τις περισσότερες εμφανίσεις*, μετά τη δημιουργία των πινάκων γίνεται η αρχικοποίηση τους με το πρώτο στοιχείο που βρέθηκε στο βήμα 4. Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, η αρχικοποίηση των έξι πινάκων έγινε με τον αριθμό 671. Η διαδικασία μεταβαίνει στο βήμα 8 μέχρι τέλους των επαναλήψεων.

10. *Αντικατάσταση του στοιχείου εισαγωγής των πινάκων με τον αριθμό 0 (μηδέν)*, αφού έγινε η αρχικοποίηση των πινάκων με τον πρώτο αριθμό, γίνεται η αντικατάσταση του στον (ταξινομημένο) πίνακα εισαγωγής V με τον αριθμό 0. Σε συνέχεια του

παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, αντικαταστάθηκε ο αριθμός 671 με τον αριθμό 0 στον (ταξινομημένο) πίνακα εισαγωγής V.

11. *Εύρεση του επόμενου στοιχείου του πίνακα με τις περισσότερες εμφανίσεις*, σε αυτό το σημείο ο αλγόριθμος θα βρει το επόμενο στοιχείο που έχει τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα εισαγωγής V. Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, αυτός ο αριθμός είναι ο 421 και έχει και αυτός 6 εμφανίσεις στο (ταξινομημένο) πίνακα εισαγωγής V.

12. *Είναι ο πίνακας εισαγωγής μηδενικός* ;, σε αυτό το σημείο ο αλγόριθμος ελέγχει αν ο (ταξινομημένος) πίνακα εισαγωγής V είναι μηδενικός. Αν όχι τότε μεταβαίνει στο βήμα 13. Αν ναι τότε μεταβαίνει στο βήμα 15. Θυμίζετε ότι όταν ο αλγόριθμος κάνει εύρεση του στοιχείου που έχει τις περισσότερες εμφανίσεις, ελέγχει και αν ο (ταξινομημένος) πίνακα εισαγωγής V είναι μηδενικός.

13. *Εισαγωγή στους πινάκες των στοιχείων με τις περισσότερες εμφανίσεις*, εισάγει το επόμενο στοιχείο που βρέθηκε με τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα εισαγωγής V, στους παραγόμενους πίνακες. Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, εισάγει τον αριθμό 421 στους πίνακες που δημιουργήθηκαν.

    Σημείωση: Ο αριθμός των εισαγωγών στους πίνακες γίνεται με βάση τον αριθμό των εμφανίσεών του στοιχείο που είχε τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα εισαγωγής V. Ο αριθμός αυτός αλλάζει και δεν είναι ίδιος κάθε φορά.

14. *Αντικατάσταση του στοιχείου εισαγωγής των πινάκων με τον αριθμό 0 (μηδέν)*, αφού γίνει η εισαγωγή του επόμενου στοιχείου, γίνεται η αντικατάσταση του στον (ταξινομημένο) πίνακα εισαγωγής V με τον αριθμό 0. Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, αντικαταστάθηκε ο αριθμός 421 με τον αριθμό 0 στον (ταξινομημένο) πίνακα εισαγωγής V.

    Σημείωση: Από αυτό το βήμα (14) μεταβαίνει ο αλγόριθμος στο βήμα 16 απευθείας χωρίς να μεσολαβήσει το βήμα 15.

15. *Τύπωσε ότι ο πίνακας είναι μηδενικός*, τυπώνει ότι τελικά ο (ταξινομημένος) πίνακας εισαγωγής V είναι μηδενικός.

16. *Επανάληψη μέχρι να μηδενιστεί ο πίνακας εισαγωγής*, εάν ο (ταξινομημένος) πίνακας εισαγωγής V δεν είναι μηδενικός, τότε ο αλγόριθμος επανέρχεται στο βήμα 11 έως ότου μηδενιστεί. Αν έχει μηδενιστεί τότε μεταβαίνει στο επόμενο βήμα (17).

17. *Εμφάνισε τα περιεχόμενα των πινάκων*, σε αυτό το σημείο ο αλγόριθμος εμφανίζει τα περιεχόμενα των πινάκων που έχουν δημιουργηθεί. Σε συνέχεια του παραδείγματος της Ενότητας Α.2.2 του Παραρτήματος Α, εμφανίζονται τα περιεχόμενα των έξι πινάκων που έχουν δημιουργηθεί.

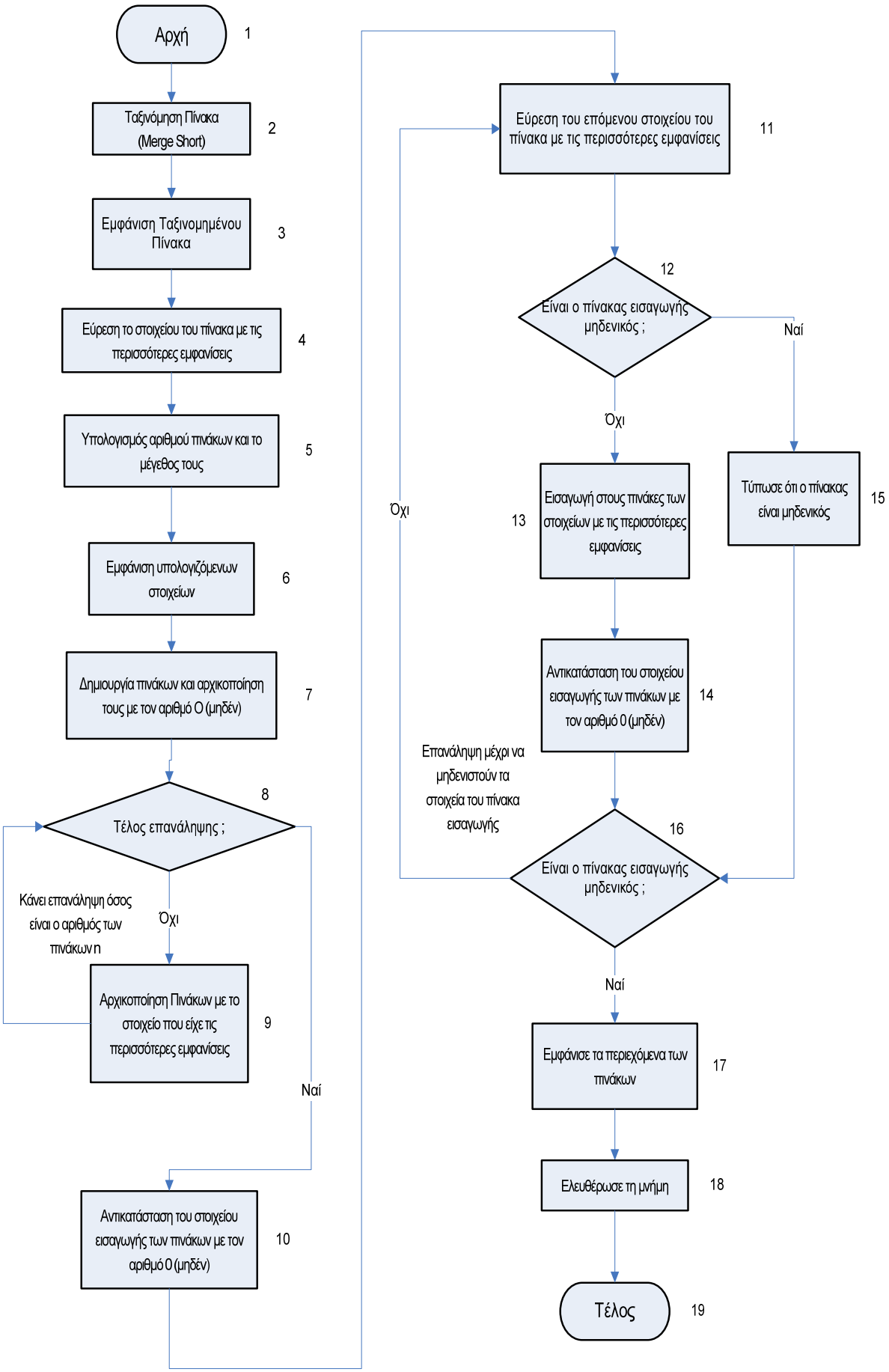
18. *Ελευθέρωσε τη μνήμη*, σε αυτό το σημείο ο αλγόριθμος ελευθερώνει τη μνήμη που δεσμευτικέ κατά τη δημιουργία των πινάκων.

19. Τέλος

Βελτιστοποίηση Αλγόριθμου

Ο αλγόριθμος που υλοποιήθηκε δεν είναι αποδοτικός, συγκεκριμένα κάνει πολλές φορές διαπέραση τον ταξινομημένο πατρικό πίνακα τόσο για να βρει το επόμενο στοιχείο που εμφανίζεται πολλές φορές, αλλά όσο και να μηδενίζει τις αντίστοιχες θέσεις αυτών. Μια αποδοτικότερη προσέγγιση είναι να γίνεται καταγραφή των στοιχείων από την πρώτη διαπεράσει του ταξινομημένου πατρικού πίνακα. Δηλαδή να κρατούνται σε μια συνδεδεμένη λίστα όλοι οι αριθμοί που καταγράφονται καθώς και οι ποσότητες εμφάνισης αυτών. Κατόπιν να χρησιμοποιηθεί η συνδεδεμένη λίστα ως μονάδα αρχικοποίησης των παραγόμενων πινάκων. Αυτή η προσέγγιση θα αφαιρέσει πολλές διαπεράσεις και θα αποσυμφόρηση τον επεξεργαστή. Η υλοποίηση του πρότυπου αλγόριθμου, κατά αυτόν τον τρόπο, είχε περισσότερο εκπαιδευτικό χαρακτήρα για να διαπιστωθεί ότι υπάρχει η δυνατότητα δημιουργίας του. Η βελτίωση του όμως δεν επιχειρήθηκε μιας και υπήρχαν άλλα σημαντικότερα θέματα που έπρεπε να διευθετηθούν.

Η ερευνητική ομάδα ευελπιστεί ότι μετά το πέρας των διαδικαστικών της παρουσίασης της μεταπτυχιακής διατριβής, θα προβεί στη βελτίωση του.



Εικόνα 4.22: Τα βήματα και η δομή του αλγόριθμου CAM_ARRAY, το διάγραμμα ροής του.

Συναρτήσεις

Στην Εικόνα 4.23 φαίνονται οι συναρτήσεις που χρησιμοποιούνται για τη διευθέτηση του αλγόριθμου.

Αυτές είναι :

- Η συνάρτηση ***void printArray(char*** *in,* ***int*** *array[],* ***int*** *n)*, η οποία τυπώνει τα περιεχόμενα ενός μονοδιάστατου πίνακα, δεν επιστρέφει τίποτα.

- Η συνάρτηση ***void mergesort(int*** *array[],* ***int*** *low,* ***int*** *high)*, η οποία ταξινομεί έναν μονοδιάστατο πίνακα σύμφωνα με τον αλγόριθμο ***merge short***, δεν επιστρέφει τίποτα.

- Η συνάρτηση ***int*** ****array_3D (int*** *x,* ***int*** *y,* ***int*** *z)*, δημιουργεί δυναμικά μονοδιάστατους πίνακες και τους επιστρέφει. Η κανονική δομή της συνάρτησης είναι να επιστρέφει δυσδιάστατους πίνακες, αλλά έχει τροποποιηθεί να επιστρέφει μονοδιάστατους πίνακες. Η δέσμευση μνήμης γίνεται για τρισδιάστατο πίνακα. Επιστρέφει τελικά ένα τρισδιάστατο πίνακα ο οποίος περιέχει μονοδιάστατους πίνακες. Η δομή του τρισδιάστατου πίνακα θα παρουσιαστεί παρακάτω.

- Η συνάρτηση *Structure* ***eyresi_arithmou*** *(****int*** *array[],* ***int*** *maxSizeArray)*, η οποία βρίσκει το στοιχείο του ταξινομημένου πίνακα εισαγωγής το οποίο έχει τις περισσότερες εμφανίσεις. Παράλληλα ελέγχει αν ο ταξινομημένος πίνακας εισαγωγής είναι μηδενικός. Επιστρέφει μια δομή τύπου *Structure* , περισσότερα για τη δομή αυτή παρακάτω.

- Η συνάρτηση ***void antikatasasi*** *(****int*** *array[],* ***int*** *in_number,* ***int*** *count,* ***int*** *maxSizeArray)*, η οποία αντικαθιστά το στοιχείο (***int*** *in_number*) του ταξινομημένου πίνακα εισαγωγής με τον αριθμό 0 (μηδέν), δεν επιστρέφει τίποτα.

- Η συνάρτηση ***void insert_items_in_array3D_main(int*** ****array,* ***int*** *x,****int*** *y,* ***int*** *z,* ***int*** *item)*, κάνει εισαγωγή στοιχείων στους παραγόμενους πίνακες, δεν επιστρέφει τίποτα.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXARRAY 11 // το μέγεθος του πίνακα εισαγωγής

// δομή Structure η οποία περιέχει τρεις μεταβλητές τύπου int,
// οι μεταβλητές αυτές χρειάζονται για τον καθορισμό του στοιχείου αυτού
// που έχει τις περισσότερες εμφανίσεις καθώς και αν ο πίνακας είναι μηδενικός.
// η δήλωση της δομής χρειάζεται για τον καθορισμό, έλεγχο και δημιουργία των πινάκων (CAM)
typedef struct {
    int sum;        // το πλήθος εμφάνισης του στοιχείου
    int item;       // ποιο στοιχείο είναι αυτό
    int zero_array; // αν ο πίνακας είναι μηδενικός
}Structure;

//ο σχολιασμός των συναρτήσεων που καλούνται, γίνεται μέσα στη main αλλά και στη θέση δήλωσης τους
void printArray(char* in, int array[], int n);
void mergesort(int array[], int low, int high);
int ***array_3D (int x, int y, int z);
Structure eyresi_arithmou (int array[], int maxSizeArray);
void antikatasasi (int array[], int in_number, int count, int maxSizeArray) ;
void insert_items_in_array3D_main(int ***array, int x,int y, int z, int item);

int main()
{
```

Εικόνα 4.23: Οι συναρτήσεις που χρησιμοποιούνται στη λειτουργία του αλγόριθμου CAM_ARRAY και η δήλωση της δομής Structure

Δομή Structure

Η δομή ***Structure*** της Εικόνας 4.23 είναι δομή τύπου ***struct*** και περιέχει τις παρακάτω 3 μεταβλητές:

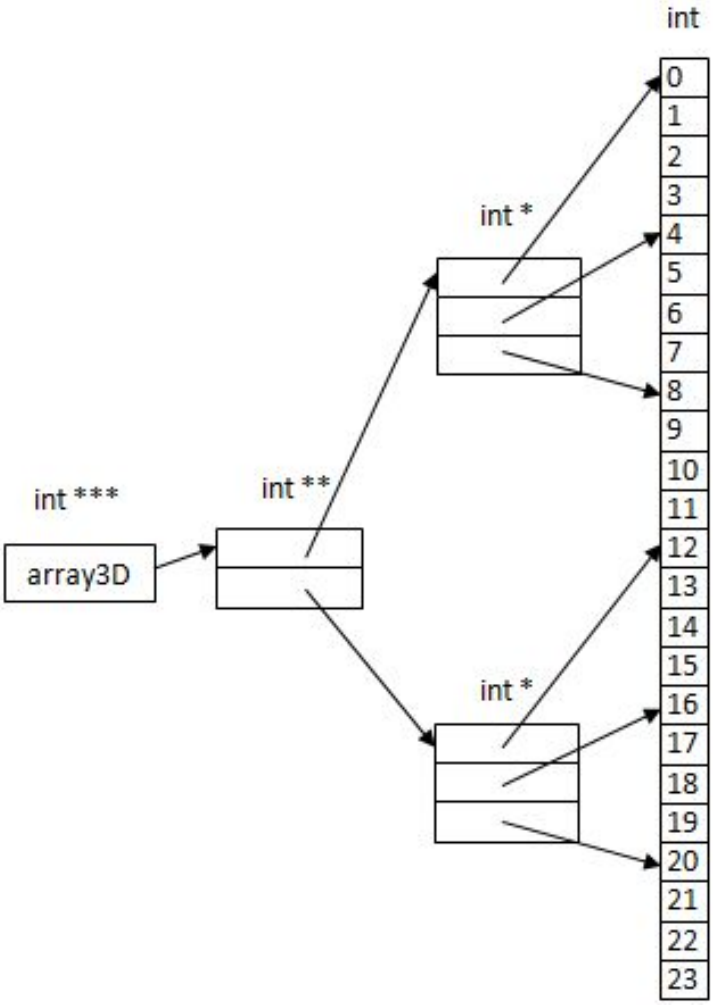
1. Μεταβλητή ***sum*** τύπου **int**, η οποία μετράει το άθροισμα των εμφανίσεων του στοιχείου που έχει τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα V.

2. Μεταβλητή ***item*** τύπου **int**, η οποία περιέχει αυτό το στοιχείο που έχει τις περισσότερες εμφανίσεις στον (ταξινομημένο) πίνακα V.

3. Μεταβλητή ***zero_array*** τύπου **int**, η οποία χρησιμοποιείται για να πιστοποιήσει το μηδενισμό του (ταξινομημένου) πίνακα V, επιστρέφει '0' ή '1'.

Δομή Τρισδιάστατου Πίνακα

Η δομή του τρισδιάστατου πίνακα που δημιουργείται από τη συνάρτηση ***eyresi_arithmou*** φαίνεται στην Εικόνα 4.24.



Εικόνα 4.24: Ο τρισδιάστατος πίνακας που δημιουργείται στη συνάρτηση eyresi_arithmou του αλγόριθμου CAM_ARRAY

Όλος ο αλγόριθμος παρουσιάζεται στην Ενότητα Α.2.1 στο Παράρτημα Α .

4.3 Block RAM - ROM

Η χρήση μνήμης RAM σε λογικά κυκλώματα που υλοποιούνται με γλώσσα περιγραφής υλικού είναι αρκετά συχνό φαινόμενο αλλά και άκρως απαραίτητο για κάποια από αυτά. Οι τρόποι που μπορεί κανείς να δημιουργήσει μία μνήμη RAM με τη γλώσσα περιγραφής υλικού VHDL σε συνδυασμό με το εργαλείο σχεδίασης ISE είναι τρεις.

Αναλυτικά έχουμε:

1. Με το να γραφεί αποκλειστικός κώδικας HDL ο οποίος θα καθορίζει το μέγεθος και τις ιδιότητες της μνήμης.

2. Με το να χρησιμοποιήσει το πρόγραμμα *Core Generator* το οποίο προσφέρει στο σχεδιαστή τη γραφική απεικόνιση (*Graphic User Interface*) της μονάδας μνήμης που θέλει να χρησιμοποιήσει. Η γραφική απεικόνιση είναι πολύ εύχρηστη μιας και ο τρόπος καθορισμού των παραμέτρων και των ιδιοτήτων είναι πολύ εύκολος. Συγκεκριμένο παράδειγμα παρουσιάσθηκε στην Ενότητα 4.2.1 όπου αναπτύχθηκε η δημιουργία της μνήμης CAM.

3. Και ο τρίτος τρόπος είναι μέσω της βιβλιοθήκης προτύπων HDL (Verilog, VHDL), όπου υπάρχουν πρότυπα σχεδίασης με χρήση κώδικα (Εικόνα 5.3). Στη βιβλιοθήκη αυτή υπάρχουν διάφορα πρότυπα μνημών RAM (ROM) που εξυπηρετούν σχεδιαστικές ανάγκες.

Από τους τρεις τρόπους μόνο ο πρώτος υπόσχεται ότι το παραγόμενο λογικό κύκλωμα της μνήμης μπορεί να δουλέψει στα περισσότερα εργαλεία σχεδίασης CAD. Τα υπόλοιπά δύο μόνο με τη σύμφωνη συγκατάθεση των κατασκευαστών των εργαλείων σχεδίασης CAD. Τα δυο τελευταία αποτελούν σχεδιαστικά πρότυπα του εργαλείου σχεδίασης ISE και ως εκ τούτου μόνο αυτό μπορεί να τα διαχειριστεί. Τέτοια πρότυπα μπορούν να βρεθούν και σε άλλα εργαλεία σχεδίασης τα οποία και αυτά αποτελούν αποκλειστική χρήση των ίδιων. Μόνο αν υπάρξει πρότυπο το οποίο να πιστοποιείται από κάποιον κατασκευαστή, μπορεί να χρησιμοποιηθεί και από άλλα εργαλεία σχεδίασης CAD.

Στην περίπτωση της υλοποίησης των επερωτήσεων Q3 και Q4 χρησιμοποιήθηκε το δεύτερο πρότυπο.

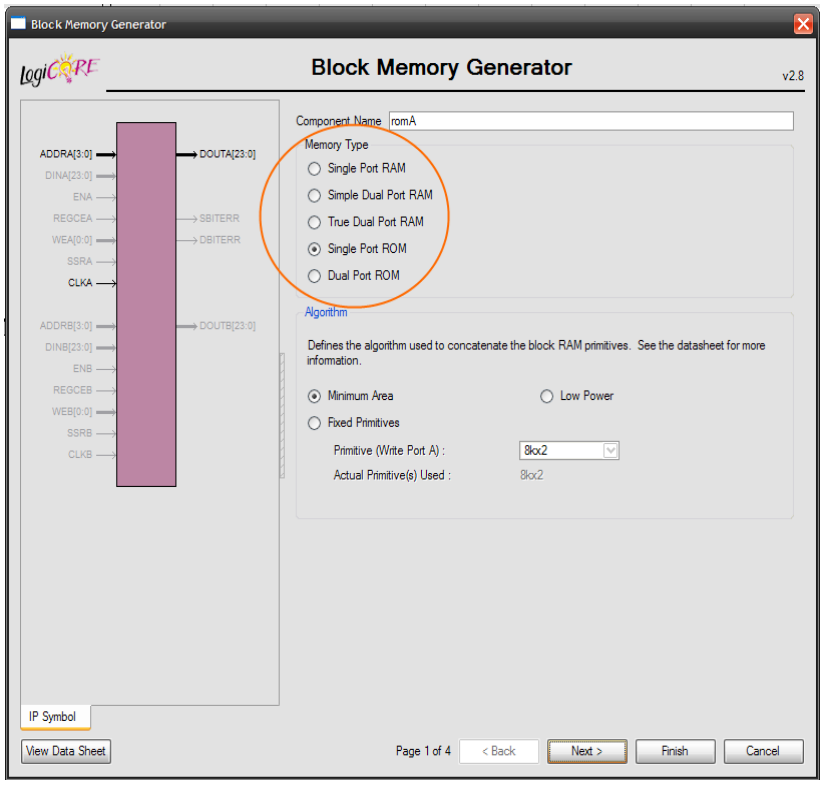
4.3.1 Ιδιότητες μνήμης RAM-ROM και ο Τρόπος Αρχικοποίησης της

Memory Type

Μία από τις ιδιότητες της μνήμης RAM είναι ότι μπορεί να υλοποιεί μια ποικιλία διαφόρων τύπων μνήμες. Στην Εικόνα 4.25 φαίνονται αυτοί οι τύποι και όπως και ο τύπος που επιλέχθηκε για να υλοποιηθεί η μνήμη **romA** (Ενότητα 5.2.4) της επερώτησης Q4.

Συγκεκριμένα υλοποιεί τις παρακάτω μνήμες :

1. Single Port RAM
2. Simple Dual Port RAM
3. True Dual Port RAM
4. Single Port ROM
5. Dual Port ROM



Εικόνα 4.25: Επιλογή μνήμης Single Port ROM για τη διευθέτηση της μνήμης romA της επερώτησης Q4, από το εργαλειο σχεδιασης Xilinx ISE

Μέγεθος Μνήμης

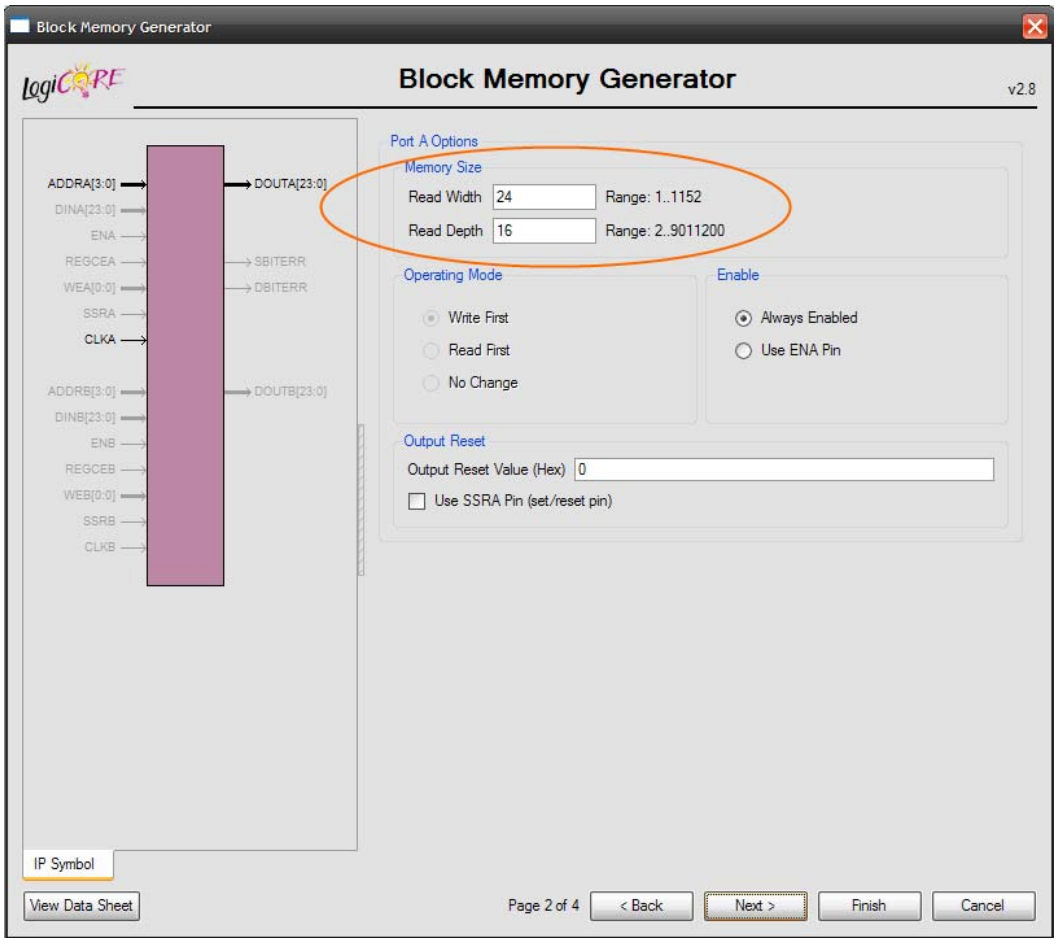
Μία δεύτερη ιδιότητα είναι το μέγεθος της μνήμης, το οποίο εξαρτάται από τη συγκεκριμένη αρχιτεκτονική δομή της συσκευής FPGA.

Συγκεκριμένα πρέπει να αρχικοποιηθούν οι παράμετροι :

1. *Read Width*, το οποίο είναι το μέγεθος των δεδομένων μέσα στη μνήμη και μπορεί να λαμβάνει τιμές από 1 bit μέχρι 1152 bit.

2. *Read Depth*, το οποίο είναι το μέγεθος των διευθύνσεων στη μνήμη, δηλαδή το βάθος της μνήμης. Οι τιμές που μπορεί να λάβει είναι από 2 έως 9011200.

Στις υλοποιήσεις που έγιναν για τις δύο επερωτήσεις Q3 και Q4, χρησιμοποιήθηκε το μέγεθος των δεδομένων *Read Width* ίσο με 24 και το μέγεθος των διευθύνσεων *Read Depth* ίσο με 16. Στην Εικόνα 4.26 στο πεδίο *Memory Size* φαίνονται οι δηλώσεις που επιλέχθηκαν.

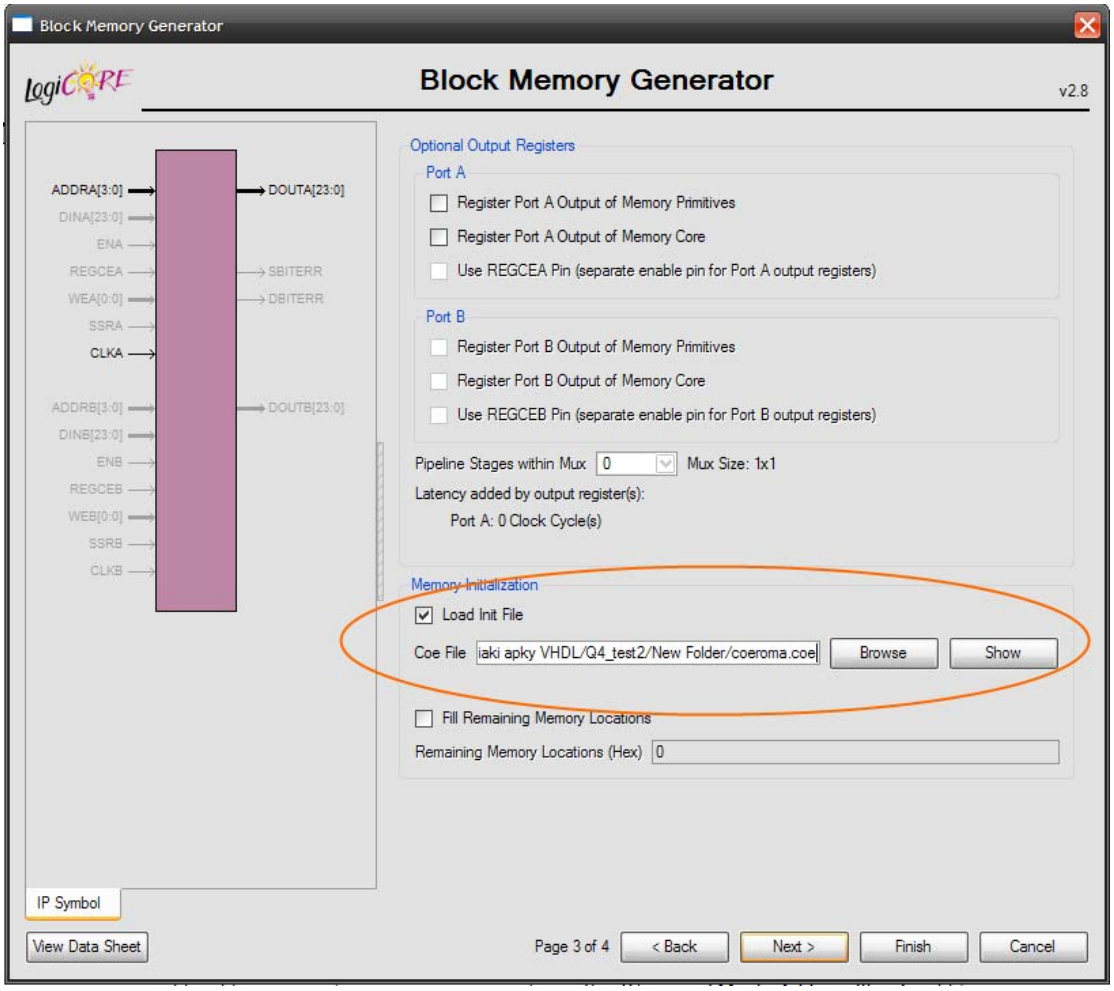


Εικόνα 4.26: Διευθέτηση μεγέθους μνήμης ROM στα πεδία Read Width και Read Depth, από το εργαλειο σχεδιασης Xilinx ISE

Αρχικοποίηση Μνήμης CAM

Η διαδικασία αρχικοποίησης της μνήμης ROM είναι η ίδια με αυτήν που ακολουθείται και για την αρχικοποίηση της μνήμης CAM (Ενότητα 4.2.1).

Για να αρχικοποιηθεί μια μνήμη ROM θα πρέπει να ενεργοποιηθεί η επιλογή *Load Init File* και αμέσως μετά να βρεθεί το αρχείο *.coe με το οποίο θα αρχικοποιηθεί η μνήμη με τα δεδομένα αρχικοποίησης. Στην Εικόνα 4.27 φαίνεται αυτή η διαδικασία επιλογής, συγκεκριμένα γίνεται αρχικοποίηση της μνήμης romA από το αρχείο αρχικοποίησης coeroma.coe.



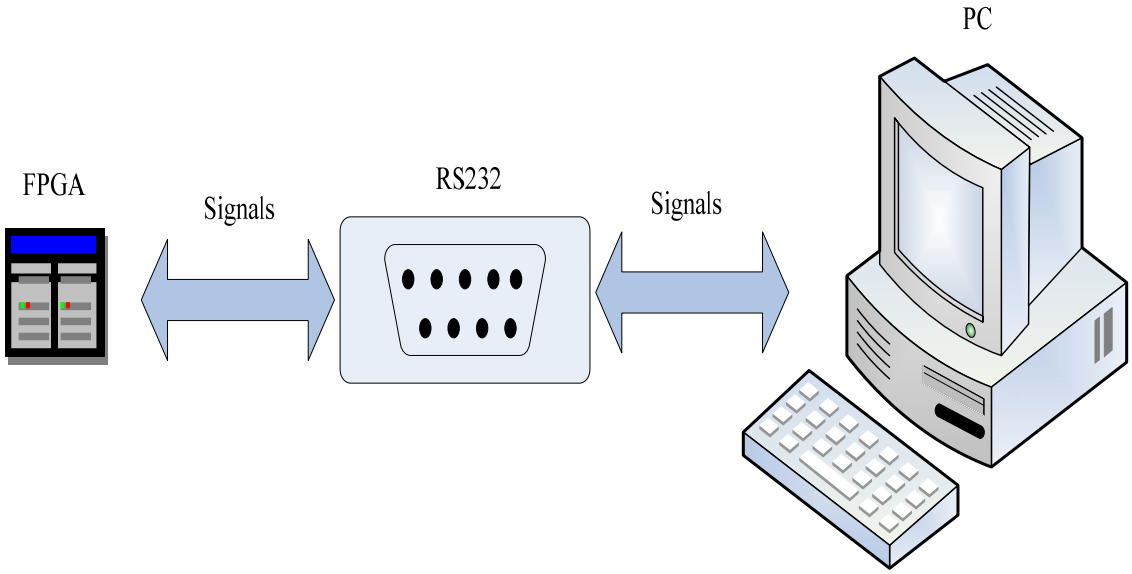
Εικόνα 4.27: Διαδικασία επιλογής για την αρχικοποίηση της μνήμης romA από το αρχείο αρχικοποίησης coeroma.coe, από το εργαλειο σχεδιασης Xilinx ISE

Υπάρχουν και άλλες ιδιότητες αλλά σε αυτήν την ενότητα παρουσιάστηκαν αυτές που χρησιμοποιήθηκαν για τη λύση των επερωτήσεων Q3 και Q4. Για περισσότερη εμβάθυνση γύρω από τις μνήμες RAM - ROM καλείται ο αναγνώστης να ανατρέξει στην προτεινόμενη βιβλιογραφία [30,31], και στο κεφάλαιο 11 του βιβλίου «*FPGA Prototyping By VHDL Examples*» [18].

4.4 RS232 – UART

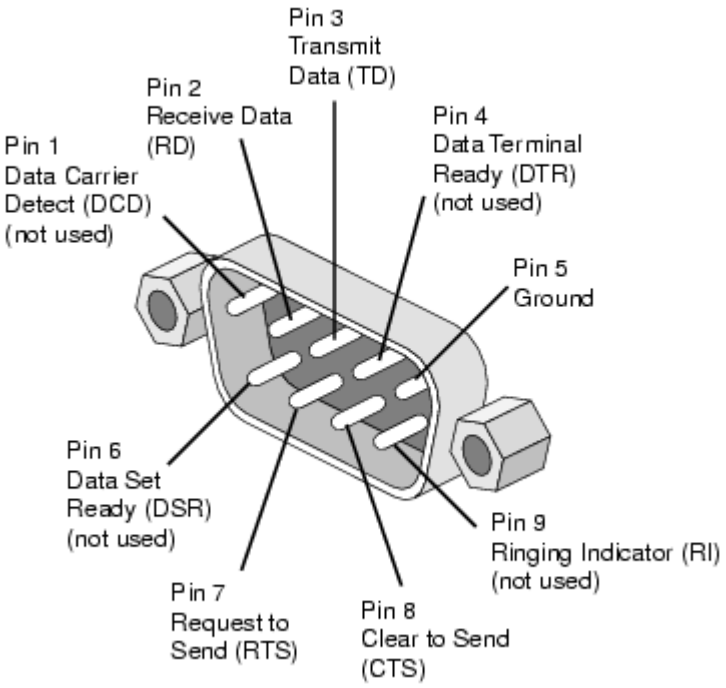
4.4.1 Πρότυπο RS232

Για να μπορέσει να γίνει η διευθέτηση της επικοινωνίας του Η/Υ με τη συσκευή θα πρέπει να χρησιμοποιηθεί η θύρα COM1 και το πρότυπο RS23. Στην Εικόνα 4.28 φαίνεται αυτός ο τρόπος διευθέτησης της επικοινωνίας.



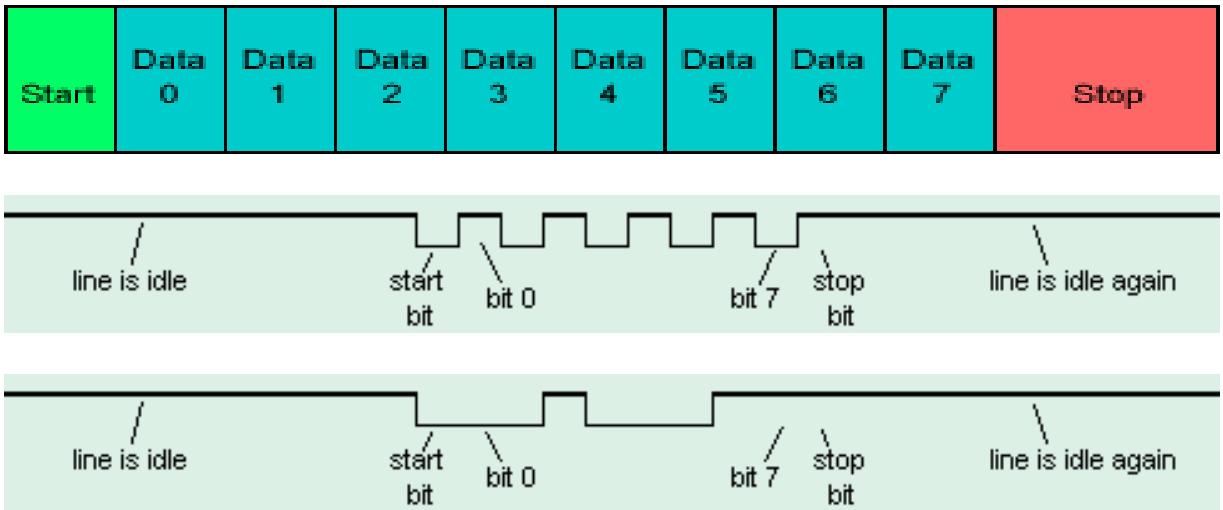
Εικόνα 4.28: Απεικόνιση επικοινωνίας μέσω του πρωτοκόλλου RS232

Το πρότυπο επικοινωνία έχει τη δομή που φαίνεται στην Εικόνα 4.29. Όπως μπορεί κανείς να παρατηρήσει οι ακροδέκτες είναι στο σύνολο 9 και ο καθένας από αυτούς επιτελεί μια λειτουργία. Στην επίλυση των επερωτήσεων χρησιμοποιήθηκαν μόνο οι δύο από τους εννιά ακροδέκτες, ο ακροδέκτης 2 και ο ακροδέκτης 3. Ο ακροδέκτης 2 λαμβάνει τα δεδομένα και ο ακροδέκτης 3 τα στέλνει.



Εικόνα 4.29: Δομή πρότυπου επικοινωνίας RS232

Το πρότυπο πέραν της υλικής δομή έχει και ένα πρότυπο επικοινωνίας το οποίο περιλαμβάνει την αποστολή των bit. Αυτό γίνεται ένα κάθε φορά και σε ομάδες των 8 bit, πριν και μετά τα 8bit περιλαμβάνονται αλλά δυο bit τα οποία υποδηλώνουν την αρχή και το τέλος της ομάδας των 8bit. Συγκεκριμένα το πρώτο από τα δύο bit όταν είναι '0' υποδηλώνει το ξεκίνημα της ομάδας (*Start Bit*) των 8bit. Το σύστημα που λαμβάνει τα σήματα είναι υποχρεωμένο να συγχρονιστεί και να κάνει σωστά την δειγματοληψία των 8bit. Κατόπιν αυτού έρχεται και το δεύτερο bit που υποδηλώνει τον τερματισμό της ομάδας των 8bit όταν είναι '1' (*Stop Bit*). Αυτή η ακολουθία γίνεται τόσο κατά την αποστολή αλλά και όσο κατά τη λήψη των δεδομένων. Στην Εικόνα 4.30 φαίνονται δύο δειγματοληψίες που έχουν γίνει με το πρωτόκολλο επικοινωνίας RS232.



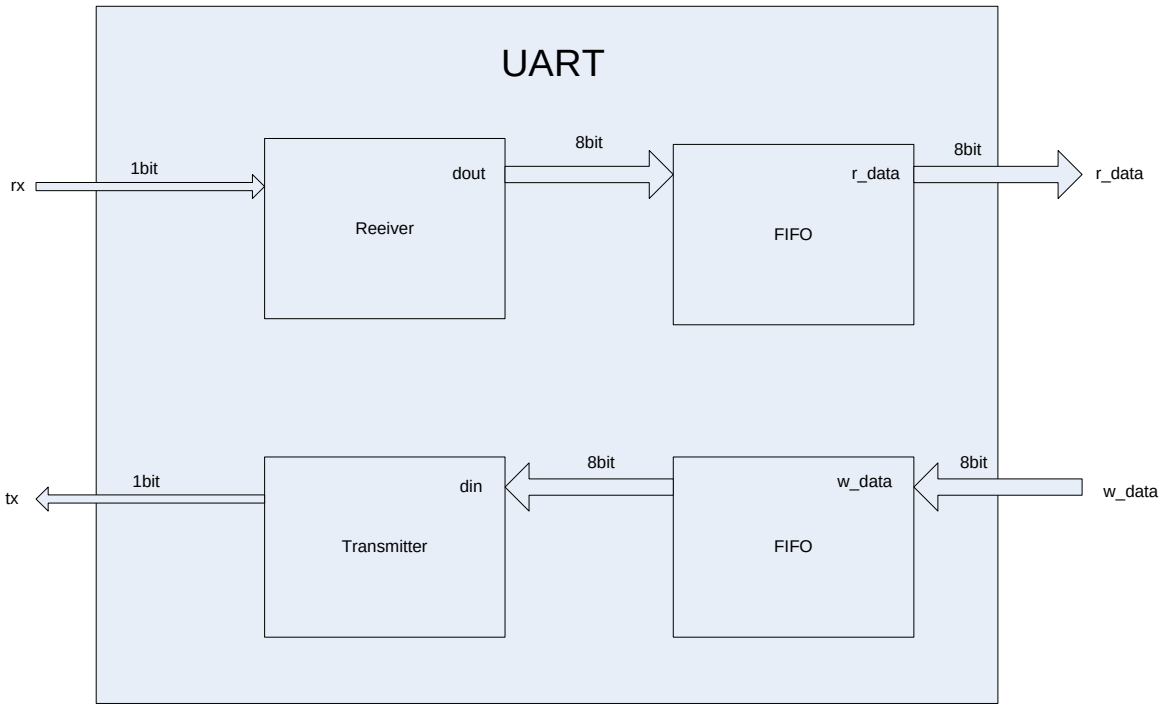
Εικόνα 4.30: Δειγματοληψίες με το πρωτόκολλο επικοινωνίας RS232

Για τη διευθέτηση της επικοινωνίας χρησιμοποιήθηκε καλώδιο της μορφής που φαίνεται στην Εικόνας 4.31, όπου από τη μια πλευρά έχει "θηλυκό" προσαρμογέα και από την άλλη "αρσενικό" προσαρμογέα.

Εικόνα 4.31: Καλώδιο RS232 με αρσενικό και θηλυκό προσαρμογέα

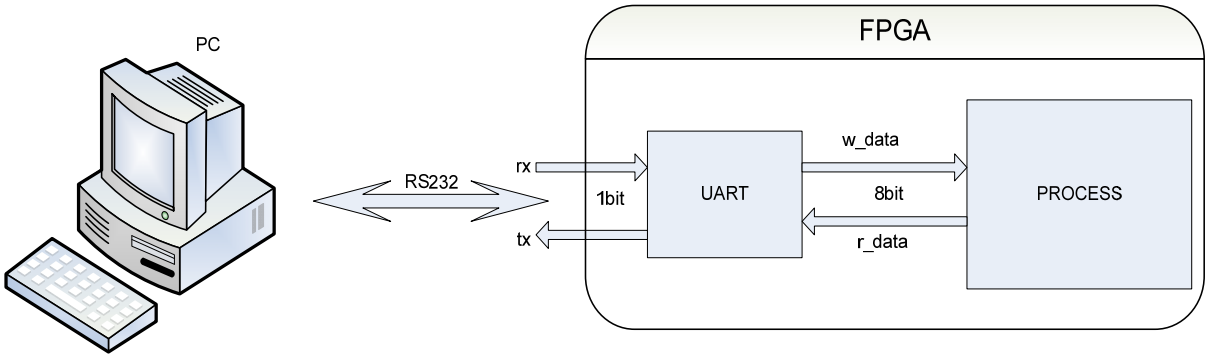
4.4.2 Πρωτόκολλο Επικοινωνίας UART

Για να μπορέσει να γίνει η επικοινωνία μέσω του προτύπου RS232 θα πρέπει να γραφεί κώδικας επικοινωνίας ο οποίος να διαχειρίζεται τα δεδομένα εισόδου και εξόδου της συσκευής FPGA. Ο κώδικας αυτός ονομάζεται UART (*Universal Asynchronous Receiver/Transmitter*). Η δομή κάθε τέτοιου κώδικα διαφέρει από κατασκευαστή σε κατασκευαστή, κάποιος ενσωματώνει τις μονάδες FIFO μέσα στις επιμέρους μονάδες της αποστολής και λήψης δεδομένων και κάποιος άλλος τις έχει εξωτερικά. Στη λύση που υλοποιήθηκε χρησιμοποιήθηκε η δεύτερη δομή, αυτή παρουσιάζεται στην Εικόνα 4.32. Αυτός ο κώδικας επικοινωνίας θα διασφαλίσει την μεταξύ τους επικοινωνία, του Η/Υ και της συσκευής FPGA.



Εικόνα 4.32: Εσωτερική δομή του κώδικα UART που χρησιμοποιείται για την επίλυση των επερωτήσεων

Η γενική δομή της χρήσης των κυκλωμάτων επικοινωνίας UART με τις αντίστοιχες λογικές μονάδες που επιτελούν την επεξεργασία των επερωτήσεων (κάθε μια από τις τέσσερις Q1, Q2, Q3 και Q4), φαίνεται στην Εικόνα 4.33. Όλες οι επερωτήσεις που θα υλοποιήσουν το πειραματικό κομμάτι απευθείας στο υλικό, θα χρησιμοποιήσουν αυτήν τη δομή που φαίνεται.



Εικόνα 4.33: Αρχιτεκτονική δομή λογικών κυκλωμάτων που επιλύουν τις επερωτήσεις

Για περισσότερη εμβάθυνση γύρο από το πρωτόκολλο επικοινωνίας που χρησιμοποιείται, παρακαλείται ο αναγνώστης να δει το κεφάλαιο 7 του βιβλίου *«FPGA Prototyping By VHDL Examples»* [18].

Ο κώδικας επικοινωνίας UART που χρησιμοποιείται παρουσιάζεται στην Ενότητα Z.2 στο Παράρτημα Z .

4.5 Αναφορές

[13] Kostas Pagiamtzis, Student Member, IEEE, and Ali Sheikholeslami, Senior Member. «Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey». IEEE, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 3, Toronto, Canada, MARCH 2006.

[18] Pong P. Chu. «FPGA Prototyping By VHDL Examples, Xilinx SpartanTM-3V version». A JOHN WILEY & SONS, INC, PUBLICATION, ISBN 978-0-470-18531-5, Hoboken, New Jersey, USA, 2008.

[29] Xilinx Corporation. «Using Virtex-II Block RAM for High Performance Read/Write CAMs, XAPP260 (v1.1) ». Xilinx, Copyright © February 27, 2002.

[30] Xilinx Corporation. «Using Block RAM in Spartan-3 Generation FPGAs, XAPP463 (v2.0) ». Xilinx, Copyright © March 1, 2005.

[31] Xilinx Corporation. «Single – Port Block Memory Core v6.2, Logic Core, DS234». Xilinx, Copyright © April 28, 2005.

[32] Xilinx Corporation. «ISE 10.1 Quick Start Tutorial». Xilinx, Copyright © 2008.

[33] Xilinx Corporation. «ISE Design Suite 10.1 Software Manuals». Xilinx, Copyright © 2008.

[34] Xilinx Corporation. «Content-Addressable Memory v6.1, Logic Core, DS253». Xilinx, Copyright © September 19, 2008.

[35] Xilinx Corporation. «Spartan-3 FPGA Family Data Sheet». Xilinx, Copyright © 2009.

[36] Xilinx Corporation. «Spartan-3 Generation FPGA User Guide». Xilinx, Copyright © 2010.

Κεφάλαιο 5

Υλοποίηση Ερωτημάτων VHDL

Σε αυτό το κεφάλαιο ο αναγνώστης θα έχει την ευκαιρία να δει την τυπική διαδικασία συγγραφής σε ένα εργαλείο CAD, για τη διευθέτηση και τον προγραμματισμό σε μία συσκευή FPGA. Επίσης θα εκτελεστούν στην προσομοίωση όλες οι επερωτήσεις Q1,Q2,Q3 και Q4 που αναπτύχθηκαν στο Κεφάλαιο 3. Τέλος θα εξαχθούν οι εκτιμήσεις των προσομοιώσεων που εκτελέστηκαν.

Αναλυτικά στην Ενότητα 5.1.1 παρουσιάζεται η τυπική σχεδίαση σε εργαλείο CAD για συσκευές FPGA, στην Ενότητα 5.1.2 παρουσιάζονται οι τρόποι σχεδίασης στα εργαλεία CAD και στην Ενότητα 5.1.3 παρουσιάζεται η διαδικασία επιλογής συσκευής FPGA και εργαλείου σχεδίασης CAD για τη διευθέτηση των προσομοιώσεων αλλά και των υλοποιήσεων στο υλικό. Στην Ενότητα 5.2.1 προσομοιώνεται η επερώτηση Q1, στην Ενότητα 5.2.2 προσομοιώνεται η επερώτηση Q2, στην Ενότητα 5.2.3 προσομοιώνεται η επερώτηση Q3 και στην Ενότητα 5.2.4 προσομοιώνεται η επερώτηση Q4. Στην Ενότητα 5.3 έγινε προσπάθεια υλοποίησης των επερωτήσεων απευθείας στο υλικό αλλά η λόγοι αποτυχίας αναλύονται στη συγκεκριμένη ενότητα. Τέλος στην Ενότητα 5.4 γίνεται εκτίμηση των αποτελεσμάτων που παρήχθησαν κατά την προσομοίωση των επερωτήσεων.

5.1 Εισαγωγή

Σε αυτή την ενότητα θα γίνει μια εισαγωγή για τον τρόπο συγγραφής σε εργαλεία σχεδίασης CAD των συσκευών FPGA, αλλά και τη διαδικασία επιλογής τους.

5.1.1 Τυπική Σχεδίαση σε Εργαλείο CAD για Συσκευές FPGA

Η τυπική ροή σχεδίασης ενός ψηφιακού κυκλώματος με τη χρήση εργαλείων σχεδίασης CAD για συσκευές FPGA, φαίνεται στην Εικόνα 5.1.

Τα βήματα είναι τα εξής :

1. Εισαγωγή του σχεδίου (*Design Entry*), μπορεί να γίνει με διάφορους τρόπους οι οποίοι θα αναφερθούν παρακάτω.

2. Σύνθεση και προσδιορισμός διασύνδεσης (*Synthesis*), σε αυτό το βήμα γίνεται η σύνθεση των λογικών βαθμίδων που χρειάζεται να υλοποιηθούν. Αυτό μπορεί να γίνει από ένα ανεξάρτητο κατασκευαστή λογισμικού ή από τον κατασκευαστή του FPGA. Σε αυτό το στάδιο γίνεται η περιγραφή της συμπεριφοράς (*Behavioral Description*) της σχεδίασης η οποία μεταφράζεται σε περιγραφή δομής (*Structural Netlist*). Ο μεταφραστής (*Netlist Translators*) μεταφράζει την περιγραφή δομής σε μορφή που υποστηρίζει το εργαλείο του κατασκευαστή.

3. Λειτουργική προσομοίωση συνολικού συστήματος (*Functional Simulation*), εδώ γίνεται η πιστοποίηση της ορθότητας της λογικής σχεδίασης πριν υλοποιηθεί. Εκτελείται πάντα στα πρώτα στάδια της σχεδίασης είτε σε περιγραφή συμπεριφοράς είτε σε περιγραφή δομής.

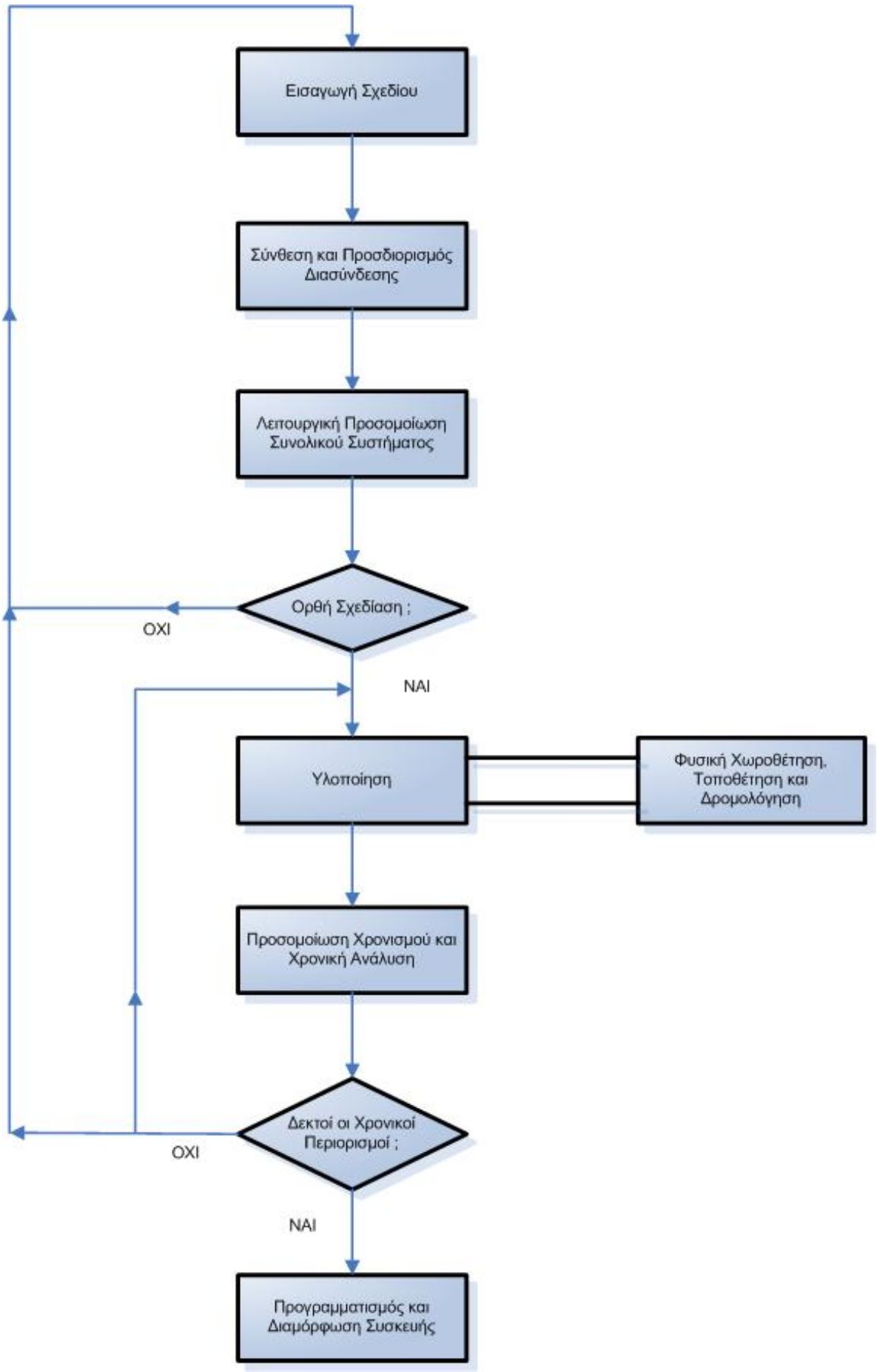
4. Αν η σχεδίαση είναι σωστή τότε συνεχίζει στα επόμενα βήματα (βήμα 5), αν δεν είναι τότε ο σχεδιαστής επανέρχεται στο πρώτο στάδιο και ελέγχει την ορθότητα της σχεδίασης ή αναθεωρεί τη σχεδίαση με νέα (βήμα 1).

5. Υλοποίηση (*Implementation*), σε αυτό το βήμα εκτελούνται τρεις επιμέρους λειτουργίες. Η μία έχει να κάνει με τη φυσική χωροθέτηση ή αντιστοίχηση (*Fitting* ή *Mapping*) όπου τεμαχίζονται οι συναρτήσεις στα λογικά στοιχεία, δηλαδή πως υλοποιούνται οι λογικές λειτουργίες στο εσωτερικό ενός λογικού στοιχείου. Η δεύτερη λειτουργία έχει να κάνει με την τοποθέτηση (*Placement*), δηλαδή που τοποθετούμε κάθε κομμάτι λογικής στη διάταξη των λογικών στοιχείων. Και τέλος η τρίτη λειτουργία έχει να κάνει με τη δρομολόγηση (*Routing*), τη διασύνδεση των λογικών στοιχείων μεταξύ τους, δηλαδή ποιες καλωδιώσεις χρησιμοποιούνται για να συνδεθούν τα λογικά στοιχεία μεταξύ τους καθώς και με τους ακροδέκτες.

6. Προσομοίωση χρονισμού (*Timing Simulation*) και Χρονική ανάλυση (*Timing Analysis*), το μεν πρώτο εκτελείται μετά την υλοποίηση και πιστοποιεί ότι η σχεδίαση λειτουργεί στην επιθυμητή ταχύτητα. Καθορίζει τα κρίσιμα μονοπάτια κάτω από τις χειρότερες συνθήκες και ανιχνεύει χρονικές παραβάσεις (setup time and hold time violations). Το δε δεύτερο εκτελείται και αυτό μετά την υλοποίηση και υπολογίζει την καθυστέρηση των μονοπατιών του κυκλώματος. Πιστοποιεί ότι η σχεδίαση ικανοποιεί τις χρονικές προδιαγραφές, πληροφορεί το σχεδιαστή για «αργά» μονοπάτια και προσφέρει υλικό για την τεκμηρίωση της σχεδίασης.

7. Αν η σχεδίαση πληροί τους χρονικούς περιορισμούς και τις χρονικές προδιαγραφές τότε συνεχίζει στο τελευταίο βήμα, αν όχι τότε ο σχεδιαστής : ή επανέρχεται στο πρώτο στάδιο και ελέγχει την ορθότητα της σχεδίασης, ακόμη και να αναθεωρήσει τη σχεδίαση με μια νέα (βήμα 1), ή επανέρχεται στο στάδιο της υλοποίησης (βήμα 5) και επιδιώκει επαναδιαμόρφωση και νέα υλοποίηση. Το τελευταίο μπορεί να γίνει είτε αυτόματα από το εργαλείο σχεδίασης είτε χειροκίνητα από τον ίδιο το χρήστη.

8. Προγραμματισμός και διαμόρφωση συσκευής (*Programming and Configuration*), το τελευταίο βήμα είναι αυτό που θα παράγει το αρχείο *BitStream* με το οποίο θα διαμορφωθεί και θα προγραμματιστεί η συσκευή FPGA.



Εικόνα 5.1: Τυπική μεθοδολογία σχεδίασης με χρήση εργαλείων CAD για συσκευές FPGA

Θεωρείται ότι η διαδικασία ανάπτυξης στοχεύει στην παραγωγή ενός προϊόντος που πληροί κάποιες προδιαγραφές. Οι πιο προφανείς απαιτήσεις είναι ότι το προϊόν θα πρέπει να λειτουργεί σωστά και ότι θα πρέπει να έχει κάποιο συγκεκριμένο βαθμό απόδοσης. Για να γίνει ένας πιο λεπτομερής έλεγχος των προδιαγραφών αυτών δεν αρκεί μόνο η μεθοδολογία σχεδίασης που αναλύθηκε παραπάνω. Θα πρέπει μετά το προγραμματισμό και τη διαμόρφωση της συσκευής, να γίνει έλεγχος και της συσκευής FPGA σε πραγματικές συνθήκες αν όντως πληροί της προδιαγραφές αυτές. Αυτή η διαδικασία ονομάζεται *πιστοποίηση στο κύκλωμα* (*in–circuit verification*) και πιστοποιεί ότι η σχεδίαση λειτουργεί σωστά στην τελική εφαρμογή, κάτω από πραγματικές συνθήκες λειτουργίας. Οι κατασκευαστές παρέχουν εργαλεία για να βοηθήσουν την πιστοποίηση στο κύκλωμα.

5.1.2 Τρόποι Σχεδίασης στα Εργαλεία CAD

Σε κάθε εργαλείο σχεδίασης CAD για την εξυπηρέτηση του χρήστη υπάρχει μια ποικιλία από εναλλακτικούς τρόπους σχεδίασης, κάποια από αυτά είναι:

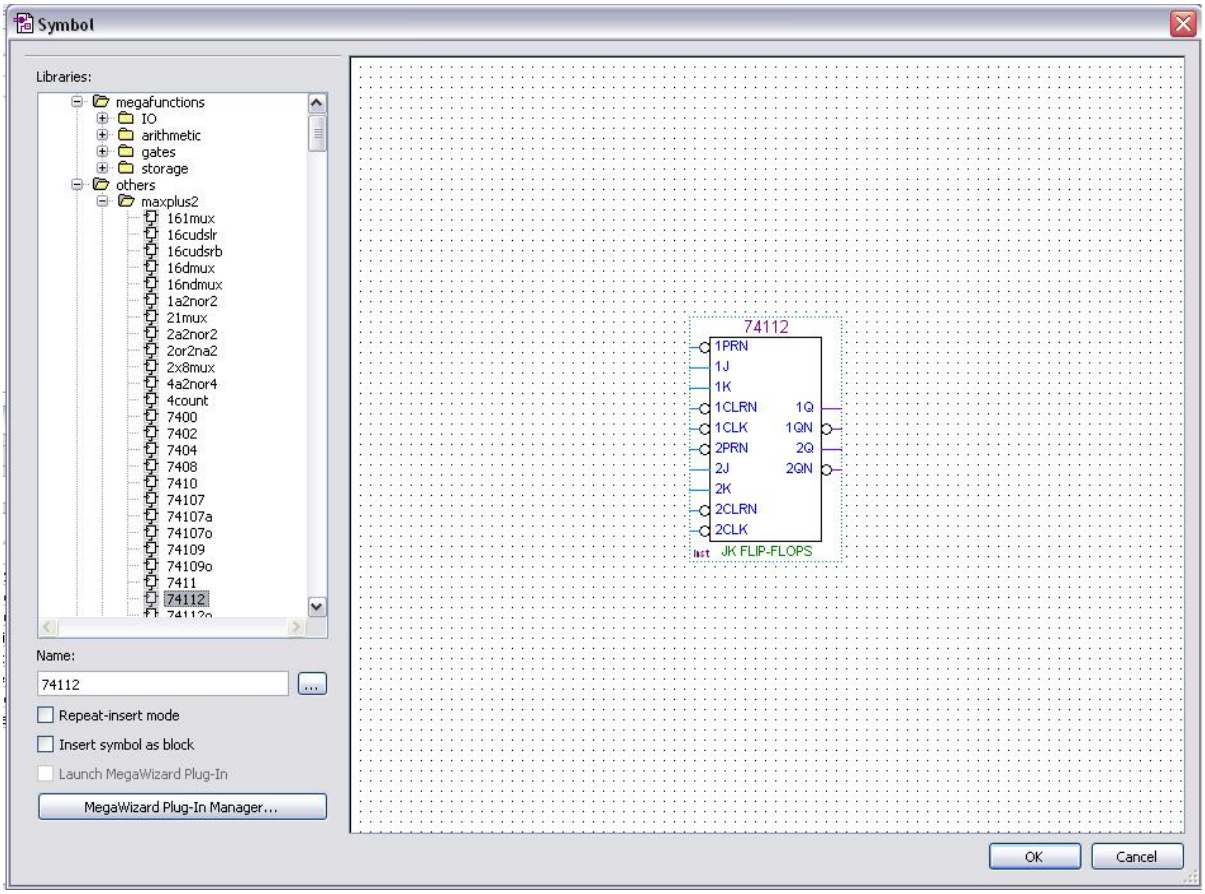
- Σύνθεση προγραμμάτων με χρήση έτοιμων σχηματικών προτύπων και ονομάζεται *σχηματική εισαγωγή* (*Schematic Entry*).

- Σύνθεση προγραμμάτων με χρήση γλωσσών HDL όπως Verilog και VHDL.

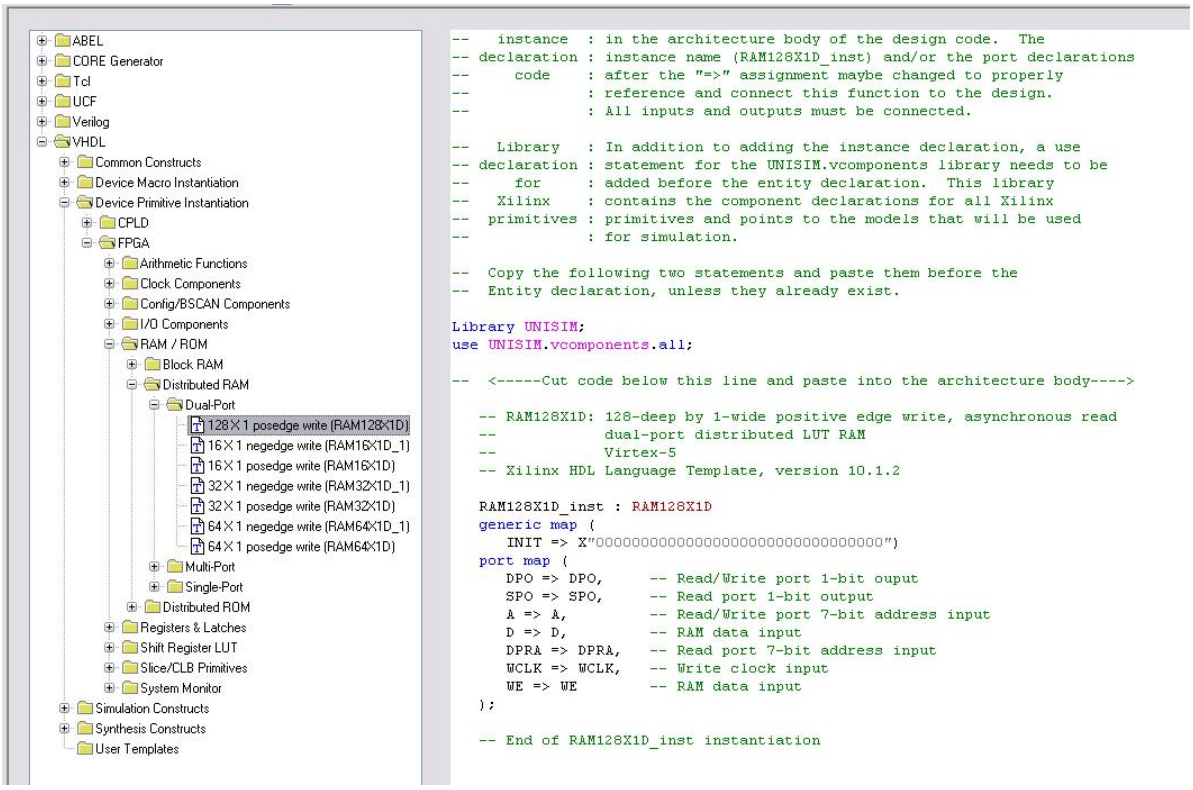
- Σύνθεση προγραμμάτων μέσω άλλων λογικών μονάδων που ήδη έχουν χρησιμοποιηθεί και κατασκευαστεί. Αυτά συνήθως τα δημιουργεί ο κάθε σχεδιαστής ξεχωριστά, σύμφωνα πάντα με τις ανάγκες του αλλά και την ειδίκευση κατασκευής που κάνει.

Και στα τρία πρότυπα υπάρχουν βιβλιοθήκες μέσα στις οποίες υπάρχουν έτοιμα σχεδιαστικά πρότυπα όπως πολυπλέκτες, λογικές πύλες, μνήμες RAM, flip flop και άλλα λογικά κυκλώματα, τα οποία μπορεί ο χρήστης να χρησιμοποιήσει και να δημιουργήσει το λογικό κύκλωμα της επιθυμίας του. Στο μεν σχηματικό υπάρχουν σχηματικά πρότυπα (Εικόνα 5.2) ενώ στη χρήση με γλώσσες HDL υπάρχουν πρότυπα σχεδίασης με χρήση κώδικα (Εικόνα 5.3). Στην τελευταία μέθοδο πάλι, το κάθε λογισμικό παρέχει μια ξεχωριστή βιβλιοθήκη στην οποία μπορεί ο χρήστης να αποθηκεύει τα πρότυπα που ο ίδιος δημιουργεί. Στις δυο πρώτες μεθόδους τα πρότυπα των βιβλιοθηκών είναι δημιούργημα των εταιριών κατασκευής των λογισμικών CAD, ενώ στην τρίτη μέθοδο τα πρότυπα είναι δημιούργημα (κατασκευή) του εκάστοτε χρήστη. Επιπλέον παρέχουν

υποδομές για τη χρήση των γλωσσών Verilog και VHDL. Το κάθε λογικό κύκλωμα που θα δημιουργηθεί από αυτές τις τρεις μεθόδους, θα εκτελεστεί για να παράγει το αρχείο BitStream με το οποίο θα προγραμματιστεί η συσκευή FPGA μέσω της θύρας *JTAG* (Εικόνα Β.10, Παράρτημα Β).



Εικόνα 5.2: Σχηματικό πρότυπο τυπικού ολοκληρωμένου κυκλώματος 74112 JK flip flop, από τη βιβλιοθήκη του εργαλείου σχεδίασης Quartus II [02,03] της εταιρίας Altera [04]



Εικόνα 5.3: Πρότυπο σχεδίασης μνήμης 128 x 1 Dual Port RAM με χρήση γλώσσας VHDL, από τη βιβλιοθήκη του εργαλείου σχεδίασης ISE [32,33] της εταιρίας Xilinx [37]

Στη διαδικασία ανάπτυξης που θα ακολουθηθεί θα έχει την ευκαιρία να δει ο αναγνώστης τον τρόπο ελέγχου των παραγόμενων κυκλωμάτων, δηλ την ορθότητα της σχεδίασης.

5.1.3 Επιλογή Συσκευής FPGA και Εργαλείου Σχεδίασης CAD

Από μια παλιότερη συγκριτική μελέτη [38] πάνω σε οικογένειες συσκευών FPGAs αλλά και των αντίστοιχων εργαλείων σχεδίασης τους, διαπιστώθηκε ότι τα αποτελέσματα της υλοποίησης που πετυχαίνουν είναι σχεδόν παρόμοια, δηλαδή η κάθε ομάδα πετυχαίνει παρόμοιες επιδώσεις.

Οι συσκευές αυτές ήταν :

- Cyclone [01] της εταιρίας Altera [04]
- AT40K [05] της εταιρίας Atmel [08]
- Spartan-3 [35,36] της εταιρίας Xilinx [37]

Και τα αντίστοιχα εργαλεία σχεδίασης CAD είναι:

- Quartus II [02,03]
- IDS – Figaro [06,07]
- ISE [32,33]

Κατόπιν αυτών των αποτελεσμάτων, επιλέχθηκαν η συσκευή Spartan-3E και το αντίστοιχο εργαλείο σχεδίασης CAD ISE, της εταιρίας Xilinx. Οι συγκεκριμένες επιλογές, πέραν του γεγονότος της αξιοπιστίας των επιδόσεων τους, αποτελούν και γνώριμο υλικό για την ερευνητική ομάδα.

Για την καλύτερη παρακολούθηση της διαδικασίας προσομοίωσης και εκτέλεσης, παρακαλείται ο αναγνώστης να μεταβεί στο Παράρτημα Γ και Παράρτημα Δ να δει την αρχιτεκτονική δομή της συσκευής FPGA Spartan-3E και το αντίστοιχο εργαλείο σχεδίασης CAD Xilinx ISE 10.1. Σε κάθε περίπτωση όμως μπορεί να παρακάμψει αυτή τη συμβουλή και να μεταβεί απευθείας στην Ενότητα 5.2.

5.2 Υλοποίηση Κυκλωμάτων VHDL και Εκτέλεση τους στην Προσομοίωση

Να σημειωθεί ότι οι χρόνοι προσομοίωσης για κάθε επερώτηση είναι :

- Κύκλος ρολογιού 20 ns, 10 ns για την ανοδική ακμή και 10 ns για την καθοδική ακμή
- Μετά τα πρώτα 40 ns το σήμα reset γίνεται 1, ενεργοποιείται δηλαδή και δεν επιτρέπει την ανάγνωση των αρχείων
- Στα 100ns το σήμα reset γίνεται 0, απενεργοποιείται δηλαδή και αρχίζει η ανάγνωσης των αρχείων και η λειτουργία των κυκλωμάτων

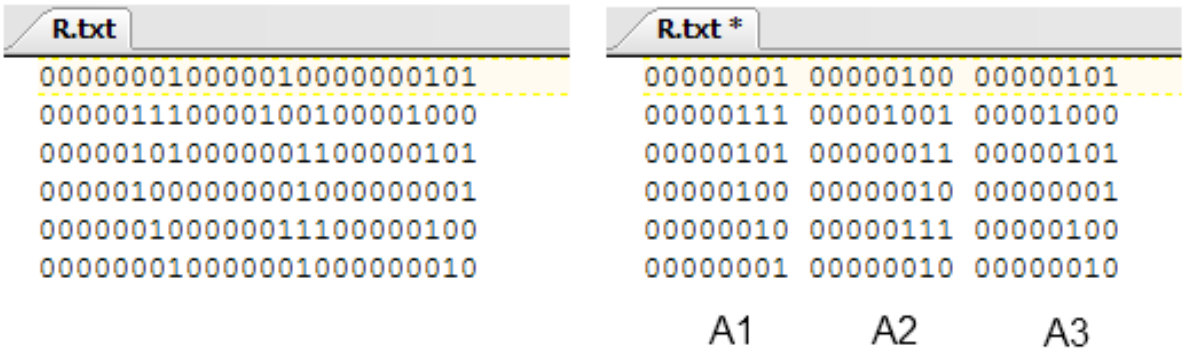
Συγκεκριμένα θα χρησιμοποιηθεί ο ενσωματωμένος προσομοιωτής ISIM του εργαλείου ISE.

5.2.1 Προσομοίωση Επερώτησης Q1

Για να πιστοποιηθεί η θεωρητική προσέγγιση των χρονικών επιδόσεων της επερώτησης Q1(Τύπος 3.14) που αναπτύχθηκε στην Ενότητα 3.3.1 και συγκεκριμένα στην Εικόνα 3.19, θα προσομοιωθεί το λογικό κύκλωμα που υλοποιεί αυτήν την επερώτηση.

Διάβασμα Αρχείου R.txt

Πριν ξεκινήσει η προσομοίωση καλό θα είναι να παρουσιαστεί λίγο το αρχείο **R.txt** το οποίο περιέχει τη σχέση **R** (Εικόνα 3.1) και το οποίο φαίνεται στην Εικόνα 5.4. Όπως μπορεί να διακρίνει ο αναγνώστης τα περιεχόμενα του αρχείου είναι σε δυαδική μορφή και κάθε γραμμή αποτελεί μια πλειάδα της σχέσης R. Για καλύτερη διευκόλυνση στην ανάγνωση παρουσιάζεται στην ίδια εικόνα (Εικόνα 5.4) ο διαχωρισμός των γνωρισμάτων A_1,A_2 και A_3 της σχέσης, από το αρχείο **R.txt ***. Αυτό το αρχείο χρησιμεύει ως τη βάση δεδομένων που θα πρέπει να ελέγξει το λογικό κύκλωμα. Μέσω λοιπόν της προσομοίωσης, θα φορτώνεται σε κάθε κύκλο ρολογιού μία πλειάδα και θα στέλνεται για περαιτέρω επεξεργασία. Το διαχωρισμό των γνωρισμάτων την κάνει αυτόματα το κύκλωμα. Αυτό που πρέπει να κρατήσει ο αναγνώστης στη μνήμη του είναι ότι σε κάθε κύκλο ωρολογίου μία πλειάδα θα στέλνεται για επεξεργασία.



Εικόνα 5.4: Εμφάνιση περιεχομένων αρχείου R.txt, για χρήση του κατά την προσομοίωση της επερώτησης Q1

Προσομοίωση Επερώτησης Q1

Στην Εικόνα 5.5 φαίνεται το αποτέλεσμα της προσομοίωσης του λογικού κυκλώματος της επερώτησης Q1. Όπως μπορεί να παρατηρηθεί μετά τα 100ns όπου το σήμα reset (rst) είναι 1, αρχίζει η λειτουργία του κυκλώματος. Στα πρώτα 20 ns (100 … 120) διαβάζεται η πρώτη πλειάδα 24h010405, όπου το **24** είναι τα bit της πλειάδας το **h** συμβολίζει ότι η εμφάνιση είναι

σε 16δική μορφή, οι υπόλοιποι αριθμοί είναι οι αριθμοί των γνωρισμάτων. Φαίνεται ότι σε κάθε κύκλο ρολογιού διαβάζεται και μια πλειάδα. Η μεταβλητές **y** είναι η μεταβλητή εισόδου (αυτή που διαβάζει την πλειάδα από το αρχείο R.txt) και η μεταβλητή **x** είναι μεταβλητή εξόδου(αυτή που γράφει την πλειάδα στο αρχείο OUT.txt). Τα σήματα **s_valid_bit** και **ok** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου και επιπλέον το σήμα **ok** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT.txt. Το σήμα **eog** δηλώνει τον τερματισμό της ανάγνωσης του αρχείου εισαγωγής.

Όπως φαίνεται λοιπόν οι χρόνοι απόκρισης είναι πολύ γρήγοροι και πιστοποιούν τη θεωρητική προσέγγιση που αναπτύχθηκε στην Ενότητα 3.3.1. Συγκεκριμένα χρειάζονται 2 κύκλοι ρολογιού για την **εισαγωγή - έλεγχο - εγγραφή** μιας πλειάδας, δηλαδή *χρόνος εκτέλεσης* (*latency*) ίσο με 2 κύκλοι ρολογιού (latency = 2), όπως ακριβώς παρουσιάστηκε και στην Εικόνα 3.21.

Συγκεκριμένα:

1. Στον πρώτο κύκλο (100ns – 120ns) γίνεται η ανάγνωση της πλειάδας από το αρχείο R.txt από τη λογική μονάδα FILE READ (Εικόνα 3.21).

2. Στο δεύτερο κύκλο (120ns – 140ns) γίνεται η επεξεργασία και η εγγραφή της πλειάδας (A1, A2, A3) από τις λογικές μονάδες Q1 και FILE WRITE (Εικόνα 3.21).

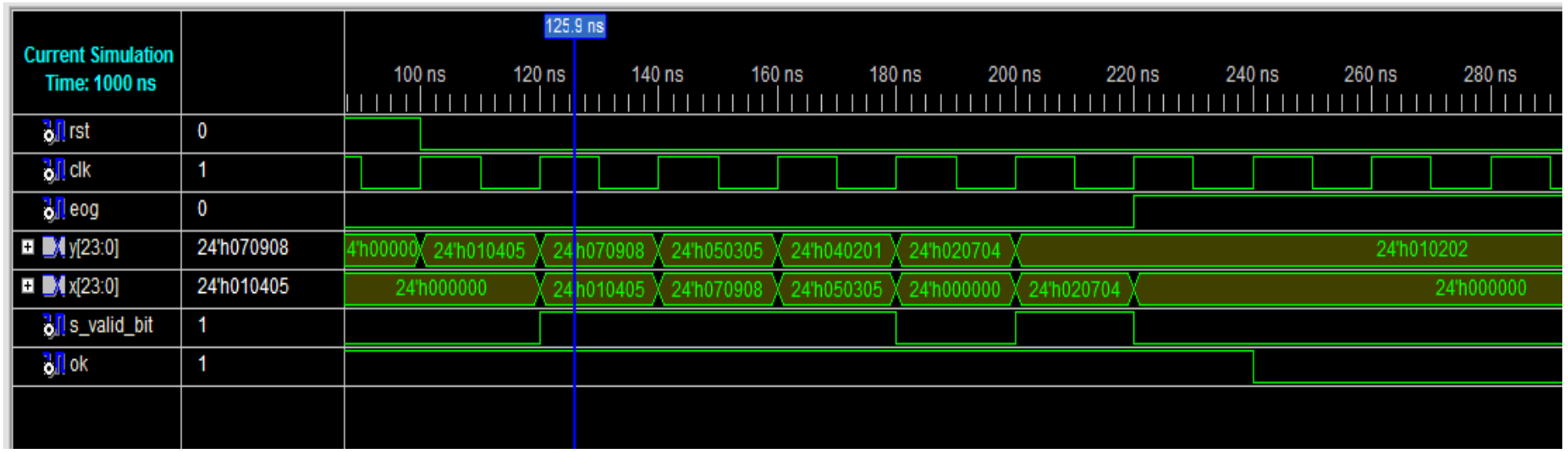
Ο λόγος που η επεξεργασία (εκτέλεση επερώτησης Q1) και η εγγραφή γίνονται σε ένα κύκλο ρολογιού, οφείλεται στη γρήγορη μετάδοση των δεδομένων (σημάτων) από τη δομή διασύνδεσης της συσκευής FPGA.

Σημείωση: Σε κάθε κύκλο ρολογιού υπάρχει *διοχέτευση* (*pipelined*) πλειάδων.

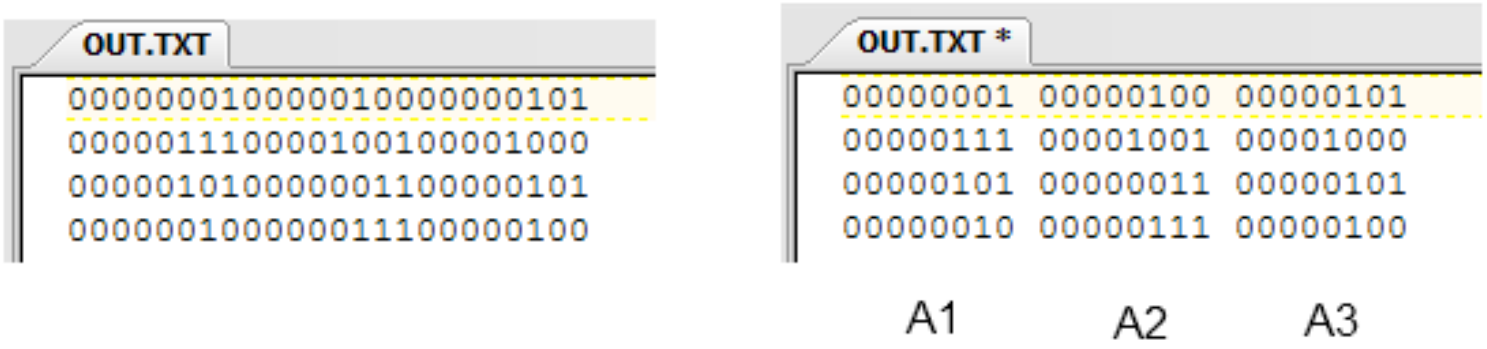
Εγγραφή Αρχείου OUT.txt

Το παραγόμενο αρχείο **OUT.txt** φαίνεται στην Εικόνα 5.6, όπου όπως φαίνεται και τα περιεχόμενα αυτού είναι δυαδικής μορφής. Για καλύτερη ανάγνωση και σε αυτήν την εικόνα τροποποιήθηκε το αρχείο για να μπορέσει ο αναγνώστης να διακρίνει τις τιμές. Όπως μπορεί να διαπιστώσει κάνεις οι δυαδικοί αριθμοί ταιριάζουν απόλυτα με τις τιμές που παρήχθησαν από την επερώτηση Q1 της Εικόνας 3.10.

Ο κώδικας VHDL που υλοποιεί την επερώτηση Q1 βρίσκεται στη Ενότητα Ε.1 στο Παράρτημα Ε.



Εικόνα 5.5: Προσομοίωση λογικού κυκλώματος επερώτησης Q1



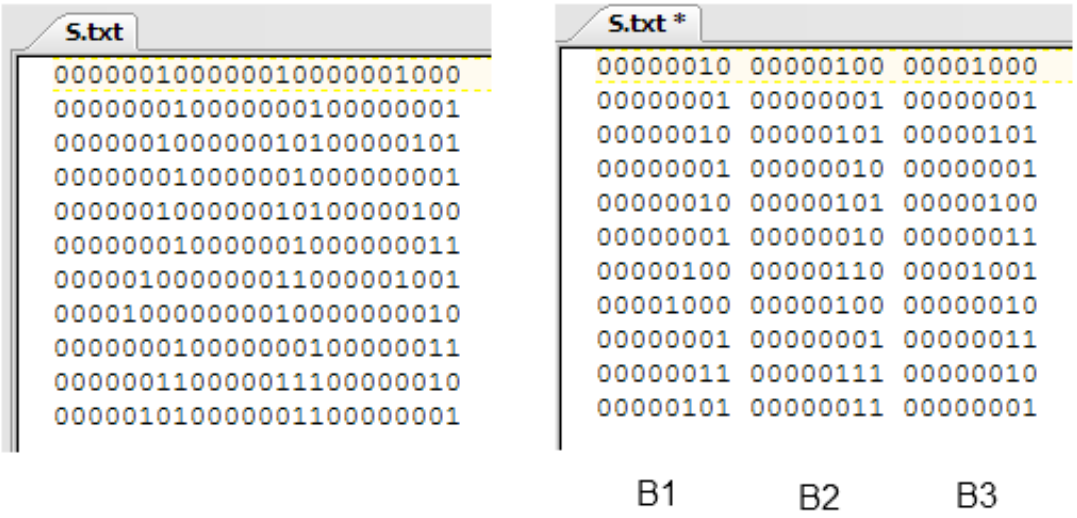
Εικόνα 5.6: Εμφάνιση περιεχομένων αρχείου OUT.txt κατά την προσομοίωση της επερώτησης Q1

5.2.2 Προσομοίωση Επερώτησης Q2

Για να πιστοποιηθεί η θεωρητική προσέγγιση των χρονικών επιδόσεων της επερώτησης Q2(Τύπος 3.16) που αναπτύχθηκε στην Ενότητα 3.3.2 και συγκεκριμένα στην Εικόνα 3.24, θα προσομοιωθεί το λογικό κύκλωμα που υλοποιεί αυτήν την επερώτηση.

Διάβασμα Αρχείου R.txt

Πριν ξεκινήσει η προσομοίωση καλό θα είναι να παρουσιαστεί λίγο το αρχείο **S.txt** το οποίο περιέχει τη σχέση **S** (Εικόνα 3.4) και το οποίο φαίνεται στην Εικόνα 5.7. Όπως μπορεί να διακρίνει ο αναγνώστης τα περιεχόμενα του αρχείου είναι σε δυαδική μορφή και κάθε γραμμή αποτελεί μια πλειάδα της σχέσης S. Για καλύτερη διευκόλυνση στην ανάγνωση παρουσιάζεται στην ίδια εικόνα (Εικόνα 5.7) ο διαχωρισμός των γνωρισμάτων B_1,B_2 και B_3 της σχέσης, από το αρχείο **S.txt ***. Αυτό το αρχείο χρησιμεύει ως τη βάση δεδομένων που θα πρέπει να ελέγξει το λογικό κύκλωμα. Μέσω λοιπόν της προσομοίωσης, θα φορτώνεται σε κάθε κύκλο ρολογιού μία πλειάδα και θα στέλνεται για περαιτέρω επεξεργασία.



Εικόνα 5.7: Εμφάνιση περιεχομένων αρχείου S.txt, για χρήση του κατά την προσομοίωση της επερώτησης Q2

Προσομοίωση Επερώτησης Q2

Στην Εικόνα 5.8 φαίνεται το αποτέλεσμα της προσομοίωσης του λογικού κυκλώματος της επερώτησης Q2. Όπως και με την εκτέλεση της επερώτησης Q1 έτσι και με αυτήν η διαδικασία της επεξεργασίας είναι η ίδια. Βλέπουμε ότι σε κάθε κύκλο ρολογιού διαβάζεται μια πλειάδα. Η

μεταβλητές **y** είναι η μεταβλητή εισόδου (αυτή που διαβάζει την πλειάδα από το αρχείο S.txt) και η μεταβλητή **x** είναι μεταβλητή εξόδου(αυτή που γράφει την πλειάδα στο αρχείο OUT.txt). Τα σήματα **s_valid_bit** και **ok** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου και επιπλέον το σήμα **ok** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT.txt. Το σήμα **eog** δηλώνει τον τερματισμό της ανάγνωσης του αρχείου εισαγωγής.

Όπως φαίνεται οι χρόνοι απόκρισης (και εδώ) είναι πολύ γρήγοροι και πιστοποιούν τη θεωρητική προσέγγιση που αναπτύχθηκε στην Ενότητα 3.3.2. Συγκεκριμένα χρειάζονται 2 κύκλοι ρολογιού για την **εισαγωγή - έλεγχο - εγγραφή** μιας πλειάδας, δηλαδή *χρόνος εκτέλεσης* (*latency*) ίσο με 2 κύκλοι ρολογιού (latency = 2), όπως ακριβώς παρουσιάστηκε και στην Εικόνα 3.26.

Συγκεκριμένα:

1. Στον πρώτο κύκλο (160ns – 180ns) γίνεται η ανάγνωση της πλειάδας από το αρχείο S.txt από τη λογική μονάδα FILE READ (Εικόνα 3.26).

2. Στο δεύτερο κύκλο (180ns – 200ns) γίνεται η επεξεργασία και η εγγραφή της πλειάδας (B1, B2, B3) από τις λογικές μονάδες Q2 και FILE WRITE (Εικόνα 3.26).

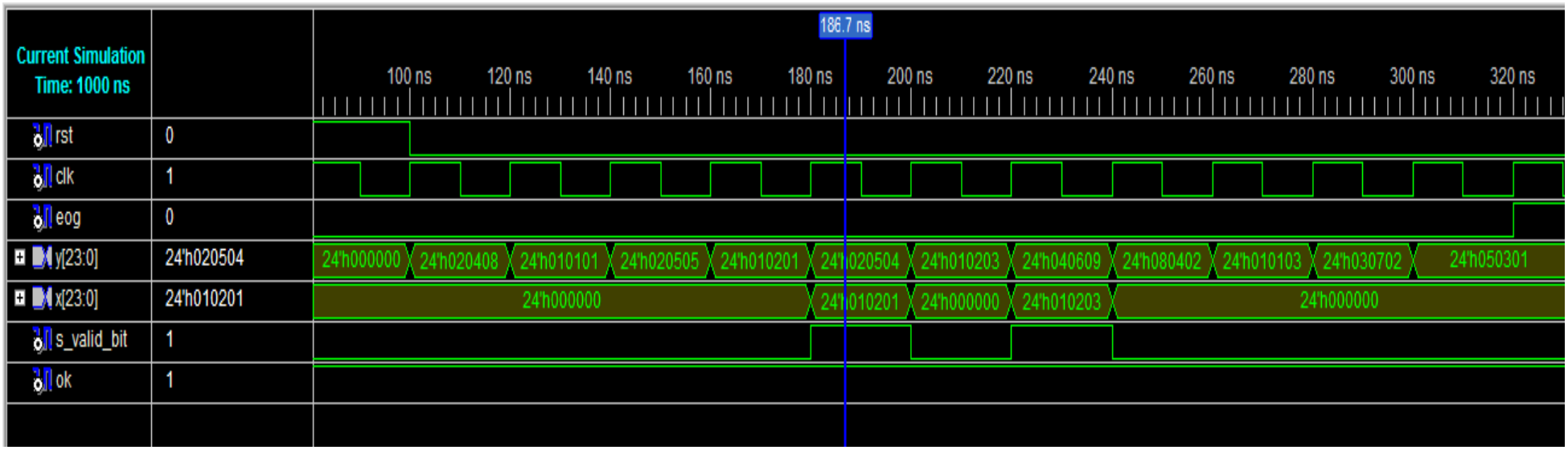
Και εδώ, η δομή διασύνδεσης του FPGA και η λογική σχεδίαση του κυκλώματος κατάφεραν να διευθετήσουν την επεξεργασία μίας πλειάδας, από τη στιγμή της ανάγνωσης της μέχρι τη στιγμή της εγγραφής της, σε δύο κύκλους ρολογιού όπως και με την επερώτηση Q1.

Σημείωση: Σε κάθε κύκλο ρολογιού υπάρχει *διοχέτευση* (*pipelined*) πλειάδων.

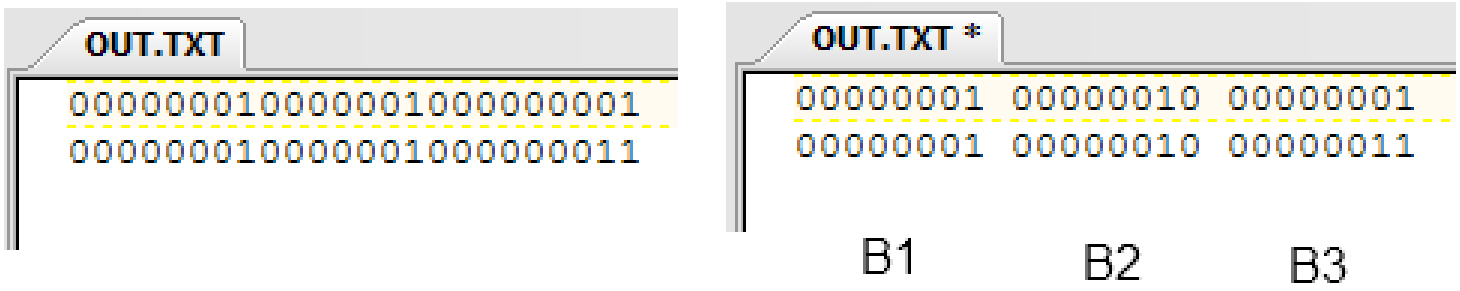
Εγγραφή Αρχείου OUT.txt

Το παραγόμενο αρχείο **OUT.txt** φαίνεται στην Εικόνα 5.9, όπου όπως φαίνεται και τα περιεχόμενα αυτού είναι δυαδικής μορφής. Για καλύτερη ανάγνωση και σε αυτήν την εικόνα τροποποιήθηκε το αρχείο για να μπορέσει ο αναγνώστης να διακρίνει τις τιμές. Όπως μπορεί να διαπιστώσει κάνεις οι δυαδικοί αριθμοί ταιριάζουν απόλυτα με τις τιμές που παρήχθησαν από την επερώτηση Q2 της Εικόνας 3.12.

Ο κώδικας VHDL που υλοποιεί την επερώτηση Q2 βρίσκεται στη Ενότητα Ε.2 στο Παράρτημα Ε.



Εικόνα 5.8: Προσομοίωση λογικού κυκλώματος επερώτησης Q2



Εικόνα 5.9: Εμφάνιση περιεχομένων αρχείου OUT.txt κατά την προσομοίωση της επερώτησης Q2

5.2.3 Προσομοίωση Επερώτησης Q3

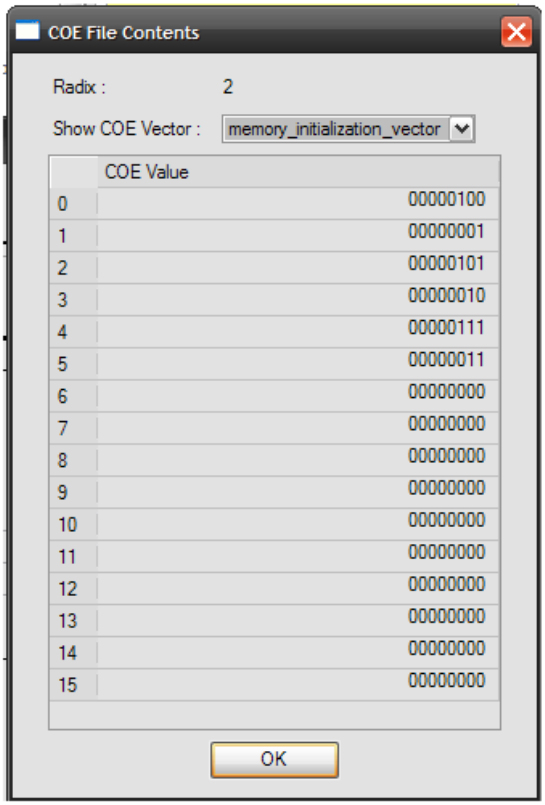
Για να πιστοποιηθεί η θεωρητική προσέγγιση των χρονικών επιδόσεων της επερώτησης Q3(Τύπος 3.18) που αναπτύχθηκε στην Ενότητα 3.3.3 και συγκεκριμένα στην Εικόνα 3.29, θα προσομοιωθεί το λογικό κύκλωμα που υλοποιεί αυτήν την επερώτηση.

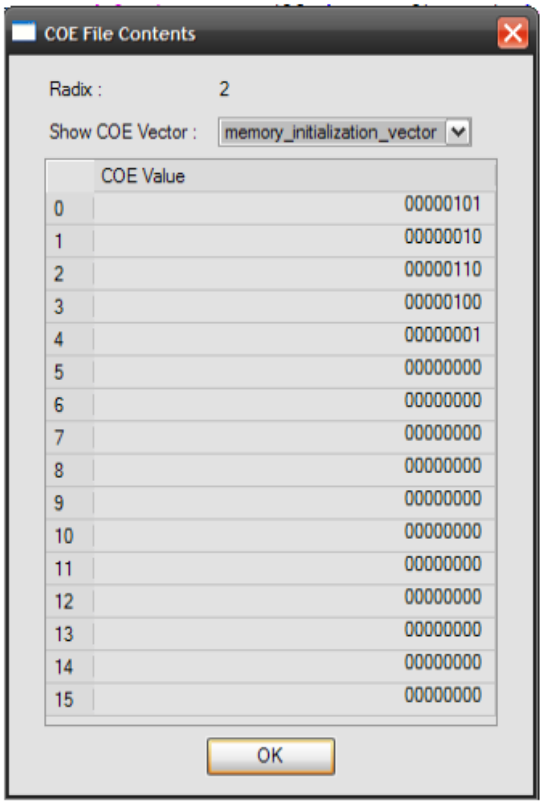
Διάβασμα Αρχείου και Φόρτωμα Μνημών CAM και ROM

Πριν ξεκινήσει η προσομοίωση καλό θα είναι να αναφερθεί λίγο η προεργασία που πρέπει να γίνει πριν την προσομοίωση. Όπως και στις δύο προηγούμενες επερωτήσεις, το κύκλωμα θα διαβάσει το ένα από τα δύο αρχεία, δηλαδή θα διαβάσει τη μια από τις δύο σχέσης. Η δεύτερη σχέση θα βρίσκεται "φορτωμένη" στη συσκευή FPGA. Συγκεκριμένα η δεύτερη σχέση θα πρέπει να φορτωθεί στις μνήμες CAM και ROM. Το πλήθος των μνημών CAM και ROM που χρειάζεται για τη διευθέτηση του κυκλώματος καθορίζεται από την εκτέλεση του αλγόριθμου *CAM_ARRAY* (Ενότητα 4.2.3). Ως είσοδο του αλγόριθμου ορίζονται τα στοιχεία του γνωρίσματος τα οποία ορίζονται στην <συνθήκη> και επιτελούν την πράξη της συνένωσης της δεύτερης σχέσης.

Κατόπιν των τροποποιήσεων που έχουν γίνει κατά την πρόταση λύσης της επερώτησης Q3, ορίζεται ότι η "φορτωμένη" σχέση στη συσκευή θα είναι η **S** και η σχέση που θα διαβάζεται θα είναι η **R**. Θα διαβαστεί το αρχείο **R.txt** το οποίο περιέχει τη σχέση **R** (Εικόνα 3.1) και το οποίο φαίνεται στην Εικόνα 5.4. Για τη δεύτερη σχέση S ο αλγόριθμος *CAM_ARRAY* όρισε ότι θα πρέπει να υπάρχουν δύο μνήμες CAM και δύο μνήμες ROM (Εικόνα 3.31). Για αυτό το λόγο θα πρέπει να φορτωθούν στις αντίστοιχες μνήμες τα δεδομένα. Στην Εικόνα 5.10 φαίνονται τα περιεχόμενα των μνημών **camA** και **camB** όπως αυτά φορτώθηκαν από το αρχείο αρχικοποίησης μνήμών ***.coe**. Ο αναγνώστης μπορεί να επιβεβαιώσει ότι τα περιεχόμενα των μνημών είναι ίδια με αυτά που ορίστηκαν στην Ενότητα 3.3.3 και συγκεκριμένα στην Εικόνα 3.33.

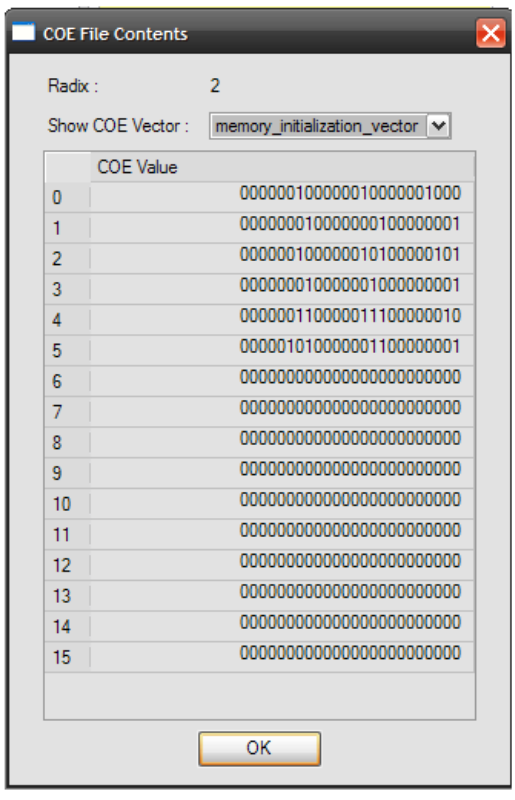
Ενώ στην Εικόνα 5.11 φαίνονται τα περιεχόμενα των μνημών **romA** και **romB** όπως και αυτά ορίστηκαν στην Εικόνα 3.33.

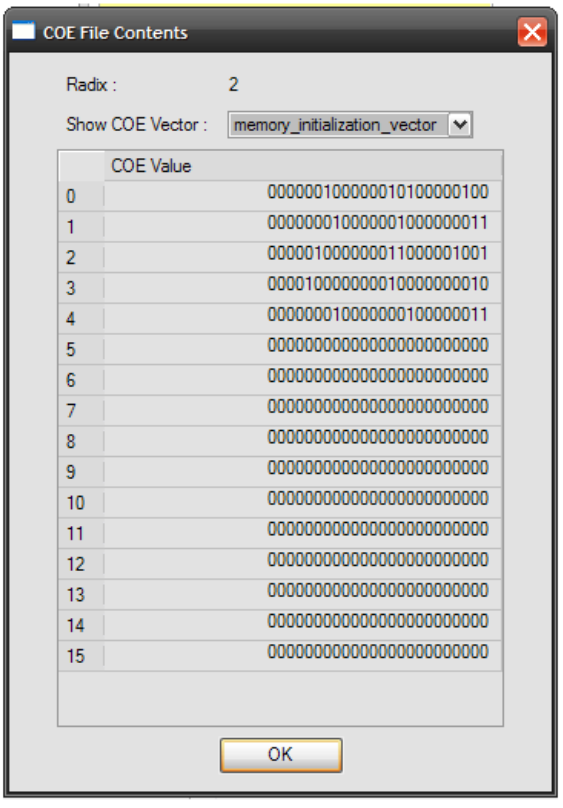




camA camB

Εικόνα 5.10: Εμφάνιση περιεχομένων μνημών camA και camB





romA romB

Εικόνα 5.11: Εμφάνιση περιεχομένων μνημών romA και romB

Προσομοίωση Επερώτησης Q3

Στην Εικόνα 5.12 φαίνεται το αποτέλεσμα της προσομοίωσης του λογικού κυκλώματος της επερώτησης Q3. Βλέπουμε ότι σε κάθε κύκλο ρολογιού διαβάζεται μια πλειάδα. Η μεταβλητές **y** είναι η μεταβλητή εισόδου, αυτή που διαβάζει την πλειάδα από το αρχείο R.txt. Η μεταβλητή **x_a** είναι μεταβλητή εξόδου του πρώτου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT1.txt. Η μεταβλητή **x_b** είναι μεταβλητή εξόδου του δεύτερου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT2.txt. Τα σήματα **s_valid_bit_a** και **ok_a** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του πρώτου κυκλώματος και επιπλέον το σήμα **ok_a** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT1.txt. Τα σήματα **s_valid_bit_b** και **ok_b** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του δεύτερου κυκλώματος και επιπλέον το σήμα **ok_b** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT2.txt. Το σήμα **eog** δηλώνει τον τερματισμό της ανάγνωσης του αρχείου εισαγωγής.

Όπως φαίνεται λοιπόν οι χρόνοι απόκρισης είναι γρήγοροι και πιστοποιούν τη θεωρητική προσέγγιση που αναπτύχθηκε στην Ενότητα 3.3.3. Συγκεκριμένα χρειάζονται 7 κύκλοι ρολογιού για την **εισαγωγή - έλεγχο - εγγραφή** μιας πλειάδας, δηλαδή *χρόνος εκτέλεσης* (*latency*) ίσο με 7 κύκλοι ρολογιού (latency = 7), όπως ακριβώς παρουσιάστηκε και στην Εικόνα 3.34.

Συγκεκριμένα:

1. Στον πρώτο κύκλο (160ns – 180ns) η πλειάδα **24'h040201** φορτώνεται και γίνεται η ανάγνωση της από το αρχείο R.txt.

2. Στο δεύτερο κύκλο (180ns – 200ns) γίνεται η φόρτωση του γνωρίσματος A_2 (h02) της πλειάδας, στις μνήμες CAMA και CAMB για να διαπιστωθεί αν ταυτίζεται με κάποιο από τα στοιχεία τους (Εικόνα 3.35).

3. Στον τρίτο κύκλο (200ns – 220ns) γίνεται η κωδικοποίηση της 16bit παραγόμενης διεύθυνσης από τις μνήμες CAM, στη λογική μονάδα CODER σε μορφή 4bit για να διαβαστεί στη μνήμη ROM (Εικόνα 3.35).

4. Στον τέταρτο κύκλο (220ns – 240ns) γίνεται η εκμαίευση των πλειάδων της σχέσης S από τις μνήμες ROMA και ROMB και προωθούνται για περαιτέρω επεξεργασία στη μονάδα Process (Εικόνα 3.35).

5. Στον πέμπτο και έκτο κύκλο (240ns – 260ns και 260ns – 280ns) γίνεται η εκτέλεση των επερωτήσεων Q3A και Q3B και η παραγωγή των ζητούμενων πλειάδων (Εικόνα 3.34).

6. Και στον έβδομο κύκλο (280ns – 300ns) γίνεται η εγγραφή της πλειάδας (A1, A2, A3, B1, B2, B3) στα αρχεία OUT1.txt και OUT2.txt, από τις λογικές μονάδες WRITE FILE A και WRITE FILE B (Εικόνα 3.34).

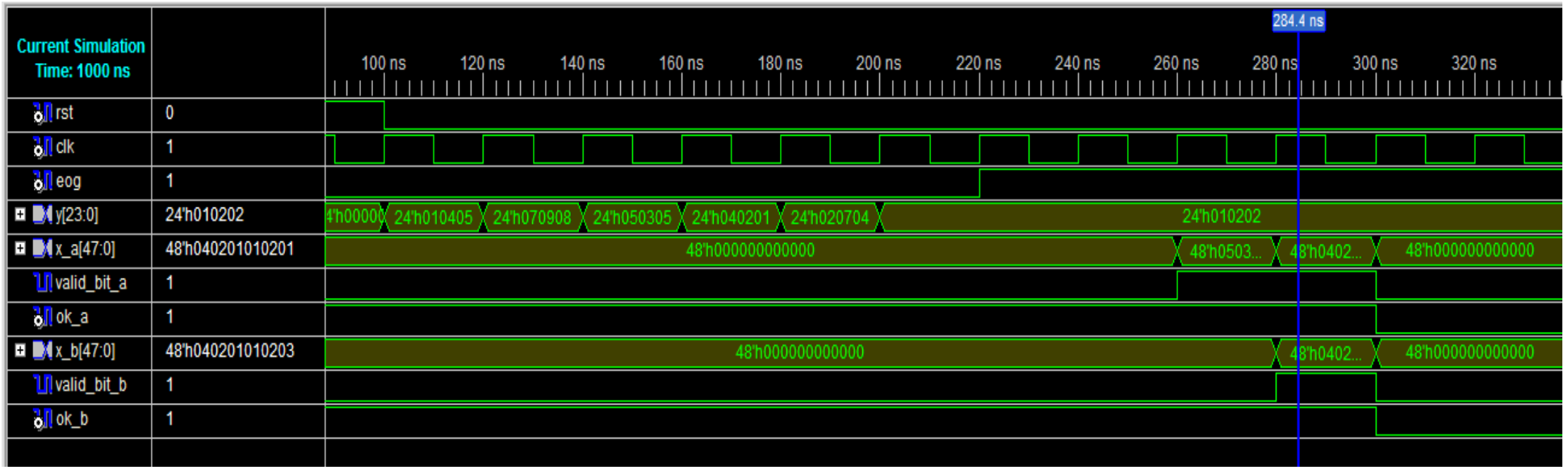
Και εδώ, η δομή διασύνδεσης του FPGA και η λογική σχεδίαση του κυκλώματος κατάφεραν να διευθετήσουν την επεξεργασία μίας πλειάδας, από τη στιγμή της ανάγνωσης της μέχρι τη στιγμή της εγγραφής της, σε επτά κύκλους ρολογιού. Και αυτό συμβαίνει και με τα δύο υποκυκλώματα παράλληλα (Εικόνα 3.34). Συγκεκριμένα στην μπλε γραμμή (285 ns) φαίνεται ότι και τα δύο κυκλώματα δέχονται την πλειάδα 24'h040201 ως αποτέλεσμα. Το πρώτο κύκλωμα δέχεται τη γενικευμένη πλειάδα 24'h040201010201 και το δεύτερο κύκλωμα τη γενικευμένη πλειάδα 24'h040201010203. Ένα κύκλο ρολογιού πριν, το πρώτο κύκλωμα δέχτηκε και την πλειάδα 24'h050305. Οι αριθμοί αυτοί ταυτίζονται πλήρως με τις Εικόνες 3.14 και 5.13.

Σημείωση: Σε κάθε κύκλο ρολογιού υπάρχει *διοχέτευση* (*pipelined*) πλειάδων.

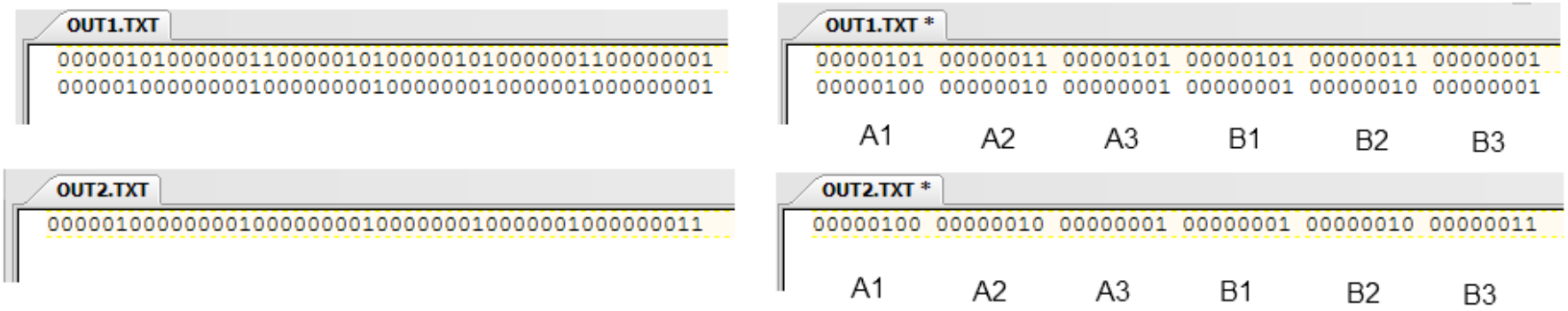
Εγγραφή Αρχείων OUT1.txt και OUT2.txt

Το παραγόμενα αρχείο **OUT1.txt** και **OUT2.txt** φαίνονται στην Εικόνα 5.13, όπου όπως φαίνεται και τα περιεχόμενα αυτών είναι σε δυαδική μορφή. Για καλύτερη ανάγνωση και σε αυτήν την εικόνα τροποποιήθηκε το αρχείο για να μπορέσει ο αναγνώστης να διακρίνει τις τιμές. Όπως μπορεί να διαπιστώσει κάνεις οι δυαδικοί αριθμοί ταιριάζουν απόλυτα με τις τιμές που παρήχθησαν από την επερώτηση Q3 της Εικόνας 3.14.

Ο κώδικας VHDL που υλοποιεί την επερώτηση Q3 βρίσκεται στη Ενότητα Ε.3 στο Παράρτημα Ε.



Εικόνα 512: Προσομοίωση λογικού κυκλώματος επερώτησης Q3



Εικόνα 5.13: Εμφάνιση περιεχομένων αρχείου OUT1.txt και OUT2.txt κατά την προσομοίωση της επερώτησης Q3

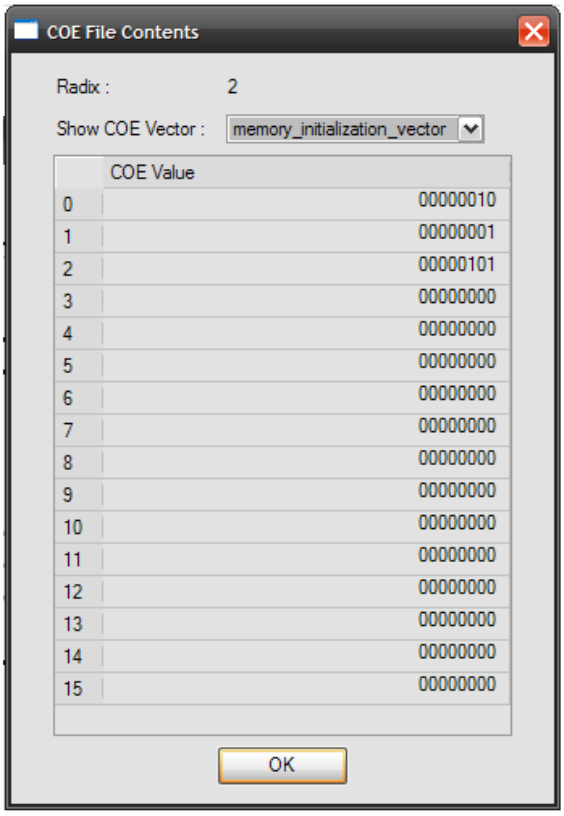
5.2.4 Προσομοίωση Επερώτησης Q4

Για να πιστοποιηθεί η θεωρητική προσέγγιση των χρονικών επιδόσεων της επερώτησης Q4(Τύπος 3.20) που αναπτύχθηκε στην Ενότητα 3.3.4 και συγκεκριμένα στην Εικόνα 3.39, θα προσομοιωθεί το λογικό κύκλωμα που υλοποιεί αυτήν την επερώτηση.

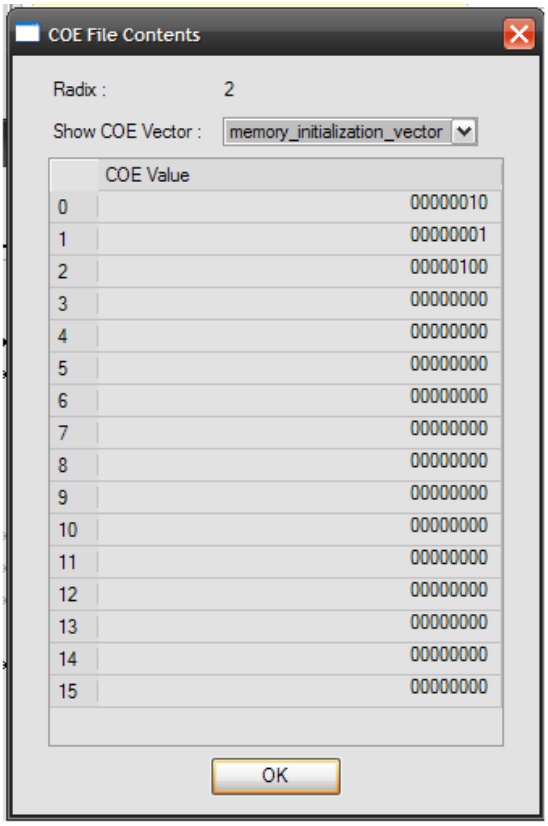
Διάβασμα Αρχείου και Φόρτωμα Μνημών CAM και ROM

Όπως και με την επερώτηση Q3 έτσι και με αυτήν την επερώτηση θα πρέπει να καθοριστούν οι συνθήκες λειτουργίας της συσκευής FPGA και να αρχικοποιηθούν οι μνήμες CAM και RAM.

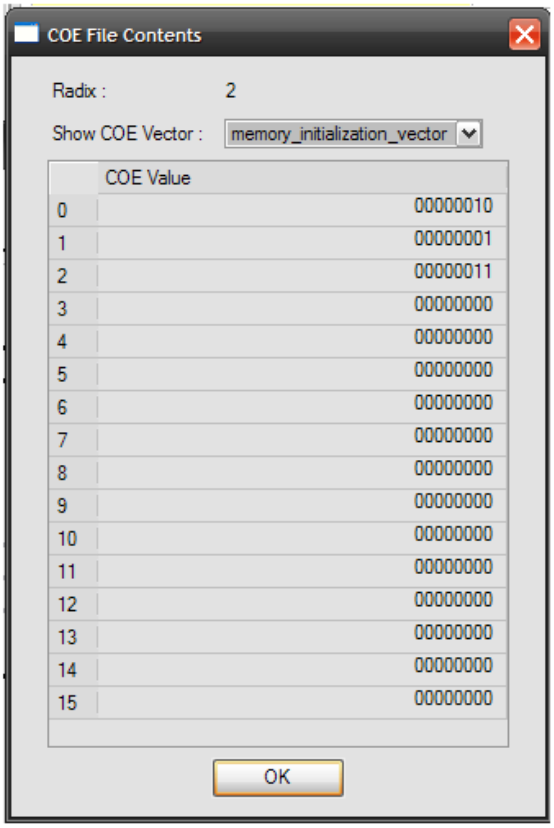
Κατόπιν των τροποποιήσεων που έχουν γίνει κατά την πρόταση λύσης της επερώτησης Q4, ορίζεται ότι η "φορτωμένη" σχέση στη συσκευή θα είναι η **S** και η σχέση που θα διαβάζεται θα είναι η **R**. Θα διαβαστεί το αρχείο **R.txt** το οποίο περιέχει τη σχέση **R** (Εικόνα 3.1) και το οποίο φαίνεται στην Εικόνα 5.4. Για τη δεύτερη σχέση S ο αλγόριθμος *CAM_ARRAY* όρισε ότι θα πρέπει να υπάρχουν τέσσερις μνήμες CAM και τέσσερις μνήμες ROM (Εικόνα 3.40). Για αυτό το λόγο θα πρέπει να φορτωθούν στις αντίστοιχες μνήμες τα δεδομένα. Στην Εικόνα 5.14 φαίνονται τα περιεχόμενα των μνημών **camA**, **camB**, **camC** και **camD** όπως αυτά φορτώθηκαν από το αρχείο αρχικοποίησης μνήμών ***.coe**. Ο αναγνώστης μπορεί να επιβεβαιώσει ότι τα περιεχόμενα των μνημών είναι ίδια με αυτά που ορίστηκαν στην Ενότητα 3.3.4 και συγκεκριμένα στην Εικόνα 3.42.



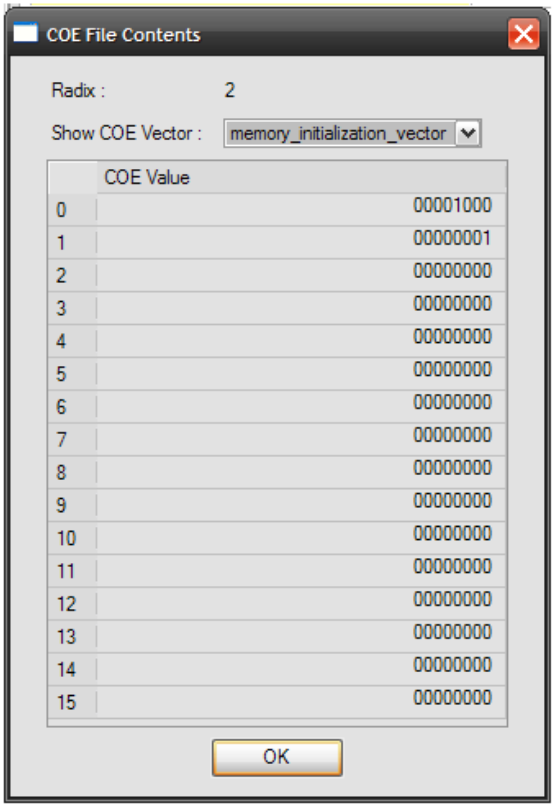
camA



camB



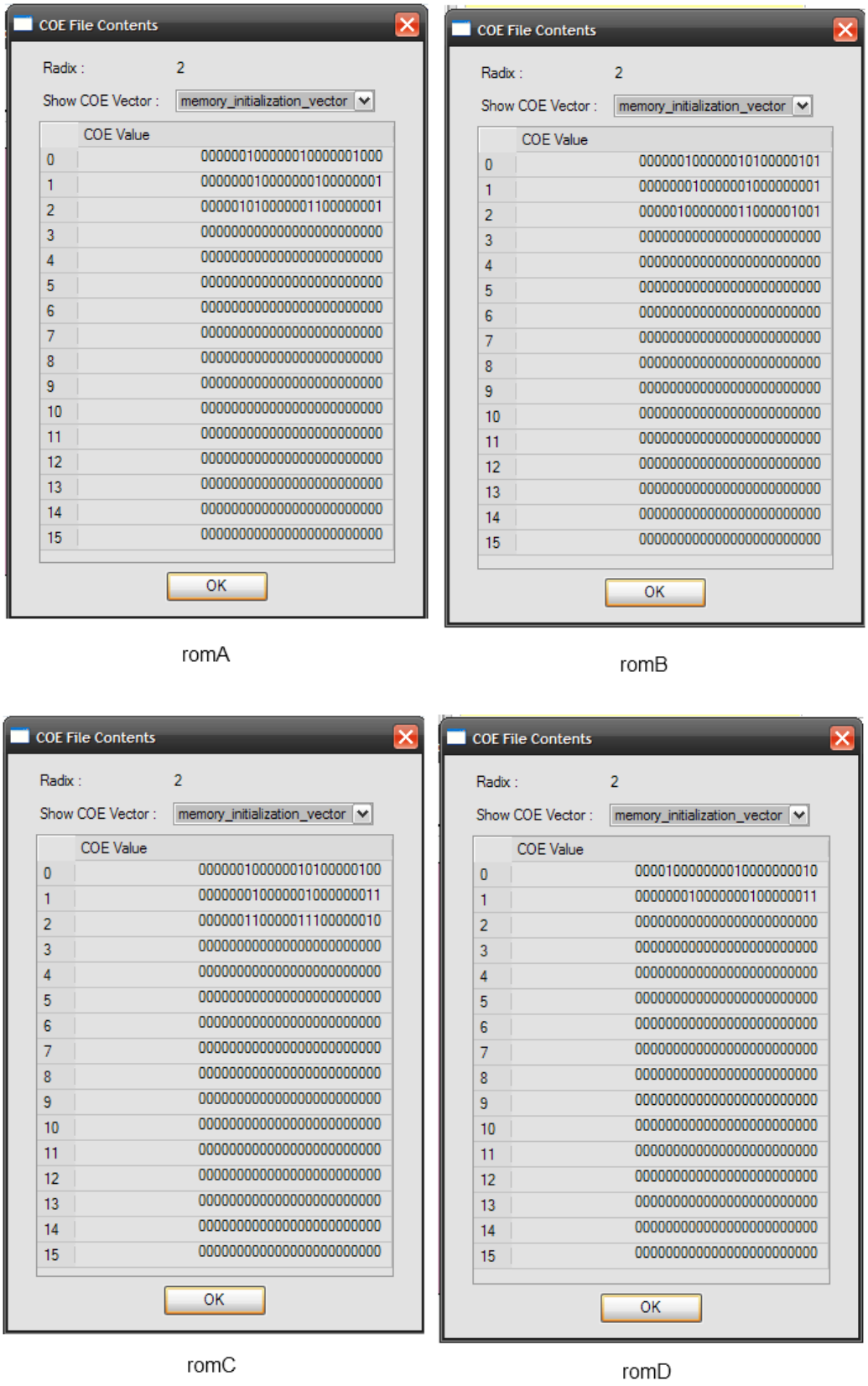
camC



camD

Εικόνα 5.14: Εμφάνιση περιεχομένων μνημών camA, camB, camC και camD

Ενώ στην Εικόνα 5.15 φαίνονται τα περιεχόμενα των μνημών **romA**, **romB**, **romC** και **romD** όπως και αυτά ορίστηκαν στην Εικόνα 3.42.



Εικόνα 5.15: Εμφάνιση περιεχομένων μνημών romA, romB, romC και romD

Προσομοίωση Επερώτησης Q4

Στην Εικόνα 5.16 φαίνεται το αποτέλεσμα της προσομοίωσης του λογικού κυκλώματος της επερώτησης Q4. Όπως και με την εκτέλεση της επερώτησης Q1 έτσι και με αυτήν η διαδικασία της επεξεργασίας είναι η ίδια. Βλέπουμε ότι σε κάθε κύκλο ρολογιού διαβάζεται μια πλειάδα. Η μεταβλητές **y** είναι η μεταβλητή εισόδου, αυτή που διαβάζει την πλειάδα από το αρχείο R.txt. Η μεταβλητή **x_a** είναι μεταβλητή εξόδου του πρώτου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT1.txt. Η μεταβλητή **x_b** είναι μεταβλητή εξόδου του δεύτερου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT2.txt. Η μεταβλητή **x_c** είναι μεταβλητή εξόδου του τρίτου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT3.txt. Η μεταβλητή **x_d** είναι μεταβλητή εξόδου του τέταρτου κυκλώματος, αυτή που γράφει την πλειάδα στο αρχείο OUT4.txt. Τα σήματα **s_valid_bit_a** και **ok_a** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του πρώτου κυκλώματος και επιπλέον το σήμα **ok_a** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT1.txt. Τα σήματα **s_valid_bit_b** και **ok_b** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του δεύτερου κυκλώματος και επιπλέον το σήμα **ok_b** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT2.txt. Τα σήματα **s_valid_bit_c** και **ok_c** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του τρίτου κυκλώματος και επιπλέον το σήμα **ok_c** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT3.txt. Τα σήματα **s_valid_bit_d** και **ok_d** πιστοποιούν ποια πλειάδα θα γραφεί στο αρχείο εξόδου του τέταρτου κυκλώματος και επιπλέον το σήμα **ok_d** πιστοποιεί και μέχρι πότε θα πρέπει να γίνεται εγράφη στο αρχείο OUT4.txt. Το σήμα **eog** δηλώνει τον τερματισμό της ανάγνωσης του αρχείου εισαγωγής.

Όπως φαίνεται λοιπόν οι χρόνοι απόκρισης είναι γρήγοροι και πιστοποιούν τη θεωρητική προσέγγιση που αναπτύχθηκε στην Ενότητα 3.3.4. Συγκεκριμένα χρειάζονται 6 κύκλοι ρολογιού για την **εισαγωγή - έλεγχο - εγγραφή** μιας πλειάδας, δηλαδή *χρόνος εκτέλεσης* (*latency*) ίσο με 6 κύκλοι ρολογιού (latency = 6), όπως ακριβώς παρουσιάστηκε και στην Εικόνα 3.43.

Συγκεκριμένα:

1. Στον πρώτο κύκλο (100ns – 120ns) η πλειάδα **24'h010405** φορτώνεται και γίνεται η ανάγνωση της από το αρχείο R.txt.

2. Στο δεύτερο κύκλο (120ns – 140ns) γίνεται η φόρτωση του γνωρίσματος A_1 (h01) της πλειάδας, στις μνήμες CAMA, CAMB, CAMC και CAMD για να διαπιστωθεί αν ταυτίζεται με κάποιο από τα στοιχεία τους (Εικόνα 3.44).

3. Στον τρίτο κύκλο (140ns – 1600ns) γίνεται η κωδικοποίηση της 16bit παραγόμενης διεύθυνσης από τις μνήμες CAM, στη λογική μονάδα CODER σε μορφή 4bit για να διαβαστεί στη μνήμη ROM (Εικόνα 3.44).

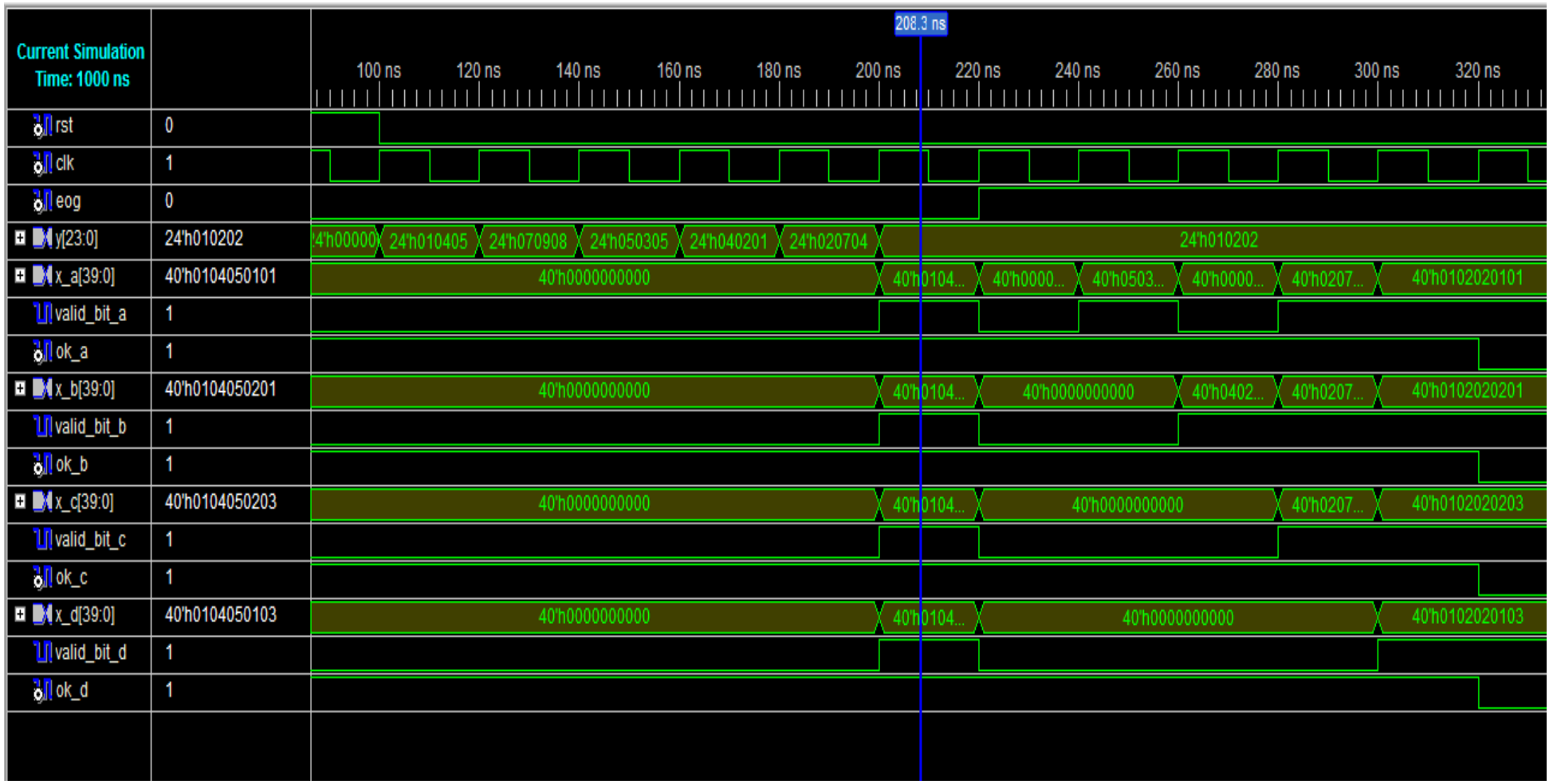
4. Στον τέταρτο κύκλο (160ns – 180ns) γίνεται η εκμαίευση των πλειάδων της σχέσης S από τις μνήμες ROMA, ROMB, ROMC και ROMD και προωθούνται για περαιτέρω επεξεργασία στη μονάδα Process (Εικόνα 3.44).

5. Στον πέμπτο (180ns – 200ns) γίνεται η εκτέλεση των επερωτήσεων Q4A, Q4A, Q4C και Q4D και η παραγωγή των ζητούμενων πλειάδων (Εικόνα 3.43).

6. Και στον έκτο κύκλο (200ns – 220ns) γίνεται η εγγραφή της πλειάδας (A1, A2, A3, B2, B3) στα αρχεία OUT1.txt, OUT2.txt, OUT3.txt και OUT4.txt , από τις λογικές μονάδες WRITE FILE A, WRITE FILE B, WRITE FILE C και WRITE FILE D (Εικόνα 3.43).

Και εδώ, η δομή διασύνδεσης του FPGA και η λογική σχεδίαση του κυκλώματος κατάφεραν να διευθετήσουν την επεξεργασία μίας πλειάδας, από τη στιγμή της ανάγνωσης της μέχρι τη στιγμή της εγγραφής της, σε έξι κύκλους ρολογιού. Και αυτό συμβαίνει και με τα τέσσερα υποκυκλώματα (Εικόνα 3.43) παράλληλα. Συγκεκριμένα στην μπλε γραμμή (208 ns) φαίνεται ότι και τα τέσσερα κυκλώματα δέχονται την πλειάδα 24'h010405 ως αποτέλεσμα. Το πρώτο κύκλωμα δέχεται τη γενικευμένη πλειάδα 24'h0104050101. Το δεύτερο κύκλωμα τη γενικευμένη πλειάδα 24'h0104050201. Το τρίτο κύκλωμα δέχεται τη γενικευμένη πλειάδα 24'h0104050203. Και το τέταρτο κύκλωμα τη γενικευμένη πλειάδα 24'h0104050103. Αυτό το παράδειγμα αναφέρθηκε για να αναδείξει και στην προσομοίωση αυτά που λέγονται στη θεωρία, τα κυκλώματα λειτουργού παράλληλα! Οι παραγόμενοι αριθμοί ταυτίζονται πλήρως με τις Εικόνες 3.16 και 5.17.

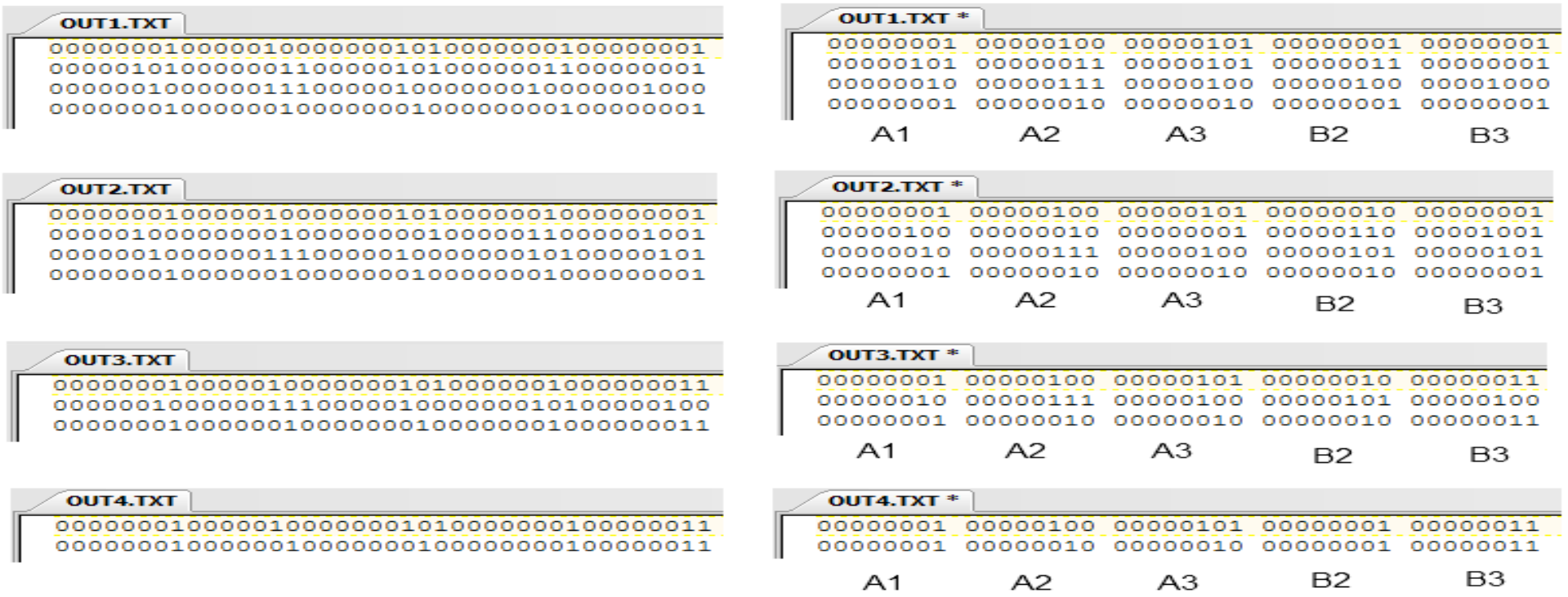
Σημείωση: Σε κάθε κύκλο ρολογιού υπάρχει *διοχέτευση* (*pipelined*) πλειάδων.



Εικόνα 516: Προσομοίωση λογικού κυκλώματος επερώτησης Q4.

Εγγραφή Αρχείων OUT1.txt, OUT2.txt, OUT3.txt και OUT4.txt

Το παραγόμενα αρχείο **OUT1.txt, OUT2.txt, OUT3.txt** και **OUT4.txt** φαίνονται στην Εικόνα 5.17, όπως φαίνεται και τα περιεχόμενα αυτών είναι σε δυαδική μορφή. Για καλύτερη ανάγνωση και σε αυτήν την εικόνα τροποποιήθηκε το αρχείο για να μπορέσει ο αναγνώστης να διακρίνει τις τιμές. Όπως μπορεί να διαπιστώσει κάνεις οι δυαδικοί αριθμοί ταιριάζουν απόλυτα με τις τιμές που παρήχθησαν από την επερώτηση Q4 της Εικόνας 3.16.



Εικόνα 5.17: Εμφάνιση περιεχομένων αρχείων OUT1.txt, OUT2.txt, OUT3.txt και OUT4.txt κατά την προσομοίωση της επερώτησης Q4

Ο κώδικας VHDL που υλοποιεί την επερώτηση Q4 βρίσκεται στη Ενότητα Ε.4 στο Παράρτημα Ε.

5.3 Υλοποίηση Κυκλωμάτων VHDL και Εκτέλεση τους στο Υλικό του FPGA

Μετά την υλοποίηση των ερωτημάτων στην προσομοίωση επιχειρήθηκε η υλοποίηση τους απευθείας στο υλικό. Το σενάριο περιλάμβανε μία συσκευή FPGA η οποία θα διενεργούσε την επεξεργασία των δεδομένων και έναν Η/Υ από τον οποίο θα στέλνονταν και θα αποθηκευόντουσαν τα παραγόμενα δεδομένα. Για τη διευθέτηση της επικοινωνίας των δύο συσκευών χρησιμοποιήθηκε το πρωτόκολλο επικοινωνίας UART σε μορφή VHDL του πρότυπου RS232 (Ενότητα 4.4.1). Το πρότυπο επικοινωνίας RS232 είναι της μορφής που φαίνεται στην Εικόνα 4.29, όπου η συχνότητα αποστολής και λήψης των δεδομένων από τη θύρα επικοινωνίας COM1 του Η/Υ, εξαρτάται από τον ορισμό του στη διεπαφή. Στην περίπτωση που επιχειρήθηκε, έγινε χρήση του Hyper Terminal όπου η συχνότητα αποστολής και λήψης των δεδομένων κυμαίνεται από 114 bit μέχρι 921600bit το δευτερόλεπτο.

Από την προσπάθεια που υλοποιήθηκε διαπιστώθηκε ότι υπήρχαν δυσκολίες τόσο στη διευθέτηση του συγχρονισμού του πρωτοκόλλου επικοινωνίας UART (Ενότητα 4.4.2) με την επεξεργαστική μηχανή (αυτή που επεξεργάζονταν τα δεδομένα), αλλά και κάποια φαινόμενα μη σύνθεσης των κυκλωμάτων από το εργαλείο ISE στη συσκευή FPGA (δεν έγινε σύνθεση του παραγόμενου VHDL κώδικα). Το δεύτερο πρόβλημα λύθηκε με την αλλαγή και τη συγγραφή νέου κώδικα στη VHDL με σκοπό την παραγωγή των επιθυμητών αποτελεσμάτων. Το πρώτο πρόβλημα ακόμη δεν έχει λυθεί, όπου μια από τις προσπάθειες που έχουν γίνει είναι η χρήση άλλου κυκλώματος στη VHDL που υλοποιεί το πρωτόκολλο επικοινωνίας UART. Και εκεί όμως διαπιστώθηκε το ίδιο πρόβλημα συγχρονισμού, ως εκ τούτου η προσπάθεια εγκαταλείφθηκε προσωρινά μέχρι τη διευθέτηση της παρουσίασης και ολοκλήρωσης της μεταπτυχιακής διατριβής. Μετά το πέρας των διαδικαστικών ολοκλήρωσης της μεταπτυχιακής διατριβής η προσπάθεια θα συνεχιστεί με πιο έντονους ρυθμούς.

Μια παρατήρηση είναι ότι στην προσομοίωση, τη διευθέτηση της επικοινωνίας και του συγχρονισμού της εισόδου και της εξόδου την υλοποιούσε αυτόματα το πρόγραμμα ISE, το αποτέλεσμα ήταν η λειτουργία των κυκλωμάτων να είναι σωστή. Το ίδιο συνέβη και με τον κώδικα VHDL που γράφτηκε πριν της αλλαγές. Ο παραγόμενος κώδικάς μπορεί να είναι σωστός κατά την προσομοίωση τόσο σε συντακτικό έλεγχο όσο και σε λογικό, αλλά κατά τη σύνθεση του σε μια συσκευή FPGA μπορεί να αλλάζει λόγω της ιδιομορφίας της κάθε αρχιτεκτονικής.

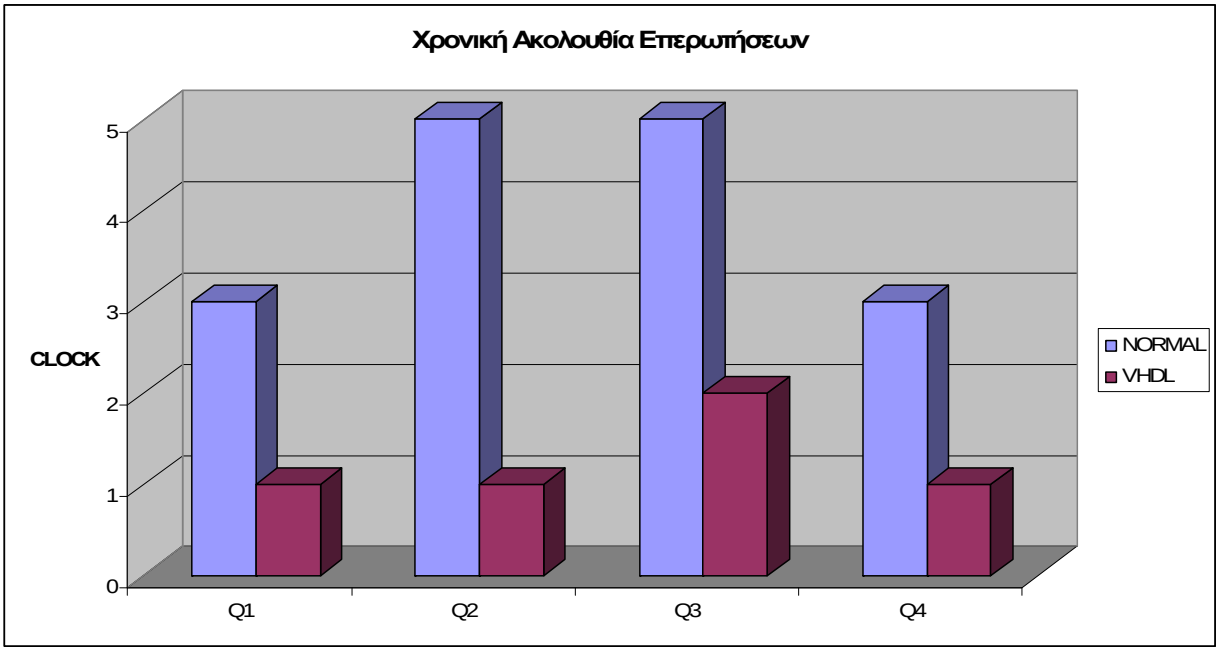
Συγκεκριμένα η κάθε αρχιτεκτονική μπορεί να διαφέρει από συσκευή σε συσκευή, ιδιαίτερα το κομμάτι της σύνθεσης και δρομολόγησης.

Μια ακόμη παρατήρηση είναι ότι ο προβλέψιμος χρόνος της διευθέτησης της υλοποίησης στο υλικό ήταν εντελώς άστοχος. Ο χρόνος που τελικά απαιτείται για τη διευθέτηση της επικοινωνίας και της λειτουργίας του πειράματος σε πραγματικές συνθήκες και με τα επιθυμητά αποτελέσματα, είναι ο διπλάσιος αν όχι ο τριπλάσιος του αρχικού (ένας μήνας)!

Δεδομένου των παραπάνω η υλοποίηση στο υλικό έχει προσωρινά "παγώσεις" μιας και (όπως αναφέρθηκε) εκκρεμεί η παράδοση της παρούσης μεταπτυχιακής διατριβής.

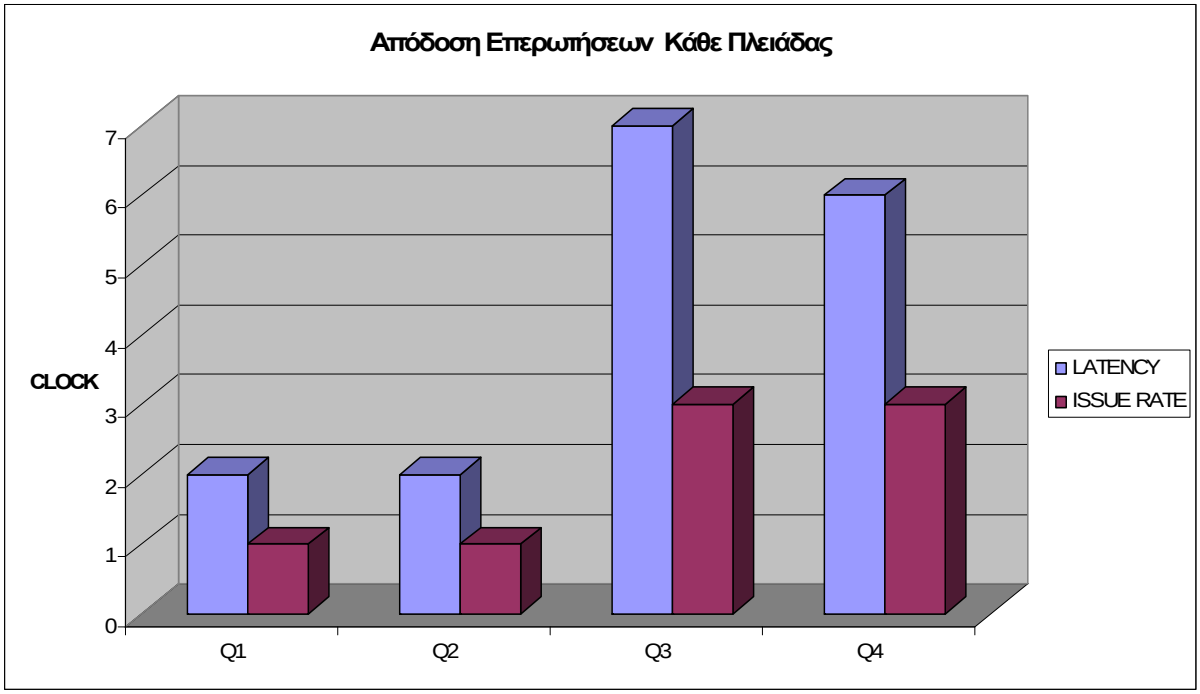
5.4 Εκτίμηση Αποτελεσμάτων

Μετά την εκτέλεση και των προσομοιώσεων στα λογικά κυκλώματα που υλοποιούσαν τις επερωτήσεις Q1, Q2, Q3 και Q4, είναι εμφανές ότι η θεωρητική προσέγγιση που αναπτύχθηκε στην Ενότητα 3.3, ταυτίζεται απόλυτα με τα δεδομένα αποτελέσματα των προσομοιώσεων. Στην Εικόνα 5.18 φαίνονται οι θεωρητικοί χρόνοι συγκρίσει με αυτούς που πραγματοποιήθηκαν στις προσομοιώσεις για κάθε επερώτηση, η διαφορά είναι εμφανής!



Εικόνα 5.18: Χρονική ακολουθία επερωτήσεων Q1, Q2, Q3 και Q4 σε κανονικές συνθήκες αλλά και με χρήση γλώσσας VHDL

Οι χρόνοι εκτέλεσης (latency) ταυτίζονται πλήρως με αυτούς της θεωρητικής προσέγγισης, όπου αυτό αποδεικνύει την ικανότητα των συσκευών FPGA να διεκπεραιώνουν τέτοιες εργασίες γρήγορα και αξιόπιστα. Στην Εικόνα 5.19 φαίνονται οι χρόνοι εκτέλεσης (latency) και τα ζητούμενα ποσοστά (*issue rate*) των επερωτήσεων κατά την εκτέλεση τους στην προσομοίωση. Οι παραγόμενοι χρόνοι είναι εντυπωσιακοί, τέτοιοι ώστε να θέτουν της βάσης για σκέψη για μελλοντική χρήση τους τόσο ως σύστημα υποστήριξης ΣΒΔ αλλά και όσο ως ανεξάρτητες μονάδες επεξεργασίας και διαχείρισης. Η χρόνοι εκτέλεσης που έχουν επιτευχθεί δεν αφήνουν κανέναν αδιάφορο, πόσο μάλλον αυτούς που ασχολούνται με μεγάλα συστήματα βάσεων δεδομένων. Η χρήση των συσκευών FPGA σε συνδυασμό με την ελάχιστη κατανάλωση ισχύος που έχουν, τα καθιστούν πολύ ελκυστικά για συν-επεξεργασία σε μεγάλα συστήματα βάσεων δεδομένων.



Εικόνα 5.19: Οι χρόνοι εκτέλεσης (latency) και τα ζητούμενα ποσοστά (*issue rate*) των επερωτήσεων Q1, Q2, Q3 και Q4 κατά την εκτέλεση τους στην προσομοίωση

Η συγκεκριμένη έρευνα ευελπιστεί να δώσει τροφή για σκέψη σε όσους ψάχνουν και ασχολούνται με τον μη παραδοσιακό τρόπο επεξεργασίας δεδομένων. Το κεφάλαιο αυτό, σκανδαλίζει το μυαλό για τέτοιου είδους σκέψεις.

Η υλοποίηση στο υλικό δυστυχώς δεν κατέστη δυνατόν και ως εκ τούτου δεν έχουμε χειροπιαστά αποτελέσματα. Αν και είναι κοινή πεποίθηση όλων ότι, όταν ένα κύκλωμα λειτουργεί σωστά και αποδοτικά στην προσομοίωση θα λειτουργεί το ίδιο και στης πραγματικές

συνθήκες, όχι πάντα αλλά τις περισσότερες φορές. Τα προβλήματα που συνήθως ελλοχεύουν στις συσκευές FPGA, κατόπιν σωστής προσομοίωσης, είναι φύσεως υλικό και όχι σχεδιαστικά.

5.5 Αναφορές

[01] Altera Corporation. «Cyclone Device Handbook». Altera, Copyright © 2008.

[02] Altera Corporation. «Quartus II Introduction for VHDL Users. Altera, Copyright © 2010.

[03] Altera Corporation. «Quartus II Handbook v11.0.0». Altera, Copyright © 2011.

[04] Altera Corporation. http://www.altera.com/

[05] Atmel Corporation. «AT40K Device Datasheets. Atmel», Copyright © 2002.

[06] Atmel Corporation. «Integrated Development System – Figaro Tutorial. Atmel», Copyright © 2002.

[07] Atmel Corporation. «Integrated Development System – Figaro User Guide. Atmel», Copyright © 2002.

[08] Atmel Corporation. http://www2.atmel.com/

[32] Xilinx Corporation. «ISE 10.1 Quick Start Tutorial». Xilinx, Copyright © 2008.

[33] Xilinx Corporation. «ISE Design Suite 10.1 Software Manuals». Xilinx, Copyright © 2008.

[37] Xilinx Corporation. http://www.xilinx.com/

[38] Ιωάννης Βενετικίδης, Μιχάλης Ψαράκης , Μεταπτυχιακή Διατριβή. «Συγκριτική Μελέτη Αρχιτεκτονικών και Εργαλείων Σχεδίασης Προγραμματιζόμενων Διατάξεων Λογικής». Πανεπιστήμιο Πειραιώς, Πειραιάς 2011.

Κεφάλαιο 6

Επίλογος

Σε αυτό το κεφάλαιο ο αναγνώστης θα έχει την ευκαιρία να δει κάποιες παρατηρήσεις και συμπεράσματα τα οποία βγήκαν από την προσπάθεια για υλοποίηση των επερωτήσεων από τη μορφή SQL στη μορφή VHDL. Τέλος θα γίνει και μια πρόταση για μελλοντική επέκταση της συγκεκριμένης έρευνας που υλοποιήθηκε.

Αναλυτικά στην Ενότητα 6.1 παρουσιάζονται οι παρατηρήσεις της ερευνητικής ομάδας στα παραγόμενα αποτελέσματα των προσομοιώσεων. Στην Ενότητα 6.2 παρουσιάζονται τα συμπεράσματα της ερευνητικής ομάδας στα παραγόμενα αποτελέσματα των προσομοιώσεων. Και τέλος στην Ενότητα 6.3 παρουσιάζονται οι προτάσεις της ερευνητικής ομάδας για μελλοντική επέκταση της έρευνας που υλοποιήθηκε.

6.1 Παρατηρήσεις

Αυτή η μεταπτυχιακή διατριβή προσπάθησε να ερευνήσει την τεχνολογία ενσωματωμένων υπολογιστικών συστημάτων και συγκεκριμένα να ασχοληθεί με την τεχνολογία των συσκευών FPGA. Έγινε προσπάθεια να υλοποιηθούν επερωτήσεις που είναι γραμμένες σε SQL (να γίνει

επέκταση τους ουσιαστικά), σε μορφή VHDL, αυτή η μορφή υποστηρίζεται από τις συσκευές FPGA. Συγκεκριμένα υλοποιήθηκα τέσσερις επερωτήσεις SQL στη μορφή VHDL και προσομοιώθηκαν ώστε να παραχθούν κάποιοι χρόνοι απόκρισης. Φάνηκε λοιπόν από τα αποτελέσματα ότι, αυτοί οι χρόνοι είναι αρκετά ικανοποιητική ώστε να μπορέσουν οι συσκευές FPGA να διαδραματίσουν μελλοντικό ρόλο στα πληροφορικά συστήματα ως συν-επεξεργαστικές μονάδες. Η συσκευές αυτές δίνουν λύσεις σε ένα ασταθές περιβάλλον με πολλές απαιτήσεις και τροποποιήσεις, προσαρμόζονται εύκολα και τροποποιούνται εύκολα.

Η έρευνα που υλοποιήθηκε στέφτηκε με επιτυχία καθώς εκμαιεύτηκαν χρήσιμα συμπεράσματα για την ταχύτητα επεξεργασίας αλλά και την ικανότητα παράλληλης επεξεργασίας.

Η προσπάθεια υλοποίησης των επερωτήσεων ήταν εύκολη μέχρι ένα σημείο αλλά όταν η σχεδίαση περιπλέκονταν τότε υπήρχαν αρκετά προβλήματα. Συγκεκριμένα το μεγαλύτερο πρόβλημα ήταν η διευθέτηση του συγχρονισμού των δεδομένων. Δηλαδή για να μπορέσει να παραχθεί το σωστό αποτέλεσμα έπρεπε τα δεδομένα των πράξεων αλλά και των καταχωρήσεων να φτάνουν τη σωστή χρονική στιγμή. Αυτή η διαδικασία ήταν αρκετά χρονοβόρα και επίπονη όσο το κύκλωμα μεγάλωνε και εμπλουτιζότανε. Παρόλα αυτά τα αποτελέσματα δικαιώνουν τη σχεδίαση ως αρκετά καλή. Έγινε υλοποίηση και εκτελέσει επερωτήσεων σε έναν έως το πολύ δυο κύκλων ρολογιού. Παράλληλα στα παραγόμενα κυκλώματα αν χρειαστεί μια μελλοντική τροποποίηση μεταβλητών και τελεστών συγκρίσεις, ώστε να αλλάζει η φύση της επερώτησης, να γίνεται πολύ εύκολα. Επιπλέον η σχεδίαση των κυκλωμάτων αυτών έγινε έτσι ώστε να επιτρέπεται η μελλοντική χρήση τους ως υπομονάδες άλλων μεγαλύτερων λογικών κυκλωμάτων.

Ένα δεύτερο θέμα που πρέπει να παρατηρηθεί είναι η προσπάθεια υλοποίησης των επερωτήσεων στο υλικό. Ο συγχρονισμός του υλικού με τον Η/Υ ήταν αρκετά δύσκολο και θέλει αρκετή εμπειρία στο να επιτευχθεί. Έγιναν προσπάθειες αλλά τα αποτελέσματα δεν δικαίωσαν το σχεδιαστή. Παράλληλα υπήρξαν θέματα σύνθεσης του παραγόμενου κώδικα στο εργαλείο σχεδίασης CAD ISE τα οποία όμως διευθετήθηκαν με τη συγγραφή νέου κώδικα. Η συγκεκριμένη παρέμβαση δεν έγινε στις λογικές μονάδες που εκτελούν τις επερωτήσεις αλλά σε κώδικα που διευθετεί την επικοινωνία της συσκευής με τον Η/Υ.

6.2 Συμπεράσματα

Κατόπιν των προσομοιώσεων και της ανάλυσης που υλοποιήθηκε σε αυτήν τη μεταπτυχιακή διατριβή για την επέκταση επερωτήσεων από τη μορφή SQL στη μορφή VHDL, παράχθηκαν τα παρακάτω συμπεράσματα για τις συσκευές FPGA και τη χρήση τους.

- Οι συσκευές FPGA εκτελούν αρκετά γρήγορα και με αξιοπιστία επερωτήσεις μεταφρασμένες από γλώσσα επερωτήσεων SQL σε VHDL.
- Τα παραγόμενα κυκλώματα μπορούν να χρησιμοποιηθούν ως μελλοντικές επεκτάσεις άλλων μεγαλύτερων λογικών κυκλωμάτων ή και μικρών αυτόνομων.
- Η κατασκευή τέτοιων κυκλωμάτων απαιτεί εμπειρία και καλή γνώση του υλικού που χρησιμοποιείται από το σχεδιαστή και όλη την ερευνητική ομάδα.
- Η χρήση τέτοιων συσκευών μπορεί να βοηθήσουν μεγαλύτερα συστήματα στην επεξεργασία δεδομένων.
- Αποκαλύφθηκε η ικανότητα των συσκευών στην παράλληλη επεξεργασία δεδομένων.
- Μεγάλη ευελιξία τροποποίησης των κυκλωμάτων και διαμόρφωσης τους.
- Ικανότητα εύκολης προσαρμογής και αντοχής.
- Υψηλές αποδόσεις με μικρή κατανάλωση ισχύος.
- Γρήγορος και αξιόπιστος έλεγχος πριν την τελική διαμόρφωση της συσκευής.
- Γρήγορη ικανότητα τροποποίησης των όποιων προβλημάτων.

6.3 Μελλοντικές Επεκτάσεις

Μετά από αυτή τη έρευνα σίγουρα γεννήθηκαν αρκετές σκέψεις αλλά και ιδέες για τη χρήση των συσκευών FPGA σε μελλοντικές εφαρμογές. Σίγουρα δεν φτάνουν κάποιες σελίδες μέσα σε μια μεταπτυχιακή διατριβή για να τις καταγράψουν. Η ομάδα που υλοποίησε τα παραγόμενα κυκλώματα θα αρκεστεί στην παρουσίαση των μελλοντικών επεκτάσεων αυτής της μεταπτυχιακής διατριβής και μόνο.

- Υλοποίηση στο υλικό, να εκτελεστεί σίγουρα η επέκταση των επερωτήσεων Q1, Q2, Q3 και Q4 στο υλικό. Να καταγράφουν τα αποτελέσματα και να συγκριθούν με αυτά των προσομοιώσεων.

- Να εκτελεστούν οι επερωτήσεις σε πολύ μεγάλο όγκο δεδομένων τόσο κατά την προσομοίωση όσο και κατά την υλοποίηση στο υλικό.

- Να καταγραφεί η πραγματική ταχύτητα των εκτελέσεων που γίνονται με μετρητές απευθείας στο υλικό.

- Να γίνει επέκταση των επερωτήσεων πολύπλοκης μορφής.

- Να γίνει χρήση επερωτήσεων με περισσότερες παράλληλες επεξεργασίες.

- Να γίνει χρήση μεγαλύτερων μνημών CAM και ROM- RAM.

- Να γίνει χρήση των ενσωματωμένων επεξεργαστών τόσο μαλακού πυρήνα (soft core) όσο και σκληρού (hard core), στην επεξεργασία επερωτήσεων.

- Να συγκριθούν οι επιδώσεις των συσκευών FPGA με αυτές των συμβατικών επεξεργαστικών μηχανών που χρησιμοποιούνται στην επεξεργασία δεδομένων.

- Να γίνει η καταμέτρηση ισχύος των συσκευών FPGA κατά την εκτέλεση των επερωτήσεων με αυτές των κοινών επεξεργαστικών μηχανών (CPU).

- Να τροποποιηθεί ο αλγόριθμος CAM_ARRAY ώστε να διαβάζει απευθείας από ένα αρχείο *.txt τον πατρικό πίνακα και να δημιουργεί απευθείας τα αρχεία αρχικοποίησης *.coe των μνημών CAM και RAM – ROM. Η τροποποίηση πρέπει να περιλαμβάνει την

εισαγωγή δυσδιάστατου πίνακα αλλά και τη βελτίωση που προτάθηκε στην Ενότητα 4.2.3.

- Να δημιουργηθεί μία «βιβλιοθήκη» η οποία θα περιέχει όλα τα λογικά κυκλώματα που θα δημιουργούνται, με τις τεκμηριώσεις τους, ώστε να χρησιμοποιούνται για μελλοντική χρήση από άλλα λογικά κυκλώματα.

- Χρήση άλλων θυρών επικοινωνίας της συσκευής FPGA όπως Ethernet και PCI για τη διενέργεια πειραμάτων.

Βιβλιογραφία

[01] Altera Corporation. «Cyclone Device Handbook». Altera, Copyright © 2008.

[02] Altera Corporation. «Quartus II Introduction for VHDL Users. Altera, Copyright © 2010.

[03] Altera Corporation. «Quartus II Handbook v11.0.0». Altera, Copyright © 2011.

[04] Altera Corporation. http://www.altera.com/

[05] Atmel Corporation. «AT40K Device Datasheets. Atmel», Copyright © 2002.

[06] Atmel Corporation. «Integrated Development System – Figaro Tutorial. Atmel», Copyright © 2002.

[07] Atmel Corporation. «Integrated Development System – Figaro User Guide. Atmel», Copyright © 2002.

[08] Atmel Corporation. http://www2.atmel.com/

[09] Clive "Max" Maxfield. «The Design Warrior's Guide to FPGAs». Mentor Graphics Corporation and Xilinx, Inc. Elsevier, ISBN 0-7506-7604-3, Copyright 2004.

[10] David A.Patterson, John L. Hennessy. «Οργάνωση και Σχεδίαση Υπολογιστών, Διασύνδεση Υλικού και Λογισμικού, 3 Αμερικάνικη Έκδοση». Εκδόσεις Κλειδάριθμος, ISBN 960-209-906-2, Αθήνα 2006.

[11] Hardi Electronics AB. «VHDL Handbook». Hardi Electronics AB, Copyright © 2000.

[12] IEEE. http://www.ieee.org/index.html

[13] Kostas Pagiamtzis, Student Member, IEEE, and Ali Sheikholeslami, Senior Member. «Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey».

IEEE, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 3, Toronto, Canada, MARCH 2006.

[14] Louis Woods, Jens Teubner, Gustavo Alonso. «Complex Event Detection at Wire Speed with FPGAs». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2010.

[15] Mentor Graphics. http://model.com/content/modelsim-downloads

[16] Morris Mano. «Ψηφιακή σχεδίαση, 3 Έκδοση». Εκδόσεις Παπασωτηρίου, ISBN 960-7530-63-2, Αθήνα 2003.

[17] N. Shirazi, D. Benyamin, W. Luk, P.Y.K. Cheung, S. Guo. «Quantitative Analysis of FPGA-based Database Searching». Received July 1999 - Revised December 1999, Journal of VLSI Signal Processing 28, 85–96, 2001. 2001 Kluwer Academic Publishers. Manufactured in The Netherlands, Copyright © 2001.

[18] Pong P. Chu. «FPGA Prototyping By VHDL Examples, Xilinx SpartanTM-3V version». A JOHN WILEY & SONS, INC, PUBLICATION, ISBN 978-0-470-18531-5, Hoboken, New Jersey, USA, 2008.

[19] R. Elmasri, S.B. Navathe. «Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων». Εκδόσεις Δίαυλος, ISBN 978-960-531-219-0, Αθήνα 2001.

[20] Rene Mueller, Jens Teubner, Gustavo Alonso. «Streams on Wires — A Query Compiler for FPGAs». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2009.

[21] Rene Mueller, Jens Teubner, Gustavo Alonso. «Data Processing on FPGAs». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2009.

[22] Rene Mueller, Jens Teubner. «FPGA: What's in it for a Database? ». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2009.

[23] Rene Mueller, Jens Teubner. «FPGAs: A New Point in the Database Design Space». Systems Group, Department of Computer Science, ETH Zurich, Switzerland, Copyright © 2010.

[24] Stefanvhdl. http://www.stefanvhdl.com/vhdl/html/file_write.html

[25] Stefanvhdl. http://www.stefanvhdl.com/vhdl/html/file_read.html

[26] Stephen Brown, Zvonko Vranesic. «Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL». Εκδόσεις Τζιόλα Θεσσαλονίκη, ISBN 960-8050-50-20, Θεσσαλονίκη 2001.

[27] Texas Instruments. http://www.ti.com/

[28] Wikipedia. «Hardware_description_language». http://en.wikipedia.org/wiki/Hardware_description_language.

[29] Xilinx Corporation. «Using Virtex-II Block RAM for High Performance Read/Write CAMs, XAPP260 (v1.1) ». Xilinx, Copyright © February 27, 2002.

[30] Xilinx Corporation. «Using Block RAM in Spartan-3 Generation FPGAs, XAPP463 (v2.0) ». Xilinx, Copyright © March 1, 2005.

[31] Xilinx Corporation. «Single – Port Block Memory Core v6.2, Logic Core, DS234». Xilinx, Copyright © April 28, 2005.

[32] Xilinx Corporation. «ISE 10.1 Quick Start Tutorial». Xilinx, Copyright © 2008.

[33] Xilinx Corporation. «ISE Design Suite 10.1 Software Manuals». Xilinx, Copyright © 2008.

[34] Xilinx Corporation. «Content-Addressable Memory v6.1, Logic Core, DS253». Xilinx, Copyright © September 19, 2008.

[35] Xilinx Corporation. «Spartan-3 FPGA Family Data Sheet». Xilinx, Copyright © 2009.

[36] Xilinx Corporation. «Spartan-3 Generation FPGA User Guide». Xilinx, Copyright © 2010.

[37] Xilinx Corporation. http://www.xilinx.com/

[38] Ιωάννης Βενετικίδης, Μιχάλης Ψαράκης , Μεταπτυχιακή Διατριβή. «Συγκριτική Μελέτη Αρχιτεκτονικών και Εργαλείων Σχεδίασης Προγραμματιζόμενων Διατάξεων Λογικής». Πανεπιστήμιο Πειραιώς, Πειραιάς 2011.

[39] Μιχάλης Ξένος, Δημήτρης Χρηστοδουλάκης. «Βάσεις Δεδομένων». Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα 2000.

[40] Μιχάλης Ψαράκης. «Σημειώσεις / Διαφάνειες Διδασκαλίας, Προηγμένης Ψηφιακής Σχεδίασης». Πανεπιστήμιο Πειραιώς. Τμήμα Πληροφορικής, Πρόγραμμα Μεταπτυχιακών Σπουδών "Προηγμένα συστήματα πληροφορικής", κατεύθυνση "Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων", Πειραιάς 2009.

[41] Μιχάλης Ψαράκης. «Εργαστηριακές Σημειώσεις: Σύντομος Οδηγός Εκμάθησης του Λογισμικού Xilinx ISE, Προηγμένης Ψηφιακής Σχεδίασης». Πανεπιστήμιο Πειραιώς, Τμήμα Πληροφορικής, Πρόγραμμα Μεταπτυχιακών Σπουδών "Προηγμένα συστήματα πληροφορικής", κατεύθυνση "Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων", Πειραιάς 2009.

Προγράμματα – Υλικό

Για την υλοποίηση της μεταπτυχιακής διατριβής χρησιμοποιήθηκαν τα παρακάτω προγράμματα:

- Microsoft Office Word 2003
- Microsoft Office Excel 2003
- Microsoft Office Visio 2003
- Microsoft PowerPoint 2003
- Microsoft Hyper Terminal
- ISE 10.1, Xilinx
- Eclipse

Οι προσομοιώσεις έγιναν σε ηλεκτρονικό υπολογιστή με λειτουργικό σύστημα Microsoft Windows XP Professional Version 2002 Service Pack 3 και φέρει επεξεργαστή Intel Core2 CPU 4400@2.00GHz 2.01GHz, 1,97 GB RAM.

Η προσπάθεια υλοποίησης στο υλικό του FPGA έγινε στη συσκευή της Xilinx, Spartan-3E 1600E.

Σύμβολο Γλώσσας	Όνομα Γλώσσας	Σημείωση

Παράρτημα Α

Πρόσθετα Στοιχεία

Α.1 Πίνακες - Εικόνες

ABEL	Advanced Boolean Expression Language	
AHDL	Altera HDL	a proprietary language from Altera
AHPL	A Hardware Programing language	
Bluespec		high-level HDL originally based on Haskell, now with a SystemVerilog
C-to-Verilog		Converter from C to Verilog
Confluence		a functional HDL; has been discontinued)
CoWareC		a C-based HDL by CoWare. Now discontinued in favor of SystemC
CUPL		a proprietary language from Logical Devices, Inc.
Handel-C		a C-like design language
HJJ	Hardware Join Java	based on Join Java
HML		based on SML
Hydra		based on Haskell
Impulse C	another C-like HDL	
ParC	Parallel C++	C++ extended with HDL style threading and communication for task-parallel programming
JHDL		based on Java
Lava		based on Haskell
M		A HDL from Mentor Graphics
MyHDL		based on Python
PALASM		for Programmable Array Logic (PAL) devices
ROCCC 2.0	Riverside Optimizing Compiler for Configurable Computing	Free and open-source C to HDL tool
RHDL		based on the Ruby programming language
Ruby [hardware description		
SystemC		a standardized class of C++ libraries for high-level behavioral and
SystemVerilog		a superset of Verilog, with enhancements to address system-level design
SystemTCL		SDL based on Tcl.
Verilog		most widely-used and well-supported HDL
VHDL	VHSIC HDL	most widely-used and well-supported HDL

Πίνακας Α.1: [28] Είδη γλωσσών HDL που έχουν αναπτυχθεί κατά καιρούς

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Εικόνα Α.1: ASCII κώδικας

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Εικόνα Α.2: ASCII κώδικας

A.2 Αλγόριθμος CAM_ARRAY

A.2.1 Κώδικας Αλγόριθμου

```
/*
 ============================================================================
 Name        : CAM_Array.c
 Author      : venetikidis ioannis
 Version     :
 Copyright   : Your copyright notice
 Description : Make CAM array in C, Ansi-style
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXARRAY 11 // το μέγεθος του πίνακα εισαγωγής

// δομή Structure η οποία περιέχει τρεις μεταβλητές τύπου int,
// οι μεταβλητές αυτές χρειάζονται για τον καθορισμό του στοιχείου αυτού
// που έχει τις περισσότερες εμφανίσεις καθώς και αν ο πίνακας είναι μηδενικός.
// η δήλωση της δομής χρειάζεται για τον καθορισμό, έλεγχο και δημιουργία των πινάκων (CAM)
typedef struct {
        int sum;              // το πλήθος εμφάνισης του στοιχείου
        int item;             // ποιο στοιχείο είναι αυτό
        int zero_array; // αν ο πίνακας είναι μηδενικός
}Structure;

//ο σχολιασμός των συναρτήσεων που καλούνται, γίνεται μέσα στη main αλλά και στη θέση δήλωσης
τους
void printArray(char* in, int array[], int n);
void mergesort(int array[], int low, int high);
int ***array_3D (int x, int y, int z);
Structure eyresi_arithmou (int array[], int maxSizeArray);
void antikatasasi (int array[], int in_number, int count, int maxSizeArray) ;
void insert_items_in_array3D_main(int ***array, int x,int y, int z, int item);

int main()
{
// αυτοί είναι οι πίνακες δοκιμής που χρησιμοποιούσα για να ελέγξω τον αλγόριθμο
//int *array[MAXARRAY]= { 9 , 1 , 1 , 1 , 1 , 1 , 3 , 4 , 11 , 4 , 5 , 2 , 1, 8, 9 , 10 } ;
//int *array[MAXARRAY]= { 9 , 1 , 1 , 1 , 1 , 2 , 3 , 4 , 4 , 4 , 5 , 2 , 7, 8, 9 , 1 } ;
//int *array[MAXARRAY]= { 1 , 2 , 2 , 2 , 3 , 7 , 4 , 5 , 5 , 5 , 5 , 6 , 7, 8, 9 , 10 } ;
//int *array[MAXARRAY]= { 15 , 3 , 9 , 4 , 13 , 6 , 7, 8, 2 , 10 , 11 , 12 , 5 , 14 , 1 ,16} ;
//int *array[MAXARRAY]= { 1 , 1 , 2 , 2 , 3 , 3 , 4 , 4 , 5 , 5 , 5 , 7 , 7, 7, 7 , 7 } ;
//int *array[MAXARRAY]= { 1 , 2 , 2 , 2 , 3 , 4 , 4 , 4 , 5 , 5 , 5 , 7 , 7, 7, 7 , 7 } ;
//int *array[MAXARRAY]= { 7 , 2 , 2 , 2 , 3 , 4 , 4 , 4 , 5 , 5 , 5 , 7 , 7, 7, 7 , 7 } ;
        int *array[MAXARRAY]= { 2 , 1 , 2 , 1 , 2 , 1 , 4 , 8 , 1 , 3 , 5 } ;
        //int *array[MAXARRAY];

        //  Array 3 Dimensions
              int x2, y2, z2;

        //  Array Iterators
        int i, j, k,num =0,num2=1;

        //μεταβλητές οι οποίες χρειάζονται για τον καθορισμό των πινάκων (μνημών CAM) καθώς και
        το μέγεθος αυτών
        int div_number,mod_number,number_of_cam, size_cam_array;

        //δήλωση τρισδιάστατου πίνακα 3D Array
        int ***array3D_main;

        //δήλωση μεταβλητής τύπου Structure, έχει στη δομή της τρεις μεταβλητές τύπου int
        //οι οποίες χρειάζονται για την αρχικοποίηση των πινάκων, ποιο στοιχείο, πόσες
        εμφανίσεις και αν ο πίνακας είναι μηδενικός
        Structure ret;


        //srand(time(NULL));
        // εισαγωγή τυχαίων αριθμών στον πίνακα array
```

```
	//for (i=0; i<MAXARRAY; i++)
	//{
		// επανάληψη (ουσιαστικά έλεγχος) η οποία αποτρέπει την εισαγωγή του 0 (μηδέν) αριθμού στον πίνακα array
		//do
		//{
			//array[i] = rand()% 1000;
		//}while (array[i] == 0);
	//}//end for

	//εμφάνιση περιεχομένων πίνακα εισόδου
	printArray("\nO πίνακας εισαγωγής V: {", array, MAXARRAY);

	//ταξινόμηση πίνακα εισόδου
	mergesort(array, 0, MAXARRAY - 1);

	//εμφάνιση περιεχομένων ταξινομημένου πίνακα
	printArray("\nO ταξινομημένος πίνακας εισαγωγής V: {", array, MAXARRAY);

	//καλεί τη συνάρτηση για να βρει αυτό το στοιχείο του πίνακα το οποίο έχει τις περισσότερες εμφανίσεις,
	//από τα νούμερα που θα ανακτηθούν θα μπορέσουμε να βγάλουμε το πλήθος των μνημών CAM που χρειαζόμαστε, ποιο είναι το στοιχείο αυτό
	//αλλά και αν ο πίνακας έχει (ολόκληρος) μηδενικά στοιχεία.
	ret = eyresi_arithmou (array, MAXARRAY);

	//βρίσκει το πλήθος των μνημών CAM που χρειάζονται
	number_of_cam = ret.sum;

	//κάνει διαίρεση το μέγεθος του πίνακα εισόδου με το μέγιστο πλήθος του αριθμού των στοιχείων εισόδου
	div_number = MAXARRAY / ret.sum;

	//βρίσκει το υπόλοιπο της διαίρεσης
	mod_number = MAXARRAY % ret.sum;

	//εδώ υπολογίσουμε το πλήθος των στοιχείων που χρειαζόμαστε σε κάθε μνήμη CAM (το μέγεθος κάθε CAM)
	if (mod_number !=0)//εάν το υπόλοιπο της διαίρεσης MAXARRAY % ret.sum είναι διάφορο του μηδέν κάνε τα παρακάτω
	{
		size_cam_array = div_number+1;
	}//end if
	else if(mod_number == 0) //εάν είναι μηδέν κάνε τα παρακάτω
	{
		size_cam_array = div_number;
	}//end else

	//printf("div_number: %d \n", div_number);
	//printf("mod_number: %d \n", mod_number);

	if (number_of_cam ==1)
	{
		printf("\nΧρειαζόμαστε μόνον %d πίνακα για να δημιουργήσουμε την μνήμη CAM "
			"και το βάθος αυτού είναι %d εγγραφές \n", number_of_cam,
size_cam_array);
	}//end if
	else
	{
		printf("\nO αριθμός των πινάκων που χρειαζόμαστε για την δημιουργία των μνημών CAM είναι %d  "
			"και το βάθος αυτών είναι %d εγγραφές για το καθένα \n",
number_of_cam, size_cam_array);
	}//end else

	x2 = number_of_cam;
	y2 = size_cam_array;
	z2 = 1;

	//καλεί την συνάρτηση για να δημιουργήσει τους πίνακες που χρειαζόμαστε (ουσιαστικά το πλήθος των CAM)
	//και παράλληλα τους αρχικοποιεί με τον αριθμό 0 (μηδέν)
	array3D_main= array_3D(number_of_cam,size_cam_array,z2);

	//αρχικοποιεί τους πίνακες που δημιουργήθηκαν με τον αριθμό ο οποίος έχει τις περισσότερες εμφανίσεις στον ταξινομημένο πίνακα εισόδου
	for(i = 0; i < number_of_cam; i++)
	{
		insert_items_in_array3D_main(array3D_main,i , 0, 0, ret.item);
	}//end for
```

```
//καλεί την συνάρτηση η οποία αντικαθιστά το στοιχείο εισαγωγής
//με τον αριθμό 0 (μηδέν)στον ταξινομημένο πίνακα εισαγωγής,
//η αντικατάσταση γίνεται με βάση το πλήθος των εμφανίσεων του στοιχείου αυτού.
antikatasasi(array, ret.item, ret.sum, MAXARRAY);

//εμφάνιση περιεχομένων πίνακα
//printArray("\nV: ", array, MAXARRAY);

//θα γίνει επανάληψη η οποία θα εισάγει όλους τους υπόλοιπους αριθμούς οι οποίοι έχουν
παραμείνει στον ταξινομημένο πίνακα εισαγωγής
//η επανάληψη θα σταματήσει όταν όλα τα στοιχεία του πίνακα μηδενιστούν
do
{
        //εύρεση επόμενου αριθμού ο οποίος έχει τις περισσότερες εμφανίσεις στον
        ταξινομημένο πίνακα εισαγωγής
        ret = eyresi_arithmou (array, MAXARRAY);

        //στα παρακάτω βήματα γίνεται η εισαγωγή των υπολοίπων στοιχείων του πίνακα
        εισαγωγής στους δημιουργηθέντες πίνακες (CAM)
        if (ret.zero_array == 1)//αν ο πίνακας έχει μηδενιστεί βγάλε το μήνυμα
        {
                printf("\nO πίνακας V πλέον είναι μηδενικός \n");

        }//end if
        else //αν ο πίνακας δεν έχει μηδενιστεί κάνε τα παρακάτω
        {
                if (num == number_of_cam ) //συνθήκη ελέγχου για μετρητές εισαγωγής
                                           δεδομένων
                {
                        num = 0;
                        num2++;
                }//end if

                if (num != number_of_cam ) //συνθήκη ελέγχου για μετρητές εισαγωγής
                                           δεδομένων
                {
                        for(i = 0; i < ret.sum; i++)
                        {
                                if (num == number_of_cam ) //συνθήκη ελέγχου για μετρητές
                                                           εισαγωγής δεδομένων
                                {
                                        num = 0;
                                        num2++;
                                        insert_items_in_array3D_main(array3D_main, num,
                                                num2, 0, ret.item);
                                                //καλείται η συνάρτηση εισαγωγής δεδομένων
                                        num++;
                                }//end if
                                else
                                {
                                        insert_items_in_array3D_main(array3D_main, num,
                                                num2, 0, ret.item);
                                         //καλείται η συνάρτηση εισαγωγής δεδομένων
                                        num++;
                                }//end else
                        }//end for
                }//end if

                //καλεί την συνάρτηση η οποία αντικαθιστά το στοιχείο εισαγωγής (και τις
                εμφανίσεις του)
                //με τον αριθμό 0 (μηδέν)στον ταξινομημένο πίνακα εισαγωγής
                antikatasasi(array, ret.item, ret.sum, MAXARRAY);

                //εμφάνιση περιεχομένων πίνακα
                //printArray("V: ", array, MAXARRAY);

        }//end else

} while (ret.zero_array != 1); //end while //αν ο πίνακας έχει μηδενιστεί τότε σταμάτα
                               την επανάληψη

printf("\nΧρειαζόμαστε λοιπόν τα παρακάτω: \n", i+1);
//  Access array elements
//Τύπωσε όλους του δημιουργηθέντες πίνακες (CAM) καθώς τα στοιχεία αυτών
for(i = 0; i < x2; i++)
{
        printf("\n\t %dος πίνακας : {", i+1);
        for(j = 0; j < y2; j++)
        {
```

```
                        //printf("\n");
                        for(k = 0; k < z2; k++)
                        {
                                printf(" %d,", array3D_main[i][j][k]);
                        }//end for
                }//end for
                printf("}\n");
        }//end for

        //ελευθέρωσε τα στοιχεία των δημιουργηθέντων πινάκων (CAM) από την μνήμη
        for(i = 0; i < x2; i++)
        {
                free(array3D_main[i]);
        }//end for

        //ελευθέρωσε τον τρισδιάστατο πίνακα από την μνήμη
        free (array3D_main);
        return 0;
}

// συνάρτηση εισαγωγής δεδομένων, δεν επιστρέφει τίποτα. Ως είσοδο έχει έναν τρισδιάστατο πίνακα,
//τρεις αριθμούς για τον καθορισμό της θέσης εισαγωγής στον πίνακα και το στοιχείο εισαγωγής
void insert_items_in_array3D_main(int ***array, int x,int y, int z, int item)
{
        if (array[x][y][z] == 0)
        {
                array[x][y][z] = item;
        }//end if
}

//συνάρτηση η οποία βρίσκει το στοιχείο του πίνακα το οποίο έχει τις περισσότερες εμφανίσεις,
//από τα νούμερα που θα προκύψουν θα μπορέσουμε να βγάλουμε το πλήθος των μνημών CAM που χρειαζόμαστε, ποιο είναι το στοιχείο αυτό
//αλλά και αν ο πίνακας έχει (ολόκληρος) μηδενικά στοιχεία.
//επιστρέφει μια δομή (Structure) η οποία περιέχει τις παραπάνω τιμές
Structure eyresi_arithmou (int array[], int maxSizeArray)
{
        // αρχικοποίηση μεταβλητών ελέγχου
        int count1=1, count2=1, count3=0;

        // αρχικοποίηση μεταβλητών εξόδου
        int z=0,i=0,out=0,zero_array=0;

        //δήλωση μεταβλητής τύπου Structure, έχει στη δομή της τρεις μεταβλητές τύπου int
        //οι οποίες χρειάζονται για την αρχικοποίηση των πινάκων, ποιο στοιχείο, πόσες εμφανίσεις και αν ο πίνακας είναι μηδενικός
        Structure A;

        // αρχικοποίησε την μεταβλητή με το πρώτο στοιχείο του πίνακα
        z=array[0];

        for(i=0; i < maxSizeArray; i++)
        {
                if (z != '\0') // αν το στοιχείο είναι διάφορο του μηδενός κάνε τα παρακάτω
                {
                        if (z == array[i+1])
                        {
                                count1++;
                                //printf("%d %d\n", z, count1);
                        }//end if
                        else if (z<array[i+1])
                        {
                                if (count1 >= count2)
                                {
                                        count2 = count1;
                                        out = z;
                                }//end if
                                z=array[i+1];
                                count1=1;
                        }//end else if
                }//end if
                else // εδώ μετράει τα μηδενικά στοιχεία
                {
                        //printf("%d count3 \n", count3);
                        z=array[i+1];
                        count3++;
                        //printf("%d \n", z);
                }//end else
```

```
                if (count3 == maxSizeArray) // όταν ο πίνακας είναι μηδενικός τότε επιστρέφει 1
                                            στο zero_array
                {
                        zero_array = 1;
                        count2 = 0;
                        out = 0;
                }//end if
                else // αν δεν είναι μηδενικός τότε επιστρέφει 0 στο zero_array
                {
                        zero_array = 0;
                }//end else
        }//end for

        // θέσε τα στοιχεία που βρήκες στην δομή A για να εξέλθουν οι πληροφορίες
        A.sum = count2;
        A.item = out;
        A.zero_array = zero_array;

        return A; //επέστρεψε τη δομή A
}

//συνάρτηση η οποία αντικαθιστά το στοιχείο εισαγωγής
//με τον αριθμό 0 (μηδέν)στον ταξινομημένο πίνακα εισαγωγής,
//η αντικατάσταση γίνεται με βάση το πλήθος των εμφανίσεων του στοιχείου αυτού.
void antikatasasi (int array[], int in_number, int count, int maxSizeArray)
{
        int i, j;

        for(i = 0; i < maxSizeArray; i++)
        {
                if (in_number == array[i]) // αν το στοιχείο του πίνακα είναι ίδιο με το
                                           δοθέντο τότε κάνε τα παρακάτω (μηδένισε τα)
                {
                        for(j = 0; j < count; j++) // όσο είναι το πλήθος των στοιχείων αυτών
                                                   μηδένισε τα
                        {
                                array[i] = '\0';
                                if (i < 15)
                                {
                                        i++;
                                }//end if
                        }//end for
                }//end if
        }//end for
}

//συνάρτηση η οποία δημιουργεί τους πίνακες που χρειαζόμαστε (ουσιαστικά το πλήθος των CAM)
//και παράλληλα τους αρχικοποιεί με τον αριθμό 0 (μηδέν).
//Η δημιουργία γίνεται με δυναμικό τρόπο, επιστρέφει έναν τρισδιάστατο πίνακα.
int ***array_3D(int x, int y, int z)
{
        //  Array 3 Dimensions
        int *allElements = malloc(x * y * z * sizeof(int));
        int ***array3D = malloc(x * sizeof(int **));

        //  Array Iterators
        int i, j, k;

        for(i = 0; i < x; i++)
        {
                array3D[i] = malloc(y * sizeof(int *));
                for(j = 0; j < y; j++)
                {
                        array3D[i][j] = allElements + (i * y * z) + (j * z);
                }//end for
        }//end for

        //  Access array elements
        for(i = 0; i < x; i++)
        {
                for(j = 0; j < y; j++)
                {
                        for(k = 0; k < z; k++)
                        {
                                array3D[i][j][k] = '\0';
                        }//end for
                }//end for
        }//end for
        return array3D;
}
```

```
//συνάρτηση που τυπώνει τα περιεχόμενα ενός πίνακα
void printArray(char* in, int array[], int n)
{
	printf("%s", in);
	int i = 0;
	for (; i < n; ++i)
	{
		printf(" %d,", array[i]);
	}//end for
	printf("}\n");
}

//συνάρτηση η οποία ταξινομεί τα περιεχόμενα ενός πίνακα, χρησιμοποιείται ο αλγόριθμος mergesort
void mergesort(int a[], int low, int high)
{
	int i = 0;
	int length = high - low + 1;
	int pivot  = 0;
	int merge1 = 0;
	int merge2 = 0;

	//int working[length];
	int *working = (int*)malloc((length) * sizeof(int));

	if(low == high)
		return;

	pivot  = (low + high) / 2;
	mergesort(a, low, pivot);
	mergesort(a, pivot + 1, high);

	for(i = 0; i < length; i++)
	{
		working[i] = a[low + i];
	}//end for

	merge1 = 0;
	merge2 = pivot - low + 1;

	for(i = 0; i < length; i++)
	{
		if(merge2 <= high - low)
			if(merge1 <= pivot - low)
				if(working[merge1] > working[merge2])
					a[i + low] = working[merge2++];
				else
					a[i + low] = working[merge1++];
			else
				a[i + low] = working[merge2++];
		else
			a[i + low] = working[merge1++];
	}//end for
	//memcpy(a + low, working, (length) * sizeof(int));
	free(working);
}
```

A.2.2 Εκτέλεση Αλγόριθμου

Ο πίνακας εισαγωγής V: { 98, 809, 660, 938, 383, 325, 693, 602, 515, 53, 565, 981, 716, 1, 238, 815, 818, 455, 235, 309, 92, 10, 46, 785, 69, 973, 776, 759, 537, 233, 205, 782, 143, 604, 464, 199, 576, 441, 492, 954, 966, 635, 674, 805, 639, 986, 445, 274, 724, 425, 871, 459, 403, 342, 553, 73, 396, 364, 328, 967, 904, 842, 711, 650, 535, 404, 41, 366, 228, 729, 788, 485, 15, 896, 671, 408, 328, 836, 428, 833, 484, 758, 685, 968, 639, 573, 701, 743, 669, 306, 614, 587, 353, 254, 599, 427, 404, 451, 222, 40, 559, 267, 973, 753, 332, 771, 848, 729, 321, 744, 510, 110, 621, 113, 105, 220, 687, 436, 819, 777, 931, 401, 798, 131, 589, 595,

331, 627, 702, 422, 736, 289, 339, 67, 780, 145, 651, 36, 657, 253, 138, 668, 191, 741, 975, 495, 671, 39, 619, 741, 319, 388, 724, 168, 696, 786, 913, 385, 651, 233, 480, 144, 545, 549, 218, 942, 621, 592, 332, 78, 48, 107, 335, 432, 559, 231, 439, 416, 547, 3, 865, 352, 34, 684, 184, 726, 791, 370, 532, 252, 348, 438, 757, 961, 620, 690, 673, 709, 650, 590, 492, 764, 40, 527, 301, 580, 530, 803, 849, 723, 752, 814, 885, 34, 429, 931, 282, 811, 963, 353, 718, 723, 732, 593, 137, 795, 195, 289, 247, 681, 329, 96, 671, 408, 947, 635, 184, 221, 676, 552, 282, 806, 993, 196, 842, 312, 511, 311, 720, 824, 831, 979, 654, 20, 719, 599, 420, 770, 559, 343, 338, 791, 889, 375, 202, 614, 995, 892, 506, 628, 443, 168, 663, 617, 740, 331, 509, 343, 582, 333, 437, 974, 55, 932, 926, 393, 743, 991, 969, 151, 641, 944, 480, 961, 944, 216, 740, 905, 454, 644, 872, 470, 279, 705, 995, 127, 210, 31, 940, 904, 552, 866, 578, 876, 912, 985, 51, 812, 545, 11, 486, 250, 926, 27, 97, 396, 581, 473, 832, 421, 419, 761, 177, 599, 318, 719, 486, 866, 266, 108, 916, 834, 934, 770, 627, 788, 372, 659, 844, 773, 505, 749, 949, 499, 567, 352, 784, 9, 344, 581, 194, 421, 923, 666, 243, 695, 437, 432, 978, 27, 571, 595, 806, 565, 606, 810, 277, 543, 464, 384, 195, 819, 244, 906, 840, 366, 166, 221, 359, 258, 619, 711, 60, 159, 670, 304, 584, 684, 570, 234, 167, 38, 480, 285, 920, 534, 253, 176, 834, 648, 322, 55, 394, 957, 397, 197, 115, 667, 370, 904, 231, 462, 649, 234, 413, 427, 712, 928, 386, 335, 224, 425, 311, 655, 198, 139, 175, 755, 881, 797, 886, 214, 794, 286, 931, 966, 349, 415, 668, 47, 219, 326, 852, 179, 907, 148, 415, 284, 143, 115, 798, 414, 566, 319, 29, 565, 175, 526, 440, 674, 881, 537, 634, 140, 66, 498, 929, 855, 541, 699, 550, 924, 416, 970, 293, 957, 115, 839, 899, 928, 378, 168, 279, 223, 969, 558, 776, 641, 592, 146, 292, 774, 643, 963, 19, 917, 836, 707, 695, 357, 591, 330, 820, 826, 348, 873, 272, 982, 168, 510, 99, 851, 815, 398, 462, 715, 746, 531, 109, 820, 255, 966, 880, 830, 35, 54, 479, 900, 319, 737, 554, 362, 92, 445, 731, 9, 463, 345, 352, 26, 815, 91, 228, 874, 600, 393, 544, 823, 91, 726, 227, 702, 996, 327, 289, 722, 661, 453, 661, 610, 893, 775, 596, 935, 729, 234, 32, 182, 248, 168, 630, 892, 885, 274, 950, 477, 465, 143, 804, 662, 421, 920, 874, 886, 777, 426, 763, 627, 169, 949, 336, 241, 51, 90, 220, 92, 29, 588, 72, 856, 680, 322, 5, 64, 551, 392, 627, 22, 407, 925, 69, 601, 992, 678, 444, 707, 387, 768, 104, 395, 972, 423, 569, 150, 316, 190, 306, 136, 16, 500, 854, 752, 495, 707, 373, 179, 311, 748, 489, 302, 487, 300, 96, 909, 675, 421, 877, 702, 734, 995, 980, 326, 8, 867, 151, 397, 642, 517, 562, 188, 166, 7, 216, 723, 641, 683, 775, 906, 262, 793, 766, 760, 617, 899, 418, 434, 20, 817, 423, 795, 759, 398, 459, 292, 881, 294, 204, 367, 992, 956, 539, 712, 987, 330, 268, 622, 994, 608, 807, 4, 578, 59, 496, 496, 567, 864, 860, 80, 907, 65, 238, 278, 787, 290, 914, 985, 31, 465, 671, 256, 48, 588, 411, 158, 706, 219, 640, 343, 884, 550, 818, 244, 854, 829, 631, 917, 957, 463, 676, 828, 458, 40, 787, 310, 675, 949, 146, 193, 924, 33, 481, 903, 300, 897, 102, 467, 481, 173, 31, 470, 937, 241, 430, 301, 765, 551, 176, 934, 789, 269, 431, 56, 451, 287, 101, 457, 121, 164, 359, 169, 147, 495, 503, 671, 289, 689, 863, 144, 497, 700, 320, 814, 794, 171, 454, 912, 428, 250, 752, 450, 765, 57, 60, 438, 146, 497, 987, 253, 948, 784, 181, 547, 503, 703, 461, 459, 375, 564, 74, 879, 796, 33, 9, 941,

311, 417, 954, 765, 541, 426, 403, 466, 150, 536, 78, 931, 619, 409, 839, 401, 709, 790, 599, 581, 854, 507, 602, 799, 885, 707, 754, 48, 704, 810, 673, 125, 159, 696, 564, 378, 341, 522, 16, 49, 154, 540, 660, 239, 791, 456, 940, 455, 804, 103, 496, 406, 418, 117, 732, 187, 899, 735, 219, 986, 131, 667, 85, 967, 494, 133, 195, 770, 27, 440, 180, 926, 369, 832, 167, 792, 914, 444, 761, 928, 261, 519, 161, 525, 813, 120, 970, 645, 713, 169, 44, 715, 473, 739, 986, 701, 124, 889, 49, 108, 94, 502, 568, 161, 544, 226, 557, 157, 2, 671, 102, 149, 51, 458, 440, 37, 762, 708, 282, 462, 845, 895, 406, 319, 113, 284, 834, 890, 817, 11, 200, 164, 580, 814, 89, 626, 102, 77, 444, 22, 275, 39, 253, 399, 421, 393, 63, 22, 765, 575, 280, 413, 509, 802, 19, 573, 297, 92, 594, 145, 950, 319, 892, 104, 829, 643, 564, 778, 210, 688, 756, 23, 513, 973, 844, 501, 30, 145, 216, 704, 153, 477, 347, 443, 480, 361, 508, 771, 128, 634, 80, 257, 174, 347, 148, 191, 299, 899, 749, 746, 235, 120, 280, 147, 110, 379, 351, 409, 820, 469, 509, 525, 493, 991, 293, 844, 848, 617, 198, 570, 91, 998, 44, 564, 338, 609, 722, 903, 490, 584, 627, 650, 787, 910, 81, 541, 501, 432, 251, 79, 379, 794, 707, 131, 908, 517, 855, 174, 975, 148, 448, 611, 549, 763, 6, 883, 873, 352, 580, 186, 153, 234, 993, 901, 321, 46, 21, 127, 624, 161, 133, 900, 818, 896, 828, 210, 290, 571, 718, 910, 748, 331, 745, 87, 960, 362, 493, 951, 29, 764, 100, 772, 484, 204, 661, 378, 834, 341, 366, 603, 477, 803, 739, 465, 570, 672, 880, 398, 8, 576, 454, 839, 849, 178, 712, 548, 193, 818, 88, 16, 680, 293, 882, 466, 784, 379, 927, 726, 523, 68, 792, 38, 129, 495, 962, 877, 359, 16, 259, 65, 508, 527, 58, 990, 978, 421, 608, 88, 11, 76, 454, 245, 572, 268, 356, 144, 484, 169, 278, 847, 355, 669, 663, 832, 125, 72, 719, 989, 981, 940, 408, 676, 274, 918, 14, 810, 122, 932, 205, 767, 429, 205, 785, 407, 857, 16, 710, 358, 314, 797, 829,}

Ο ταξινομημένος πίνακας εισαγωγής V: { 1, 2, 3, 4, 5, 6, 7, 8, 8, 9, 9, 9, 10, 11, 11, 11, 14, 15, 16, 16, 16, 16, 16, 19, 19, 20, 20, 21, 22, 22, 22, 23, 26, 27, 27, 27, 29, 29, 29, 30, 31, 31, 31, 32, 33, 33, 34, 34, 35, 36, 37, 38, 38, 39, 39, 40, 40, 40, 41, 44, 44, 46, 46, 47, 48, 48, 48, 49, 49, 51, 51, 51, 53, 54, 55, 55, 56, 57, 58, 59, 60, 60, 63, 64, 65, 65, 66, 67, 68, 69, 69, 72, 72, 73, 74, 76, 77, 78, 78, 79, 80, 80, 81, 85, 87, 88, 88, 89, 90, 91, 91, 91, 92, 92, 92, 92, 94, 96, 96, 97, 98, 99, 100, 101, 102, 102, 102, 103, 104, 104, 105, 107, 108, 108, 109, 110, 110, 113, 113, 115, 115, 115, 117, 120, 120, 121, 122, 124, 125, 125, 127, 127, 128, 129, 131, 131, 131, 133, 133, 136, 137, 138, 139, 140, 143, 143, 143, 144, 144, 144, 145, 145, 145, 146, 146, 146, 147, 147, 148, 148, 148, 149, 150, 150, 151, 151, 153, 153, 154, 157, 158, 159, 159, 161, 161, 161, 164, 164, 166, 166, 167, 167, 168, 168, 168, 168, 168, 169, 169, 169, 169, 171, 173, 174, 174, 175, 175, 176, 176, 177, 178, 179, 179, 180, 181, 182, 184, 184, 186, 187, 188, 190, 191, 191, 193, 193, 194, 195, 195, 195, 196, 197, 198, 198, 199, 200, 202, 204, 204, 205, 205, 205, 210, 210, 210, 214, 216, 216, 216, 218, 219, 219, 219, 220, 220, 221, 221, 222, 223, 224, 226, 227, 228, 228, 231, 231, 233, 233, 234, 234, 234, 234, 235, 235, 238, 238, 239, 241, 241, 243, 244, 244, 245, 247, 248, 250, 250, 251, 252, 253, 253, 253, 253, 254, 255, 256, 257, 258,

259, 261, 262, 266, 267, 268, 268, 269, 272, 274, 274, 274, 275, 277, 278, 278,
279, 279, 280, 280, 282, 282, 282, 284, 284, 285, 286, 287, 289, 289, 289, 289,
290, 290, 292, 292, 293, 293, 293, 294, 297, 299, 300, 300, 301, 301, 302, 304,
306, 306, 309, 310, 311, 311, 311, 311, 312, 314, 316, 318, 319, 319, 319, 319,
319, 320, 321, 321, 322, 322, 325, 326, 326, 327, 328, 328, 329, 330, 330, 331,
331, 331, 332, 332, 333, 335, 335, 336, 338, 338, 339, 341, 341, 342, 343, 343,
343, 344, 345, 347, 347, 348, 348, 349, 351, 352, 352, 352, 352, 353, 353, 355,
356, 357, 358, 359, 359, 359, 361, 362, 362, 364, 366, 366, 366, 367, 369, 370,
370, 372, 373, 375, 375, 378, 378, 378, 379, 379, 379, 383, 384, 385, 386, 387,
388, 392, 393, 393, 393, 394, 395, 396, 396, 397, 397, 398, 398, 398, 399, 401,
401, 403, 403, 404, 404, 406, 406, 407, 407, 408, 408, 408, 409, 409, 411, 413,
413, 414, 415, 415, 416, 416, 417, 418, 418, 419, 420, 421, 421, 421, 421, 421,
421, 422, 423, 423, 425, 425, 426, 426, 427, 427, 428, 428, 429, 429, 430, 431,
432, 432, 432, 434, 436, 437, 437, 438, 438, 439, 440, 440, 440, 441, 443, 443,
444, 444, 444, 445, 445, 448, 450, 451, 451, 453, 454, 454, 454, 454, 455, 455,
456, 457, 458, 458, 459, 459, 459, 461, 462, 462, 462, 463, 463, 464, 464, 465,
465, 465, 466, 466, 467, 469, 470, 470, 473, 473, 477, 477, 477, 479, 480, 480,
480, 480, 481, 481, 484, 484, 484, 485, 486, 486, 487, 489, 490, 492, 492, 493,
493, 494, 495, 495, 495, 495, 496, 496, 496, 497, 497, 498, 499, 500, 501, 501,
502, 503, 503, 505, 506, 507, 508, 508, 509, 509, 509, 510, 510, 511, 513, 515,
517, 517, 519, 522, 523, 525, 525, 526, 527, 527, 530, 531, 532, 534, 535, 536,
537, 537, 539, 540, 541, 541, 541, 543, 544, 544, 545, 545, 547, 547, 548, 549,
549, 550, 550, 551, 551, 552, 552, 553, 554, 557, 558, 559, 559, 559, 562, 564,
564, 564, 564, 565, 565, 565, 566, 567, 567, 568, 569, 570, 570, 570, 571, 571,
572, 573, 573, 575, 576, 576, 578, 578, 580, 580, 580, 581, 581, 581, 582, 584,
584, 587, 588, 588, 589, 590, 591, 592, 592, 593, 594, 595, 595, 596, 599, 599,
599, 599, 600, 601, 602, 602, 603, 604, 606, 608, 608, 609, 610, 611, 614, 614,
617, 617, 617, 619, 619, 619, 620, 621, 621, 622, 624, 626, 627, 627, 627, 627,
627, 628, 630, 631, 634, 634, 635, 635, 639, 639, 640, 641, 641, 641, 642, 643,
643, 644, 645, 648, 649, 650, 650, 650, 651, 651, 654, 655, 657, 659, 660, 660,
661, 661, 661, 662, 663, 663, 666, 667, 667, 668, 668, 669, 669, 670, 671, 671,
671, 671, 671, 671, 672, 673, 673, 674, 674, 675, 675, 676, 676, 676, 678, 680,
680, 681, 683, 684, 684, 685, 687, 688, 689, 690, 693, 695, 695, 696, 696, 699,
700, 701, 701, 702, 702, 702, 703, 704, 704, 705, 706, 707, 707, 707, 707, 707,
708, 709, 709, 710, 711, 711, 712, 712, 712, 713, 715, 715, 716, 718, 718, 719,
719, 719, 720, 722, 722, 723, 723, 723, 724, 724, 726, 726, 726, 729, 729, 729,
731, 732, 732, 734, 735, 736, 737, 739, 739, 740, 740, 741, 741, 743, 743, 744,
745, 746, 746, 748, 748, 749, 749, 752, 752, 752, 753, 754, 755, 756, 757, 758,
759, 759, 760, 761, 761, 762, 763, 763, 764, 764, 765, 765, 765, 765, 766, 767,
768, 770, 770, 770, 771, 771, 772, 773, 774, 775, 775, 776, 776, 777, 777, 778,
780, 782, 784, 784, 784, 785, 785, 786, 787, 787, 787, 788, 788, 789, 790, 791,
791, 791, 792, 792, 793, 794, 794, 794, 795, 795, 796, 797, 797, 798, 798, 799,
802, 803, 803, 804, 804, 805, 806, 806, 807, 809, 810, 810, 810, 811, 812, 813,

814, 814, 814, 815, 815, 815, 817, 817, 818, 818, 818, 818, 819, 819, 820, 820, 820, 823, 824, 826, 828, 828, 829, 829, 829, 830, 831, 832, 832, 832, 833, 834, 834, 834, 834, 836, 836, 839, 839, 839, 840, 842, 842, 844, 844, 844, 845, 847, 848, 848, 849, 849, 851, 852, 854, 854, 854, 855, 855, 856, 857, 860, 863, 864, 865, 866, 866, 867, 871, 872, 873, 873, 874, 874, 876, 877, 877, 879, 880, 880, 881, 881, 881, 882, 883, 884, 885, 885, 885, 886, 886, 889, 889, 890, 892, 892, 892, 893, 895, 896, 896, 897, 899, 899, 899, 899, 900, 900, 901, 903, 903, 904, 904, 904, 905, 906, 906, 907, 907, 908, 909, 910, 910, 912, 912, 913, 914, 914, 916, 917, 917, 918, 920, 920, 923, 924, 924, 925, 926, 926, 926, 927, 928, 928, 928, 929, 931, 931, 931, 931, 932, 932, 934, 934, 935, 937, 938, 940, 940, 940, 941, 942, 944, 944, 947, 948, 949, 949, 949, 950, 950, 951, 954, 954, 956, 957, 957, 957, 960, 961, 961, 962, 963, 963, 966, 966, 966, 967, 967, 968, 969, 969, 970, 970, 972, 973, 973, 973, 974, 975, 975, 978, 978, 979, 980, 981, 981, 982, 985, 985, 986, 986, 986, 987, 987, 989, 990, 991, 991, 992, 992, 993, 993, 994, 995, 995, 995, 996, 998,}

Ο αριθμός των πινάκων που χρειαζόμαστε για την δημιουργία των μνημών CAM είναι 6 και το βάθος αυτών είναι 205 εγγραφές για το καθένα

Ο πίνακας V πλέον είναι μηδενικός

Χρειαζόμαστε λοιπόν τα παρακάτω:

1ος πίνακας : { 671, 421, 707, 627, 319, 168, 16, 899, 834, 765, 599, 495, 480, 352, 311, 253, 234, 92, 986, 966, 949, 928, 904, 885, 854, 839, 829, 815, 810, 791, 784, 752, 726, 719, 702, 661, 641, 617, 580, 565, 541, 496, 477, 462, 444, 432, 398, 379, 366, 343, 293, 274, 216, 205, 161, 146, 144, 131, 102, 51, 40, 29, 22, 9, 992, 985, 975, 967, 954, 934, 920, 912, 906, 896, 880, 873, 849, 836, 817, 803, 795, 785, 775, 763, 749, 743, 739, 722, 711, 701, 684, 674, 668, 660, 639, 621, 602, 588, 576, 567, 550, 545, 527, 510, 501, 492, 473, 464, 455, 443, 429, 426, 418, 413, 406, 401, 375, 353, 341, 332, 326, 306, 292, 280, 268, 241, 233, 221, 198, 184, 175, 166, 153, 147, 125, 110, 96, 78, 65, 49, 39, 33, 8, 982, 962, 942, 927, 909, 893, 876, 863, 847, 826, 809, 793, 778, 766, 755, 736, 713, 700, 687, 670, 654, 640, 622, 604, 593, 575, 558, 540, 531, 515, 502, 489, 461, 441, 422, 399, 386, 369, 356, 342, 325, 310, 294, 272, 259, 252, 239, 218, 196, 182, 171, 139, 124, 105, 97, 81, 68, 58, 41, 26, 7, 1,}

2ος πίνακας : { 671, 421, 707, 627, 319, 168, 931, 899, 818, 765, 564, 495, 454, 352, 289, 253, 169, 92, 986, 966, 949, 928, 904, 885, 854, 839, 829, 815, 810, 791, 784, 752, 726, 719, 702, 661, 641, 617, 580, 565, 541, 496, 477, 462, 444, 432, 398, 379, 366, 343, 293, 274, 216, 205, 161, 146, 144, 131, 102, 51, 40, 29, 22, 9, 991, 981, 970, 963, 950, 932, 917, 910, 903, 889, 877, 866,

848, 828, 806, 798, 792, 777, 771, 761, 748, 741, 732, 718, 709, 696, 680, 673, 667, 651, 635, 614, 595, 584, 573, 552, 549, 544, 525, 508, 497, 486, 470, 463, 451, 438, 428, 425, 416, 409, 404, 397, 370, 348, 338, 330, 322, 301, 290, 279, 250, 238, 231, 220, 193, 179, 174, 164, 151, 133, 120, 108, 88, 72, 60, 46, 38, 20, 998, 980, 960, 941, 925, 908, 890, 872, 860, 845, 824, 807, 790, 774, 762, 754, 735, 710, 699, 685, 666, 649, 631, 620, 603, 591, 572, 557, 539, 530, 513, 500, 487, 457, 439, 420, 395, 385, 367, 355, 339, 320, 309, 287, 269, 258, 251, 227, 214, 194, 181, 158, 138, 122, 103, 94, 79, 67, 57, 37, 23, 6, 0,}

3ος πίνακας : { 671, 421, 707, 627, 319, 16, 931, 899, 818, 765, 564, 495, 454, 352, 289, 253, 169, 92, 986, 966, 949, 928, 904, 885, 854, 839, 829, 815, 810, 791, 784, 752, 726, 719, 702, 661, 641, 617, 580, 565, 541, 496, 477, 462, 444, 432, 398, 379, 366, 343, 293, 274, 216, 205, 161, 146, 144, 131, 102, 51, 40, 29, 22, 9, 991, 981, 970, 963, 950, 932, 917, 910, 903, 889, 877, 866, 848, 828, 806, 798, 792, 777, 771, 761, 748, 741, 732, 718, 709, 696, 680, 673, 667, 651, 635, 614, 595, 584, 573, 552, 549, 544, 525, 508, 497, 486, 470, 463, 451, 438, 428, 425, 416, 409, 404, 397, 370, 348, 338, 330, 322, 301, 290, 279, 250, 238, 231, 220, 193, 179, 174, 164, 151, 133, 120, 108, 88, 72, 60, 46, 38, 20, 996, 979, 956, 938, 923, 905, 884, 871, 857, 840, 823, 805, 789, 773, 760, 753, 734, 708, 693, 683, 662, 648, 630, 611, 601, 590, 569, 554, 536, 526, 511, 499, 485, 456, 436, 419, 394, 384, 364, 351, 336, 318, 304, 286, 267, 257, 248, 226, 202, 190, 180, 157, 137, 121, 101, 90, 77, 66, 56, 36, 21, 5, 0,}

4ος πίνακας : { 671, 421, 707, 627, 168, 16, 931, 834, 818, 599, 564, 480, 454, 311, 289, 234, 169, 995, 973, 957, 940, 926, 892, 881, 844, 832, 820, 814, 794, 787, 770, 729, 723, 712, 676, 650, 619, 581, 570, 559, 509, 484, 465, 459, 440, 408, 393, 378, 359, 331, 282, 219, 210, 195, 148, 145, 143, 115, 91, 48, 31, 27, 11, 993, 987, 978, 969, 961, 944, 924, 914, 907, 900, 886, 874, 855, 842, 819, 804, 797, 788, 776, 764, 759, 746, 740, 724, 715, 704, 695, 675, 669, 663, 643, 634, 608, 592, 578, 571, 551, 547, 537, 517, 503, 493, 481, 466, 458, 445, 437, 427, 423, 415, 407, 403, 396, 362, 347, 335, 328, 321, 300, 284, 278, 244, 235, 228, 204, 191, 176, 167, 159, 150, 127, 113, 104, 80, 69, 55, 44, 34, 19, 994, 974, 951, 937, 918, 901, 883, 867, 856, 833, 813, 802, 786, 772, 758, 745, 731, 706, 690, 681, 659, 645, 628, 610, 600, 589, 568, 553, 535, 523, 507, 498, 479, 453, 434, 417, 392, 383, 361, 349, 333, 316, 302, 285, 266, 256, 247, 224, 200, 188, 178, 154, 136, 117, 100, 89, 76, 64, 54, 35, 15, 4, 0,}

5ος πίνακας : { 671, 421, 707, 319, 168, 16, 931, 834, 818, 599, 564, 480, 454, 311, 289, 234, 169, 995, 973, 957, 940, 926, 892, 881, 844, 832, 820, 814, 794, 787, 770, 729, 723, 712, 676, 650, 619, 581, 570, 559, 509, 484, 465, 459, 440, 408, 393, 378, 359, 331, 282, 219, 210, 195, 148, 145, 143, 115, 91, 48, 31, 27, 11, 993, 987, 978, 969, 961, 944, 924, 914, 907, 900, 886, 874, 855, 842, 819, 804, 797, 788, 776, 764, 759, 746, 740, 724, 715, 704, 695, 675, 669, 663, 643,

634, 608, 592, 578, 571, 551, 547, 537, 517, 503, 493, 481, 466, 458, 445, 437, 427, 423, 415, 407, 403, 396, 362, 347, 335, 328, 321, 300, 284, 278, 244, 235, 228, 204, 191, 176, 167, 159, 150, 127, 113, 104, 80, 69, 55, 44, 34, 19, 990, 972, 948, 935, 916, 897, 882, 865, 852, 831, 812, 799, 782, 768, 757, 744, 720, 705, 689, 678, 657, 644, 626, 609, 596, 587, 566, 548, 534, 522, 506, 494, 469, 450, 431, 414, 388, 373, 358, 345, 329, 314, 299, 277, 262, 255, 245, 223, 199, 187, 177, 149, 129, 109, 99, 87, 74, 63, 53, 32, 14, 3, 0,}

6ος πίνακας : { 671, 421, 627, 319, 168, 16, 899, 834, 765, 599, 495, 480, 352, 311, 253, 234, 92, 995, 973, 957, 940, 926, 892, 881, 844, 832, 820, 814, 794, 787, 770, 729, 723, 712, 676, 650, 619, 581, 570, 559, 509, 484, 465, 459, 440, 408, 393, 378, 359, 331, 282, 219, 210, 195, 148, 145, 143, 115, 91, 48, 31, 27, 11, 992, 985, 975, 967, 954, 934, 920, 912, 906, 896, 880, 873, 849, 836, 817, 803, 795, 785, 775, 763, 749, 743, 739, 722, 711, 701, 684, 674, 668, 660, 639, 621, 602, 588, 576, 567, 550, 545, 527, 510, 501, 492, 473, 464, 455, 443, 429, 426, 418, 413, 406, 401, 375, 353, 341, 332, 326, 306, 292, 280, 268, 241, 233, 221, 198, 184, 175, 166, 153, 147, 125, 110, 96, 78, 65, 49, 39, 33, 8, 989, 968, 947, 929, 913, 895, 879, 864, 851, 830, 811, 796, 780, 767, 756, 737, 716, 703, 688, 672, 655, 642, 624, 606, 594, 582, 562, 543, 532, 519, 505, 490, 467, 448, 430, 411, 387, 372, 357, 344, 327, 312, 297, 275, 261, 254, 243, 222, 197, 186, 173, 140, 128, 107, 98, 85, 73, 59, 47, 30, 10, 2, 0,}

Παράρτημα Β

Προγραμματιζόμενες Λογικές Συσκευές (*Programmable Logic Devices*)

Β.1 Εισαγωγή

Σε αυτήν την ενότητα θα γίνει μια μικρή ιστορική ανάδρομη στο τι προϋπήρχε σε σχέση με την τεχνολογία των ολοκληρωμένων κυκλωμάτων (ΟΚ) και των προγραμματιζόμενων λογικών συσκευών. Αναλύονται οι πιο διαδεδομένοι τύποι ολοκληρωμένων κυκλωμάτων από το ξεκίνημα της τεχνολογίας αυτής μέχρι και τις σημερινές εξελίξεις. Αναλυτικά στην Ενότητα Β.2 παρουσιάζονται τα *τυπικά ολοκληρωμένα κυκλώματα* (*Standard Chips*) και η κατάταξη που έχουν με βάση την κλίμακα ολοκλήρωσης. Στην Ενότητα Β.3 παρουσιάζονται οι *διατάξεις προγραμματιζόμενης λογικής* (*PLDs, Programmable Logic Devices*), με έμφαση στο πώς αυτές δομούνται αλλά και πως αυτές διαφοροποιούνται με βάση την αρχιτεκτονική τους δομή. Τέλος στην Ενότητα Β.4 παρουσιάζονται τα *ειδικά ολοκληρωμένα κυκλώματα* (*ASIC, Application Specific Integrated Circuit*), εστιάζεται ο ρόλος στον οποίο αποσκοπούν και αναλύεται ποια είναι η διαφορά τους με τα άλλα ΟΚ.

Γενικά στην αγορά υπάρχει μια πολύ μεγάλη ποικιλία ΟΚ τα οποία εκτελούν διάφορες λειτουργίες, από τις πιο απλές μέχρι τις πιο πολύπλοκες. Έτσι υπάρχουν ΟΚ πολύ απλής λειτουργικότητας έως εξαιρετικά περίπλοκης. Από αυτά μπορεί κανείς να συμπεράνει ότι κάποια από τα ΟΚ λόγω της ιδιαιτερότητας τους και λόγω διαφορετικών απαιτήσεων, χρειάζονται να σχεδιαστούν από την αρχή ως ένα νέο υπολογιστικό σύστημα. Τη λύση αυτών των νέων σχεδιαστικών απαιτήσεων (προκλήσεων) δίνουν οι τρεις μορφές ΟΚ: τα *τυπικά ΟΚ*, *οι διατάξεις προγραμματιζόμενης λογικής* και τα *ειδικά ολοκληρωμένα κυκλώματα*.

Β.2 Τυπικά Ολοκληρωμένα Κυκλώματα

Τα λογικά κυκλώματα υλοποιούνται ηλεκτρονικά με τη χρήση τρανζίστορ που υπάρχουν στα ΟΚ. Στις πρώτες μορφές τα ΟΚ είχαν μερικές δεκάδες τρανζίστορ, στις μέρες μας έχουν φτάσει σε δεκάδες - εκατοντάδες εκατομμύρια τρανζίστορ με εμβαδό μικρότερο από 100 mm^2.

Τα ΟΚ ανάλογα με την πολυπλοκότητα που έχουν κατατάσσονται σε κατηγορίες, αυτές οι κατηγορίες ονομάζονται κλίμακες ολοκλήρωσης.

Στην αγορά υπάρχουν :

- *Μικρής κλίμακας ολοκλήρωσης* (*SSI, Small Scale Integration*) που περιέχουν μέχρι 10 λογικές πύλες ανά ΟΚ. Πολύ γνωστή σειρά αυτών των ΟΚ είναι η 7400 της εταιρίας *Texas Instruments* [27].

- *Μέσης κλίμακας ολοκλήρωσης* (*MSI, Medium Scale Integration*) που περιέχουν μεταξύ 10 έως και 200 λογικές πύλες ανά ΟΚ. Συνήθως αυτά τα ΟΚ υλοποιούν πολύπλοκες λογικές συναρτήσεις όπως αριθμητικές μονάδες (αθροιστές αφαιρέτες), κωδικοποιητές αποκωδικοποιητές κτλ.

- *Μεγάλης κλίμακας ολοκλήρωσης* (*LSI, Large Scale Integration*) που περιέχουν μεταξύ 200 έως και 200.000 λογικές πύλες ανά συσκευή. Αυτές οι συσκευές υλοποιούν περισσότερο πολύπλοκες λογικές συναρτήσεις από τα προηγούμενα όπως πχ μια μικρή λογική μονάδα, μια μονάδα ελέγχου, κύκλωμα μνημών κτλ.

- *Πολύ Μεγάλης κλίμακας ολοκλήρωσης* (*VLSI, Very Large Scale Integration*) που περιέχουν περισσότερες από 200.000 λογικές πύλες ανά συσκευή. Αυτές οι συσκευές

υλοποιούν συνήθως ψηφιακά συστήματα πχ αυτόνομες μονάδες, μεγάλες μονάδες ελέγχου κτλ.

- *Υπερβολικά μεγάλης κλίμακας ολοκλήρωσης* (U*LSI, Ultra Large Scale Integration*) που περιέχουν εκατομμύρια λογικές πύλες ανά συσκευή. Αυτές οι συσκευές υλοποιούν συνήθως πλήρη ψηφιακά συστήματα πχ υπολογιστές SoC (System on Chip), δηλαδή ένα ΟΚ το οποίο να υλοποιεί ένα ολόκληρο υπολογιστικό σύστημα κτλ.

Τα μικρής κλίμακας ολοκλήρωσης (SSI) και τα μέσης κλίμακας ολοκλήρωσης (MSI) ΟΚ θα μπορούσαν να αναφερθούν και ως τυπικά ΟΚ. Τα υπόλοιπα αποτελούν πολύπλοκες συσκευές με ιδιαίτερα χαρακτηριστικά και δυνατότητες, τα οποία θα δούμε στη συνέχεια.

Κάθε ΟΚ πρέπει να ακολουθεί κάποια χαρακτηριστικά τα οποία να το κάνουν μοναδικό, να βρίσκονται δηλαδή σε συμφωνία με κάποια θεσπισμένα πρότυπα. Η παραγωγή διαφόρων τύπων ΟΚ με αυτά τα χαρακτηριστικά, αποτελούν μια οικογένεια (*family*) ΟΚ της εκάστοτε εταιρίας κατασκευής.

Τα κύρια χαρακτηριστικά ενός ΟΚ είναι :

- *Ταχύτητα* (*speed*) ή η μέγιστη συχνότητα λειτουργίας. Καθορίζεται από την *καθυστέρηση διάδοσης* (*Propagation Delay*) του σήματος μέσα από το ΟΚ, δηλαδή το χρονικό διάστημα από την αλλαγή της κατάστασης της εισόδου μέχρι την αλλαγή της κατάστασης της εξόδου του ΟΚ. Να σημειωθεί εδώ ότι αυτή η καθυστέρηση παίζει πολύ σημαντικό ρόλο στην αξιολόγηση ενός ΟΚ.

- *Fan In – Fan Out.* Ονομάζονται το μεν πρώτο το πλήθος των εισόδων που διαθέτει ένα ΟΚ (ή μια πύλη) και το δεύτερο το πλήθος των εισόδων άλλων ΟΚ (ή πυλών), της ίδια οικογενείας, που μπορεί να τροφοδοτήσει άμεσα η έξοδος αυτού του ΟΚ (ή της πύλης).

- *Ανοχή σε Θόρυβο* (*Noise Immunity*). Είναι ο βαθμός αλλοίωσης που πρέπει να βρεθεί το ηλεκτρικό σήμα ώστε να δώσει λανθασμένο αποτέλεσμα (κατάσταση) στην έξοδο του ΟΚ (ή της πύλης).

- *Ρυθμός βλαβών* (*Failure Rate*) του κυκλώματος. Καθορίζει την αξιοπιστία του κυκλώματος και τη συχνότητα βλαβών.

- *Τάση τροφοδοσίας (Supply Voltage*) και *κατανάλωση ισχύος* (*Power Dissipation*). Αποτελούν δυο παράγοντες πολύ σημαντικούς για την αξιολόγηση ενός ΟΚ, ειδικά στις μέρες μας. Τα ενσωματωμένα συστήματα πολλές φορές αποτελούν αυτόνομα συστήματα, δηλαδή έχουν μπαταρίες για την τροφοδοσία τους, ας αναλογιστεί κανείς ένα τέτοιο ΟΚ να έχει μεγάλη κατανάλωση ισχύος και τάση τροφοδοσίας. Αυτό το ΟΚ θα αποτελέσει την καταστροφή του όποιου συστήματος στο οποίο είναι ενσωματωμένο. Έτσι όταν ένα ΟΚ έχει μηδαμινές καταναλώσεις, το σύστημα θα έχει μακροβιότερη λειτουργία. Ένας ακόμη παράγοντας μεγάλης κατανάλωσης είναι και η ταχύτητα ρολογιού, όσο μεγαλύτερη είναι τόσο μεγάλη κατανάλωση ισχύος έχουμε.

- Το *γινόμενο ταχύτητας ισχύος* (*Speed Power Product*), δηλαδή της καθυστέρησης διάδοσης επί την κατανάλωση ισχύος σε ένα ΟΚ (ή μια πύλη).

- Το κόστος των ΟΚ

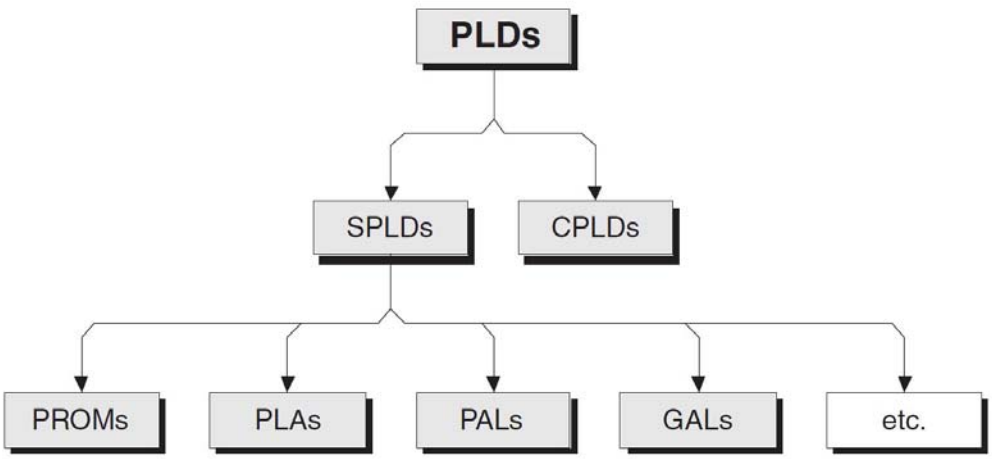
Τα τυπικά ΟΚ ήταν δημοφιλή για τη δημιουργία λογικών κυκλωμάτων έως της αρχές της δεκαετίας 1980. Όμως καθώς εξελισσόταν η τεχνολογία κατασκευής ΟΚ, κατέστη ανώφελη η διάθεση πολύτιμου χορού στις *πλακέτες τυπωμένου κυκλώματος* (*PCB,Printed Circuit Board*), για την τοποθέτηση ΟΚ με περιορισμένες δυνατότητες. Έτσι στράφηκαν σε λύσης οι οποίες να ωφελούν τις εξελίξεις, δηλαδή να υπάρχουν πολύ περισσότερα τρανζίστορ σε μια μικρή cm^2 επιφάνεια. Αυτό αυτόματα σημαίνει ότι πολύ περισσότερες λειτουργίες μπορούν να υλοποιούνται - εκτελούνται σε μια πλακέτας PCB. Επίσης ένα άλλο μειονέκτημα των τυπικών ΟΚ είναι η λειτουργία του καθενός από αυτά είναι συγκεκριμένη και δε μπορεί να μεταβληθεί κατά τη διάρκεια του χρόνου. Για να γίνει κάτι τέτοιο πρέπει να ξανασχεδιαστεί και αμέσως μετά να κατασκευαστεί, διαδικασία που είναι αρκετά χρονοβόρα.

Β.3 Διατάξεις Προγραμματιζόμενης Λογικής PLDs

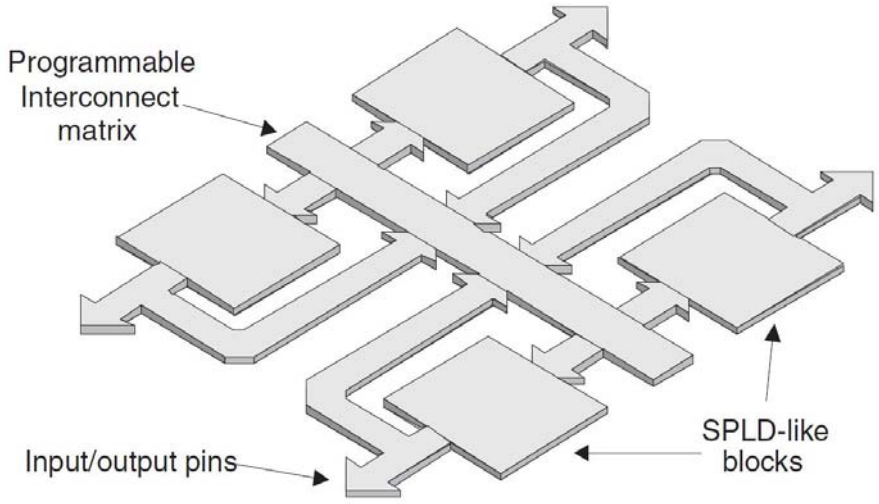
Σε αντίθεση με τα τυπικά ΟΚ που εκτελούν συγκεκριμένες λειτουργίες, υπάρχουν ΟΚ που μπορούν να σχεδιαστούν και να οργανωθούν από το χρήστη, έτσι ώστε να υλοποιούν συγκεκριμένα λογικά κυκλώματα. Αυτό πετυχαίνεται από ειδικά κυκλώματα γενικού σκοπού τα οποία έχουν μια γενική αρχιτεκτονική δομή και περιλαμβάνουν ένα σύνολο από *προγραμματιζόμενους λογικούς διακόπτες* (*PLS, Programmable Logic Switch*), αυτά είναι οι συσκευές LSI,VLSI και ULSI που αναφέρθηκαν και παραπάνω. Οι διακόπτες αυτοί επιτρέπουν τα

εσωτερικά κυκλώματα του ΟΚ να οργανώνονται με διάφορους τρόπους. Ο σχεδιαστής μπορεί να οργανώσει και να σχεδιάσει το λογικό κύκλωμα που θέλει επιλέγοντας την κατάλληλη οργάνωση των διακοπτών αυτών την ώρα της "κατασκευής" του ΟΚ. Οι διακόπτες αυτοί προγραμματίζονται από το τελικό χρήστη και όχι από την κατασκευάστρια εταιρία. Δηλαδή ο χρήστης αφού καταλήξει και σχεδιάσει το κύκλωμα που θέλει στο εργαλείο CAD, το εργαλείο προγραμματίζει τη συσκευή ενεργοποιώντας ή απενεργοποιώντας τους διακόπτες αυτούς. Ο προγραμματισμός μπορεί να γίνει όσες φορές χρειάζεται στις περισσότερες από τις συσκευές αυτές. Αν χρειαστεί για παράδειγμα ένα ΟΚ να εμπλουτιστεί με μια νέα λειτουργία, ή να επαλειφθούν κάποιες παλιές λειτουργίες από αυτό, γίνεται πολύ εύκολα και όσες φορές απαιτείται. Αυτό είναι ίσως το μεγάλο πλεονέκτημα των συσκευών αυτών που ονομάζονται *διατάξεις προγραμματιζόμενης λογικής* (*PLDs, Programmable Logic Devices*). Υπάρχουν πολλά PLDs στην αγορά σε διαφορετικά μεγέθη για να καλύπτουν όλες τις ανάγκες για ΟΚ. Το πλεονέκτημα τους στο να επαναπρογραμματίζονται έναντι των τυπικών ΟΚ, τα έχει κάνει πολύ δημοφιλή και ιδιαίτερα χρησιμοποιημένα στις μέρες μας.

Τα PLDs χωρίζονται σε δυο κατηγορίες τα *απλά PLDs* (*SPLDs, Simple Programmable Logic Devices*) και τα *πολύπλοκα* PLDs (*CPLDs, Complexity Programmable Logic Devices*), Εικόνα Β.1. Πολλοί δεν καταλαβαίνουν τη διάφορα αλλά η αιτία του διαχωρισμού είναι η πολυπλοκότητα τους. Τα SPLDs είναι αυτόνομα ΟΚ τα οποία προγραμματίζονται το καθένα ξεχωριστά, ενώ τα CPLDs αποτελούνται από πολλά αυτόνομα SPLDs τα οποία προγραμματίζονται όλα μαζί, δείτε την Εικόνα Β.2 για να καταλάβετε τη διαφορά τους. Το CPLD περιέχει περισσότερα από ένα SPLD, τα οποία αποτελούν μέρος της αρχιτεκτονικής του δομής.



Εικόνα Β.1: [09] Ιεραρχικές ομάδες PLDs



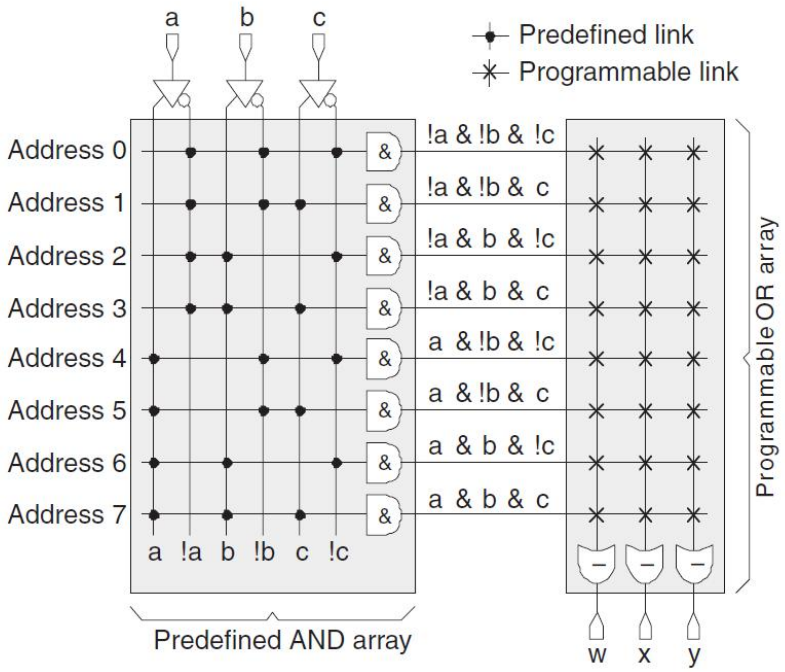
Εικόνα Β.2: [09] Η γενική δομή ενός CPLD

Παρακάτω αναφέρονται τα σημαντικότερα από τα PLDs.

- Μνήμη PROM
- Προγραμματιζόμενη λογική διάταξη (PLA)
- Προγραμματιζόμενη διάταξη λογικής (PAL)
- Πολύπλοκες προγραμματιζόμενες λογικές διάταξης (CPLD)
- Επιτόπου προγραμματιζόμενες διατάξεις πυλών (FPGA).

Β.3.1 Μνήμη PROM

Η πρώτη διάταξη SPLD είναι η *προγραμματιζόμενη μνήμη ROM (PROM, Programmable Read Only Memory*) και εμφανίστηκε το 1970. Όπως φαίνεται και από την Εικόνα Β.3, η συγκεκριμένη μνήμη PROM αποτελείται από τρεις εισόδους όπου κάθε είσοδος έχει και το συμπλήρωμά της κάνοντας χρήση της πύλης NOT. Υπάρχει επίσης μια διάταξη από προκαθορισμένες από το κατασκευαστή πύλες AND και μια διάταξη από προγραμματιζόμενες πύλες OR. Ο χρήστης μπορεί να προγραμματίσει μόνο τις πύλες OR και όχι τις πύλες AND, αυτός ο προγραμματισμός (των πυλών OR) γίνεται μόνο μια φορά και δεν μπορεί να επαναληφθεί, δηλαδή αυτή η μνήμη δεν είναι επαναπρογραμματιζόμενη. Όταν λέμε ότι προγραμματίζεται ένας διακόπτης, ουσιαστικά αυτό που γίνεται είναι να "καεί" σε αυτή τη θέση η ασφάλεια του διακόπτη.



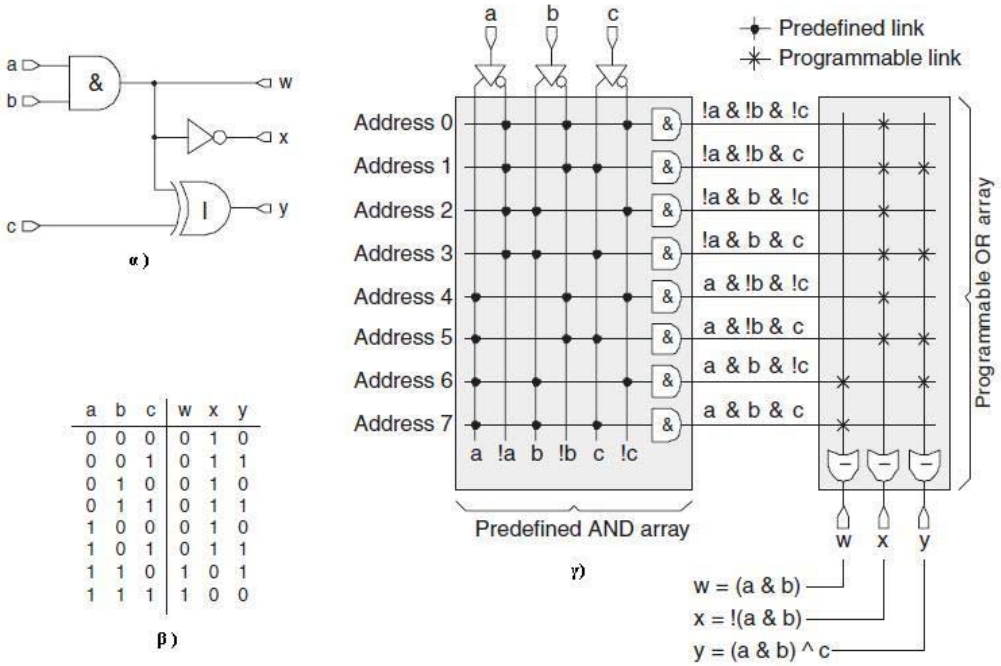
Εικόνα Β.3: [09] Προγραμματιζόμενη μνήμη PROM

Οι μνήμες PROM παρέχουν ευελιξία και είναι βολικές, το οποίο δεν ισχύει στις κοινές μνήμες ROM. Οι τελευταίες είναι οικονομικά ελκυστικές αλλά το κόστος παρασκευής μια μάσκας, η οποία απαιτείται για την αποθήκευση ενός συγκεκριμένου πληροφοριακού προτύπου σε μια μνήμη ROM, τις καθιστά αρκετά ακριβές όταν μόνο ένα μικρό πλήθος από αυτές θα χρειαστεί να κατασκευαστεί. Αυτό το μειονέκτημα έρχεται να εκμεταλλευτεί μια μνήμη PROM, η οποία παρέχει ένα γρηγορότερο και αρκετά φθηνότερο τρόπο κατασκευής, επειδή ακριβώς μπορούν να προγραμματιστούν απευθείας από το χρήστη.

Οι μνήμες PROM μπορούν να χρησιμοποιηθούν σαν μνήμες γενικού σκοπού αλλά και σαν *πίνακες αναζήτησης* (*Lookup Table*), όπου μπορεί ο χρήστης να υλοποιήσει απλές λογικές λειτουργίες - συναρτήσεις. Υπάρχουν μνήμες πολλών εισόδων πολλών εξόδων και γενικά μπορούμε να υλοποιήσουμε Μ συναρτήσεις των Ν εισόδων.

Ας δούμε για παράδειγμα πως λειτουργεί αυτή η μνήμη. Έχουμε ένα κύκλωμα τριών εισόδων a, b, c όπως αυτό που βλέπουμε στην Εικόνα Β.4 στο α). Το κύκλωμα αυτό αποτελείται από μία πύλη AND μια XOR και μία NOT, επίσης έχει τρεις εξόδους τα w,x,y. Ο πίνακας αληθείας του κυκλώματος φαίνεται στην ίδια εικόνα στο β). Στο γ) φαίνεται ότι για κάθε έξοδο η μνήμη PROM έχει ενεργοποιήσει τους κατάλληλους διακόπτες ώστε να δώσει τα σωστά αποτελέσματα. Παράδειγμα : για την έξοδο w ενεργοποιήθηκαν οι δυο τελευταίοι διακόπτες από την πρώτη

στήλη ξεκινώντας από πάνω. Ενώ για την έξοδο x ενεργοποιήθηκαν όλοι οι διακόπτες της στήλης εκτός από τα δύο τελευταία. Όποιος και να είναι ο συνδυασμός των τριών εισόδων της μνήμης PROM θα ενεργοποιηθεί η κατάλληλη έξοδος. Μπορεί να καθοριστεί απευθείας ο πίνακα αληθείας των συναρτήσεων που χρειάζεται να υλοποιηθούν στη μνήμη PROM, χωρίς να προβεί σε καμία απλοποίηση.



Εικόνα Β.4: [09] Παράδειγμα προγραμματιζόμενης μνήμης PROM

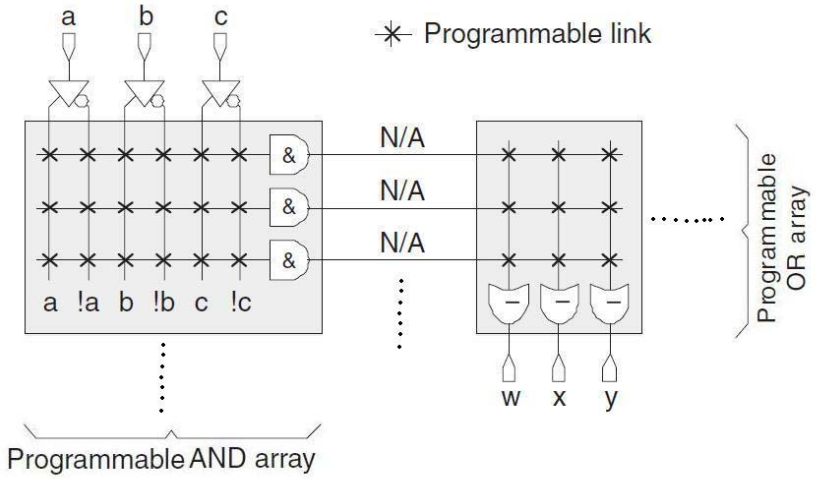
Μεταγενέστερη εξέλιξη των μνημών PROM είναι η *επαναπρογραμματιζόμενη μνήμη PROM* ή *EPROM* (*Erasable Programmable Read Only Memory*). Ο επαναπρογραμματισμός της μνήμης EPROM γίνεται με την απομάκρυνση των φορτίων που βρίσκονται στα τρανζίστορ. Αυτό γίνεται με την απόσπαση του ΟΚ από το φυσικό κύκλωμα και την έκθεση του σε υπεριώδες φως. Αυτό το μειονέκτημα οδήγησε στη δημιουργία επαναπρογραμματιζόμενων μνημών που λειτουργούν ηλεκτρονικά, τα επονομαζόμενα *EEPROM (Electrical Erasable Programmable Read Only Memory)*. Για να απαλειφθούν τα περιεχόμενα τους, οι μνήμες EEPROM δε χρειάζονται να αποσπαστούν από το φυσικό κύκλωμα στο οποίο ανήκουν, γιατί αυτό γίνεται τοπικά. Ένα ακόμη πλεονέκτημα είναι ότι μπορεί να γίνει επιλεκτική διαγραφή των πληροφοριών που χρειάζεται και όχι ολόκληρης της μνήμης όπως υφίσταται στη μνήμη EPROM.

Μια προσέγγιση παρεμφερής με την τεχνολογία μνημών EEPROM, έχει οδηγήσει στην ανάπτυξη συσκευών μνήμης *flash* (*flash memory*). Υπάρχουν ομοιότητες αλλά και διαφορές μεταξύ μια μνήμης EEPROM και μια μνήμης flash. Στη μνήμη EEPROM μπορεί να γίνει ανάγνωση και εγγραφή μίας κυψελίδας (ενός bit). Ενώ σε μια μνήμη flash, καθίσταται μεν δυνατή η ανάγνωση μίας κυψελίδας όπως και σε μια EEPROM, αλλά η εγγραφή μπορεί να γίνει μόνο σε επίπεδο τμήματος του πίνακα των κυψελίδων και όχι καθεμία ξεχωριστά. Πριν την εγγραφή τα προηγούμενα περιεχόμενα του τμήματος απαλείφονται. Οι συσκευές flash έχουν μεγαλύτερη πυκνότητα κάτι που τις οδηγεί σε αυξημένη χωρητικότητα και χαμηλότερο κόστος ανά bit. Απαιτούν μία και μοναδική τάση ρεύματος και καταναλώνουν λιγότερη ενέργεια κατά τη λειτουργία τους. Η χαμηλή κατανάλωση ενέργειας των μνημών flash τις καθιστά ιδιαίτερα ελκυστικές σε φορητές συσκευές οι οποίες ρευματοδοτούνται με μπαταρίες. Κατασκευές της τεχνολογίας flash είναι οι κάρτες flash τα γνωστά μας *USB flash* και οι δίσκοι flash εσωτερικής η εξωτερικής χρήσης με πολύ μεγάλη χωρητικότητα.

Όλες οι παραπάνω τεχνολογίες που παρουσιάσθηκαν EPROM, EEPROM και flash αποτελούν επέκταση της τεχνολογίας PROM άρα και επέκταση της ομάδας SPLD.

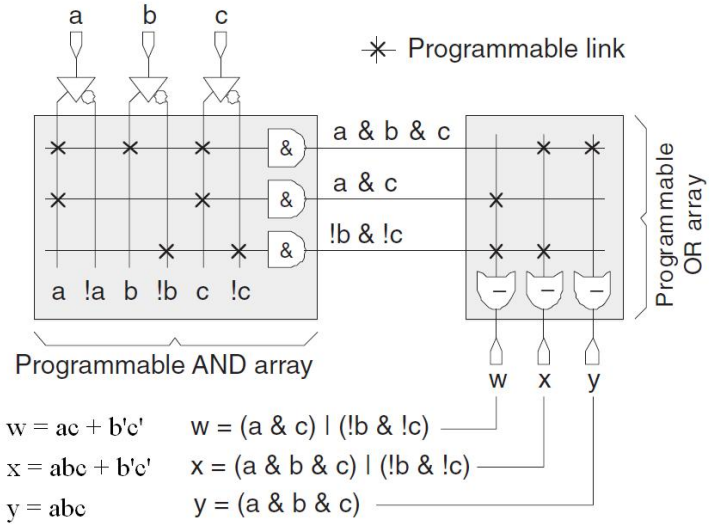
Β.3.2 Προγραμματιζόμενη Λογική Διάταξη PLA

Προγραμματιζόμενη λογική διάταξη (*PLA, Programmable Logic Array*), πρωτοεμφανίστηκε το 1975. Στηριζόμενο στο γεγονός ότι οι λογικές συναρτήσεις μπορούν να υλοποιηθούν με τη μορφή του αθροίσματος γινομένων, το PLA αποτελείται από ένα σύνολο προγραμματιζόμενων πυλών AND που τροφοδοτούν ένα σύνολο προγραμματιζόμενων πυλών OR, δηλαδή και τα δυο σύνολα μπορούν να προγραμματιστούν, Εικόνα Β.5. Ο αριθμός των συναρτήσεων που υλοποιούνται στη συστοιχία των πυλών AND είναι ανεξάρτητος από τον αριθμό των εισόδων της διάταξης. Το ίδιο συμβαίνει και με τις συναρτήσεις που υλοποιούνται στη συστοιχία των πυλών OR, δηλαδή είναι ανεξάρτητες από τον αριθμό των συναρτήσεων που υλοποιούνται στη συστοιχία των πυλών AND και των εισόδων της διάταξης. Η συστοιχία πυλών AND παράγει ένα αριθμό όρων γινομένου έστω P_1, P_2,….P_k, τα γινόμενα αυτά ενεργούν ως είσοδοι ενός επιπέδου OR, το οποίο παράγει τις εξόδους. Η κάθε έξοδος μπορεί να οργανωθεί έτσι ώστε να αποτελεί άθροισμα οποιονδήποτε όρων του συνόλου P_1, P_2,….P_k και έτσι προκύπτει η λειτουργία του αθροίσματος γινομένων που επιτελεί η διάταξη PLA. Από τα παραπάνω προκύπτει ότι το μέγεθος ενός PLA εξαρτάται από τη μορφή του αθροίσματος γινομένων της εισόδου της διάταξης.



Εικόνα Β.5: [09] Προγραμματιζόμενη λογική διάταξη PLA

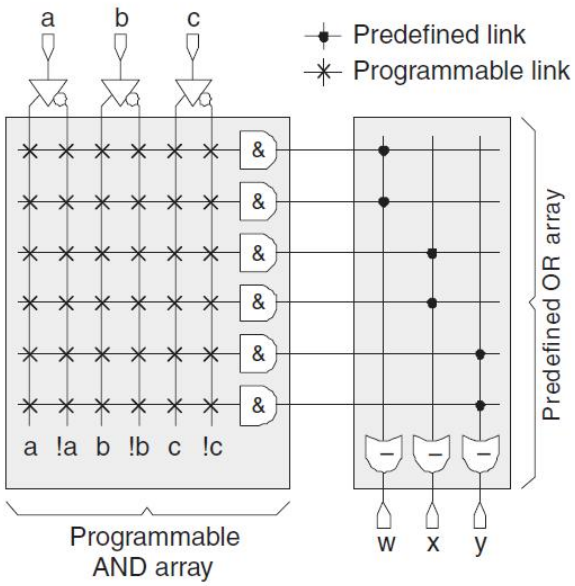
Ας εξεταστή ένα παράδειγμα προγραμματισμού μιας διάταξης PLA, έστω ότι υπάρχουν τα αθροίσματα γινομένων της Εικόνας Β.6. Όπως μπορεί κανείς να διαπιστώσει από τη διαμόρφωση που έχει γίνει, η συστοιχία των πυλών AND έχει διαμορφώσει τους κατάλληλους διακόπτες ώστε να υλοποιούν τα γινόμενα *abc, ac* και *b'c'*. Κατόπιν τα εισάγει στη συστοιχία των πυλών OR και εκεί γίνεται πάλι διαμόρφωση σε κατάλληλους διακόπτες ώστε υλοποιούν τα αθροίσματα των συναρτήσεων. Όπως μπορεί να φανεί και από την έξοδο w έχουν ενεργοποιηθεί οι δυο τελευταίοι διακόπτες από την πρώτη στήλη της συστοιχίας των πυλών OR, όπου υλοποιούν τη συνάρτηση *w = ac + b'c'*. Κατά παρόμοιο τρόπο έχουν διαμορφωθεί και οι άλλες δυο έξοδοι.



Εικόνα Β.6: [09] Παράδειγμα προγραμματιζόμενης λογικής διάταξης PLA

Β.3.3 Προγραμματιζόμενη Διάταξη Λογικής PAL

Μια άλλη διάταξη SPLD είναι η *Προγραμματιζόμενη διάταξη λογικής* PAL, η οποία πρωτοεμφανίστηκε και αυτή το 1970 όπως και οι μνήμες PROM. Οι διατάξεις PAL ουσιαστικά επιτελούν την αντίθετη λειτουργία από τις μνήμες PROM. Όπως οι μνήμες PROM έτσι και οι διατάξεις PAL είναι δύο επιπέδων με συστοιχίες πυλών AND και OR, στις μεν μνήμες PROM ο χρήστης μπορεί να προγραμματίσει τη συστοιχία πυλών OR, στις διατάξεις PAL δε ο χρήστης μπορεί να προγραμματίσει τη συστοιχία των πυλών AND και να μείνει ανεπηρέαστη η συστοιχία των πυλών OR η οποία είναι προκαθορισμένη από το κατασκευαστή, Εικόνα Β.7. Μπορούν να υλοποιηθούν συναρτήσεις χρησιμοποιώντας τους διαθέσιμους *ελαχιστόρους* (*minterms*). Οι διατάξεις GAL (Generic Array Logic) είναι ένας άλλος τύπος διάταξης PAL.

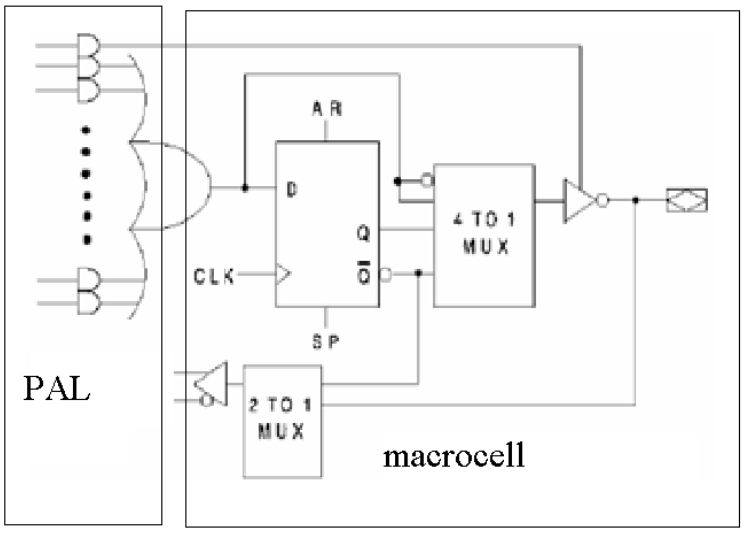


Εικόνα Β.7: [09] Προγραμματιζόμενη διάταξη λογικής PAL

Οι διατάξεις PLA είναι προγραμματιζόμενες και στα δυο πεδία συστοιχιών των πυλών AND και OR, αυτό είχε σαν αποτέλεσμα να παρουσιαστούν δυο μειονεκτήματα, το πρώτο έχει να κάνει με τη δυσκολία κατασκευής τους με ακρίβεια και το δεύτερο προκαλούσε μείωση της συνολικής ταχύτητας του κυκλώματος στο οποίο συνδέονταν. Αυτά τα μειονεκτήματα οδήγησαν στην εξέλιξη των διατάξεων PAL και τα έκαναν ιδιαίτερα δημοφιλή τόσο για την ακρίβεια κατασκευής όσο και για τη γρήγορη ταχύτητα τους (αυτό συνεπάγεται και καλύτερη απόδοση). Επιπλέον είναι πολύ οικονομικότερα από τις διατάξεις PLA και γι' αυτό χρησιμοποιούνται ευρέως σε πρακτικές εφαρμογές. Μια πολύ σημαντική αίτια της ταχύτητας των PAL έναντι των PLA, είναι

ότι στις πρώτες προγραμματίζεται μόνο μια συστοιχία πυλών, ενώ αντίθετα στις δεύτερες πρέπει να προγραμματιστούν δυο συστοιχίες πυλών.

Σε πολλά κυκλώματα PAL τοποθετούνται επιπρόσθετα κυκλώματα στις εξόδους των πυλών OR ώστε να παρέχεται επιπλέον ευελιξία. Συνηθίζεται να χρησιμοποιείται ο όρος *μακροκυψέλη* (*macrocell*) για να αναφέρεται σε πύλες OR με επιπλέον κυκλώματα, Εικόνα Β.8. Αυτό ουσιαστικά είναι και ένα σημείο αναφοράς για την εξέλιξη των macrocell σε *λογικά στοιχεία* (*LE, Logic Element*) τα οποία χρησιμοποιούνται και αποτελούν το κύριο λογικό στοιχείο μιας αρχιτεκτονικής FPGA, για περισσότερες λεπτομερές γύρω από τα λογικά στοιχεία (LE) ο αναγνώστης μπορεί ν ανατρέξει στην Ενότητα 2.2.7.



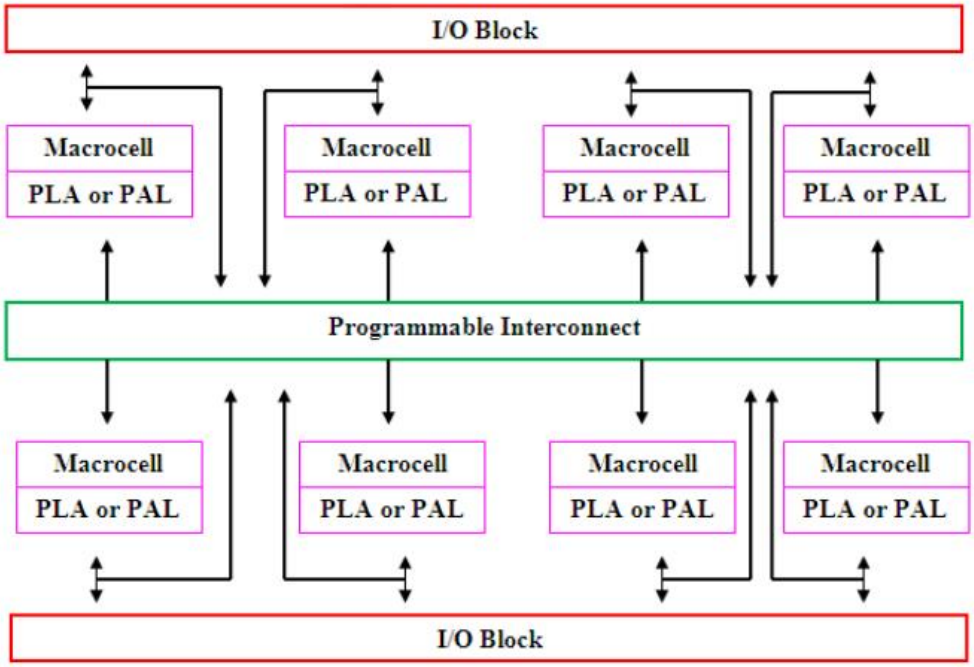
Εικόνα Β.8: Παράδειγμα μιας μακροκυψέλης (macrocell) σε διάταξη PAL

Β.3.4 Πολύπλοκες Προγραμματιζόμενες Λογικές Διάταξης CPLDs

Μέχρι στιγμής έχουν αναφερθεί μόνο ΟΚ SPLD, όπως αναφέρθηκε το χαρακτηριστικό τους είναι ότι είναι μικρά και αυτόνομα ΟΚ, αν θελήσει κάποιος να χρησιμοποιήσει ένα ΟΚ αρκετά πολύπλοκο ώστε να περιέχει περισσότερα από δύο SPLD, τότε πρέπει να αναφερθεί στις *πολύπλοκες προγραμματιζόμενες λογικές διατάξεις* (*CPLDs, Complexity Programmable Logic Devices*), οι διατάξεις αυτές πρωτοεμφανίστηκαν μεταξύ του τέλους της δεκαετίας του ‘70 και αρχές δεκαετίας ‘80.

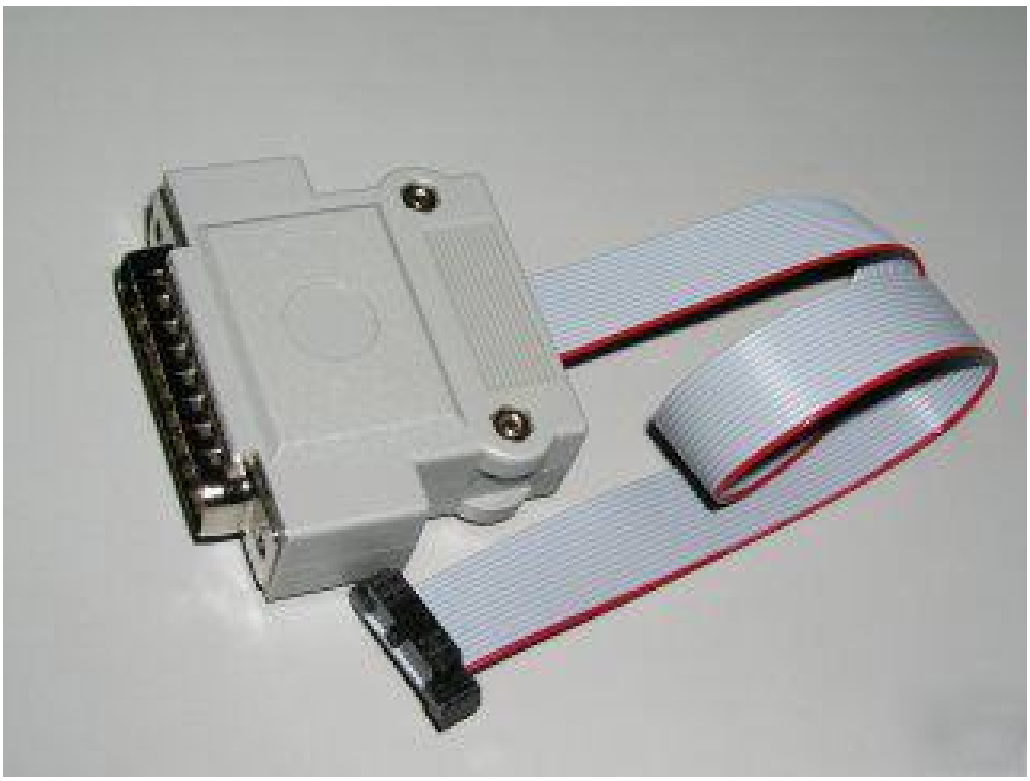
Ένα CPLD αποτελείται από πολλές βαθμίδες κυκλωμάτων που βρίσκονται μέσα στο ίδιο ΟΚ και συνδέονται μεταξύ τους με εσωτερικές καλωδιώσεις. Κάθε βαθμίδα κυκλώματος μοιάζει με ένα SPLD, συνήθως είναι PAL η PLA. Κάθε τέτοια βαθμίδα συνδέεται επίσης με ένα υποκύκλωμα, που

ονομάζεται *βαθμίδα εισόδου / εξόδου* (*I/O Block*) και προσαρμόζεται σε ένα αριθμό ακροδεκτών εισόδου και εξόδου του ΟΚ, Εικόνα Β.9. Οι εσωτερικές καλωδιώσεις περιέχουν *προγραμματιζόμενους διακόπτες διασύνδεσης* (*PIS, Programmable Interconnect Switches*), οι οποίοι χρησιμοποιούνται για να διασυνδέσου μεταξύ τους τις βαθμίδες SPLD. Στο εμπόριο τα CPLD χρησιμοποιούνται για την υλοποίηση πολλών ειδών ψηφιακών κυκλωμάτων και έχουν μεγέθη που κυμαίνονται μεταξύ δυο βαθμίδων SPLD και περισσοτέρων από εκατό SPLD.



Εικόνα Β.9: Γενική αρχιτεκτονική δομή ενός CPLD

Τα κυκλώματα της διάταξης CPLD προγραμματίζονται μέσω μια θύρας που ονομάζεται *JTAG* (*JTAG port, Joint Test Action Group*), Εικόνα Β.10. Αυτή η διαδικασία προγραμματισμού έχει προτυποποιηθεί από τον οργανισμό *IEEE* [12] τόσο για συσκευές CPLD όσο και για συσκευές FPGA. Εφόσον προγραμματιστεί ένα CPLD, διατηρεί μόνιμα την πληροφορία που έχει τοποθετηθεί σε αυτήν ακόμη και αν διακοπεί η τροφοδοσία του ΟΚ. Η διαδικασία αυτή ονομάζεται *μη-πτητικός προγραμματισμός* ή *μόνιμος προγραμματισμός* (*non-volatile programming*).



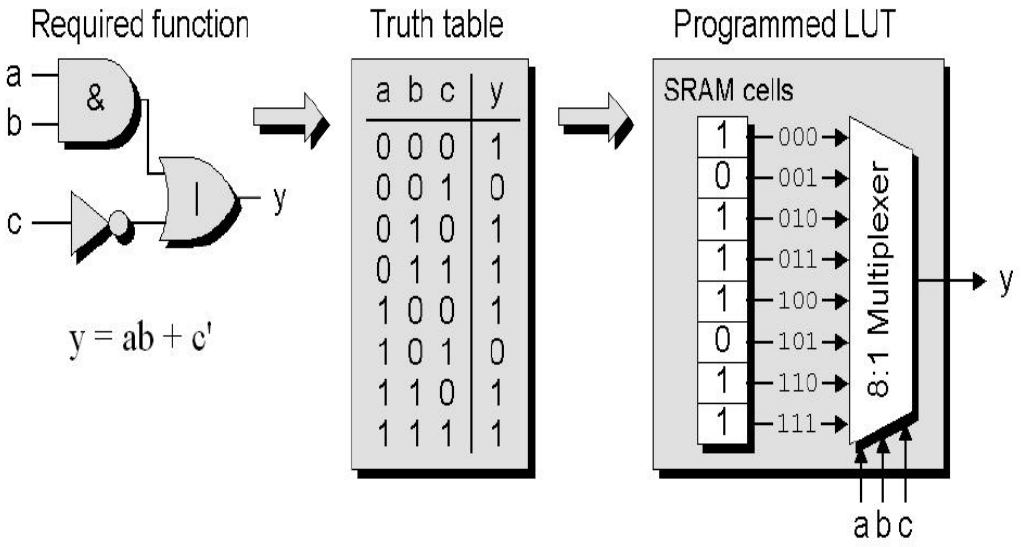
Εικόνα Β.10: Θύρα JTAG της εταιρίας Xilinx για CPLD και FPGA

Β.3.5 Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών FPGAs

Οι μορφές ΟΚ που έχουν περιγράφει έως αυτό το σημείο, η οικογένεια 7400 και τα SPLD και CPLD, είναι χρήσιμες για την υλοποίηση ενός μεγάλου αριθμού λογικών κυκλωμάτων. Εκτός από τα CPLD, τα παραπάνω ΟΚ είναι εν γένει μικρού μεγέθους και είναι κατάλληλα μόνο για σχετικά απλές εφαρμογές. Ακόμη και στην περίπτωση των CPLD, μόνο μετρίως μεγάλα λογικά κυκλώματα μπορούν να χωρέσουν σε ένα ΟΚ. Για την υλοποίηση μεγαλύτερων κυκλωμάτων είναι βολικό να χρησιμοποιηθεί ένα διαφορετικό είδος ΟΚ που έχει μεγαλύτερη χωρητικότητα. Μια από τις πλέον εξελιγμένες εκδοχές των PLDs είναι οι *Επιτόπου Προγραμματιζόμενες Διατάξεις Πυλών FPGAs* (*Field Programmable Gate Arrays*). Τα FPGAs είναι διατάξεις προγραμματιζόμενης λογικής και υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων και επιτρέπουν πλήρη ελευθερία όσον αφορά τη διαδικασία σχεδίασης. Τα FPGAs είναι συσκευές γενικής χρήσης, οι οποίες εμπεριέχουν ένα μεγάλο αριθμό *λογικών στοιχείων* (*LE, Logic Element*), καλωδίων διασύνδεσης και διακοπτών. Η συγκεκριμένη διάταξη δεν φαίνεται στην Εικόνα Β.1, αλλά αποτελεί μέλος της οικογενείας PLDs όπως θα δείτε και από τα παρακάτω.

Η αρχιτεκτονική δομή των FPGAs διαφέρει σημαντικά από αυτή των διατάξεων SPLD και CPLD επειδή δεν περιέχουν πύλες AND και OR, αλλά περιέχει *λογικές βαθμίδες* για την υλοποίηση των ζητούμενων συναρτήσεων. Η πιο ευρέως χρησιμοποιημένη λογική βαθμίδα είναι ο *πίνακας*

αναζήτησης (*LUT, Lookup Table*), Εικόνα Β.11, ο οποίος περιέχει *κυψέλες αποθήκευσης* (*storage cells*) που χρησιμοποιούνται για την υλοποίηση μιας μικρής συνάρτησης. Κάθε κυψέλη είναι μια μνήμη τύπου *SRAM* (*Static Random Access Memory*) και μπορεί να αποθηκεύσει μια λογική τιμή, 0 ή 1. Η αποθηκευμένη τιμή μεταφέρεται στην έξοδο της κυψέλης αποθήκευσης. Μπορούν να αναπτυχθούν πίνακες LUT σε διάφορα μεγέθη, όπου το μέγεθος ορίζεται από τον αριθμό των εισόδων. Ένα LUT n-εισόδων μπορεί να υλοποιήσει όλες τις πιθανές συνδυαστικές συναρτήσεις των n-εισόδων, προσθέτοντας μία ακόμη είσοδο είναι δυνατή η αναπαράσταση πιο σύνθετων συναρτήσεων. Μελέτες έχουν δείξει ότι τα LUT 4-εισόδων είναι μία "καλή" λύση [40].



Εικόνα Β.11: [09] Πίνακας αναζήτησης (LUT, Lookup Table) αρχιτεκτονικής FPGA

Οι κυψέλες αποθήκευσης των πινάκων LUT είναι *πτητικές* ή *μη μόνιμες* (*volatile*), αυτό συνεπάγεται ότι χάνουν τα δεδομένα που περιέχουν εάν διακοπεί η τροφοδοσία του ΟΚ. Δεδομένου αυτού του προβλήματος, θα έπρεπε το FPGA να προγραμματίζεται κάθε φορά που θα εφαρμόζονταν η τροφοδοσία στο κύκλωμα. Για την αποφυγή αυτού του προβλήματος, συμπεριλαμβάνεται συχνά μαζί με τη μητρική πλακέτα του FPGA και ένα μικρό ΟΚ μη-πτητικής μνήμης. Αυτή η μνήμη είναι της μορφής PROM ή EPROM ή EEROM ή οτιδήποτε άλλο. Η μνήμη αυτή διατηρεί σε μόνιμη βάση τα δεδομένα της διάταξης τα οποία έχουν εισέλθει μέσω της θύρας JTAG. Κάθε φορά που γίνεται τροφοδοσία στη συσκευή FPGA, διαβάζεται η μνήμη και διαμορφώνεται στη συσκευή η συγκεκριμένη λειτουργία για την οποία σχεδιάστηκε. Για περισσότερες λεπτομέρειες γύρω από την αρχιτεκτονική δομή ενός FPGA δείτε την Ενότητα 2.2.

Β.4 Ειδικά Ολοκληρωμένα Κυκλώματα ASICs

Ένα πολύ μεγάλο μειονέκτημα των συσκευών PLDs είναι ότι οι προγραμματιζόμενοι διακόπτες που διαθέτουν, καταλαμβάνουν πολύτιμο χώρο πάνω στο ΟΚ και περιορίζουν την ταχύτητα λειτουργίας των κυκλωμάτων που υλοποιούνται με αυτά. Επομένως σε μερικές περιπτώσεις τα PLDs ενδέχεται να μη πληρούν τις προδιαγραφές επίδοσης που επιθυμεί ο χρήστης ή το κόστος αγοράς τους να μην είναι επιτρεπτό. Στις περιπτώσεις αυτές θα ήταν προτιμότερο να κατασκευαστεί ένα ΟΚ από την αρχή, ώστε να μπορέσει να εκπληρώσει τα επιθυμητά αποτελέσματα φθηνότερα αλλά και αποτελεσματικότερα. Τέτοια κυκλώματα που προορίζονται για χρήση σε συγκεκριμένες εφαρμογές ονομάζονται *ειδικά ολοκληρωμένα κυκλώματα εφαρμογής* (*ASICs, Application-Oriented Integrated Circuit*). Η προσέγγιση που γίνεται είναι *ειδική σχεδίαση* (*custom design*) για συγκεκριμένη λειτουργία και αυτά τα ΟΚ ονομάζονται *ειδικά ΟΚ* (*custom chips*) γιατί επιτελούν μια ειδική λειτουργία (Εικόνα Β.12).

Το κύριο πλεονέκτημα των ASICs είναι ότι η σχεδίαση τους μπορεί να βελτιστοποιηθεί προς την κατεύθυνση κάποιας λειτουργίας και αυτό οδηγεί συνήθως σε καλύτερες επιδόσεις. Επιπλέον το κόστος παραγωγής τέτοιων ΟΚ είναι υψηλό, αλλά συνήθως το συγκεκριμένο προϊόν όταν πρόκειται να πουληθεί σε μεγάλες ποσότητες, το κόστος ανά τεμάχιο μπορεί να μειωθεί πολύ και να γίνει μικρότερο από το κόστος ενός εμπορικού τεμαχίου που εκτελεί την ίδια λειτουργία. Επιπρόσθετα, εάν μπορεί να χρησιμοποιηθεί για την εκτέλεση μιας λειτουργίας ένα ΟΚ αντί για δυο ή περισσότερα, εξοικονομείται χώρος στην πλακέτα (PCB) του τελικού προϊόντος και έτσι το κόστος μειώνεται ακόμη περαιτέρω.

Το μεγάλο μειονέκτημα των συσκευών ASICs είναι ότι, ακριβώς όπως προαναφέρθηκε, εκτελούν μια συγκεκριμένη λειτουργία. Αν κατά τη χρήση της συγκεκριμένης συσκευής προκύψουν νέες λειτουργίες τότε η συσκευή ASICs δεν θα μπορέσει να ανταποκριθεί στις νέες προκλήσεις. Θα πρέπει να γίνει μια νέα σχεδίαση, να συμπεριληφθούν οι νέες λειτουργίες και στη συνέχεια να κατασκευαστεί εκ νέου. Αυτή η διαδικασία είναι χρονοβόρα, διαρκεί 6 με 8 μήνες, οπότε είναι μη λειτουργικό να χρησιμοποιείς συσκευές ASICs σε καταστάσεις που δεν είναι σταθερές. Αυτό όμως είναι το μεγάλο πλεονέκτημα των συσκευών PLDs, δηλαδή ότι μπορούν να επαναπρογραμματίζονται με μηδαμινό κόστος.

Εικόνα Β.12: Ειδικό Ολοκληρωμένο Κύκλωμα ASIC

Παράρτημα Γ

Αρχιτεκτονική Δομή Xilinx Spartan -3 FPGA

Σε αυτήν την ενότητα θα παρουσιαστεί η αρχιτεκτονική δομή της οικογένειας συσκευών Spartan-3 FPGA [35,36] της εταιρίας Xilinx [37].

Γ.1 Αρχιτεκτονική και Γενικά Χαρακτηριστικά

Η αρχιτεκτονική δομή που θα παρουσιαστεί είναι της οικογένειας συσκευών Spartan-3 FPGA της εταιρίας Xilinx. Οι επεκτάσεις αυτής της οικογένειας είναι οι ομάδες συσκευών *Spartan-3A* και *Spartan-3E*. Θα εξεταστούν οι συσκευές Spartan-3 άλλα σε κάποιους από τους πίνακες και τις εικόνες που θα παρουσιαστούν, θα γίνεται αναφορά και σε κάποιες συσκευές των επεκτάσεων Spartan-3A και Spartan-3E. Κατασκευαστικά οι περισσότερες συσκευές Spartan σχετίζονται στενά με κάποιο άλλο προϊόν της Xilinx.

Κάποια σύντομα χαρακτηριστικά της συσκευής είναι τα εξής: τα σήματα της συσκευής κυμαίνονται ανάμεσα σε 1,14V και 3,465V, μονοί και διαφορικοί ακροδέκτες, 26 πρότυπα

εισόδου-εξόδου (I/O), 8 βαθμίδες εισόδου-εξόδου (I/O bank), μέχρι 1916928 bits μνήμη RAM και μέχρι 74,880 λογικά στοιχεία κτλ. Επιπλέον χαρακτηριστικά φαίνονται παρακάτω. Στο Πίνακα Γ.1 φαίνονται οι συσκευές που παρέχονται από τον κατασκευαστή για την οικογένεια Spartan-3.

Η Συσκευές της Οικογένειας Spartan-3 Προσφέρουν τις Ακόλουθες Δυνατότητες:

- Χαμηλό κόστος, υψηλής απόδοσης λύσεις για δεδομένα υψηλού όγκου, εφαρμογές προσανατολισμένες προς τον καταναλωτή.
  - Πυκνότητα έως 74.880 λογικά στοιχεία

- Επιλογή I/O διασύνδεσης.
  - Μέχρι 633 I / O ακροδέκτες.
  - 622 + Mb/s ταχύτητα μεταφοράς δεδομένων ανά I/O.
  - 18 single - ended πρότυπα σήματα.
  - 8 διαφορετικά I/O πρότυπα, συμπεριλαμβανομένων LVDS, RSDS.
  - Σήματα που κυμαίνονται από 1.14V έως 3.465V.
  - Υποστήριξη Double Data Rate (DDR).
  - Υποστήριξη DDR, DDR2 SDRAM έως και 333 Mbps.

- Λογικοί πόροι.
  - Συγκέντρωση των λογικών στοιχείων με δυνατότητα ολίσθησης καταχωρητή.
  - Ευρύς, γρήγορους πολυπλέκτες.
  - Γρήγορη πρόβλεψης κρατουμένου.
  - Αφιερωμένους 18 x 18 πολλαπλασιαστές.
  - JTAG λογική συμβατή με IEEE 1149.1/1532.

- Επιλογή RAM ιεραρχικής μνήμης.
  - Έως και 1.872 Kbits του συνόλου μπλοκ RAM.
  - Έως και 520 Kbits διανέμονται της συνολικής RAM.

- Ψηφιακό ρολόι διαχείρισης (έως και τέσσερις DCMs).
  - Εξάλειψη φαινομένου Clock Skew.
  - Σύνθεση συχνοτήτων.
  - Υψηλής ανάλυσης μετατόπιση φάσης.

- Οκτώ γενικές γραμμές ρολογιού για δρομολόγηση.
- Πλήρης υποστήριξη από την Xilinx ISE ® και WebPACK ™ λογισμικά συστήματα ανάπτυξης
- MicroBlaze ™ και PicoBlaze ™ επεξεργαστή, PCI ®, PCI Express ® PIPE Endpoint, και άλλους πυρήνες IP.

Summary of Spartan-3 FPGA Attributes											
Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	633	300
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	633	300

Πίνακας Γ.1: [35] Οικογένεια συσκευών Spartan-3 και τα χαρακτηρίστηκα κάθε μιας

Η Spartan-3 οικογένεια αποτελείται από πέντε βασικά προγραμματιζόμενα λειτουργικά στοιχεία:

- Οι διαμορφωμένες λογικές βαθμίδες (*CLBs, Configurable Logic Blocks*) περιέχουν μνήμες RAM που βασίζονται σε πίνακες αναζήτησης (LUTs). Τα CLBs μπορούν να προγραμματιστούν έτσι ώστε να εκτελούν μια ευρεία ποικιλία από λογικές

λειτουργίες, εκτελώντας τις λογικές συναρτήσεις που παρουσιάζονται, καθώς και για αποθήκευση δεδομένων.

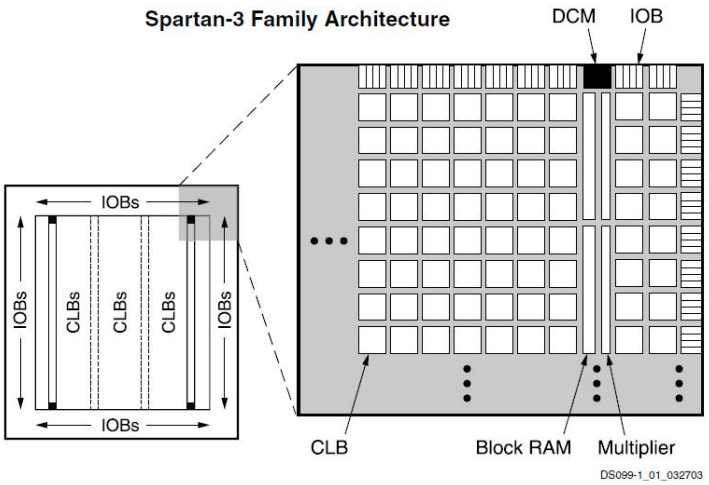
- Λογικά σύνολα εισόδου - εξόδου (*IOBs, Input / Output Blocks*) ελέγχουν τη ροή των δεδομένων μεταξύ των ακροδεκτών εισόδου εξόδου (I/O pins) και την εσωτερική λογική της συσκευής. Κάθε IOB υποστηρίζει αμφίδρομη ροή δεδομένων συν τρισταθής λειτουργία, δηλαδή παρέχεται και μία προγραμματιζόμενη διάταξη τριών καταστάσεων (*Three - State*). Είναι διαθέσιμα επιπλέον είκοσι έξι διαφορετικά πρότυπα σήματος (I/O), μεταξύ των οποίων οκτώ διαφορικά πρότυπα υψηλής απόδοσης.

- Οι βαθμίδες μνήμης (*RAM Block*) παρέχουν αποθηκευτικό χώρο για τα δεδομένα, είναι της μορφής 18Kbit διπλής θύρας (Dual Port).

- Οι βαθμίδες πολλαπλασιαστών (*Multipliers Block*) δέχονται δύο 18bit δυαδικούς αριθμούς ως είσοδο και υπολογίζουν το αποτέλεσμα βγάζοντας ένα 36bit δυαδικό αριθμό ως έξοδο.

- Ο διευθυντής ρολογιού (DCM, Digital Clock Manager) παρέχει αυτόματη βαθμονόμηση, πλήρες ψηφιακές λύσεις για διανομή, καθυστέρηση, πολλαπλασιασμό, διαίρεση καθώς και μετατόπιση φάσης, του σήματος ρολογιού.

Η γενική αρχιτεκτονική που ακολουθείται είναι της μορφής της Εικόνας Γ.1. Τα παραπάνω στοιχεία οργανώνονται όπως μπορεί να διακρίνει κανείς σε, διαμορφωμένες λογικές βαθμίδες (CLB), βαθμίδες μνημών RAM, βαθμίδες πολλαπλασιαστών (Multipliers) καθώς και βαθμίδες εισόδου εξόδου (IOB). Η οικογένεια Spartan-3 διαθέτει ένα πλούσιο δίκτυο ιχνών και διακοπτών για τη διασύνδεση των πέντε παραπάνω λειτουργικών στοιχείων. Κάθε λειτουργικό στοιχείο έχει ένα σχετικό πίνακα διακοπτών που επιτρέπει πολλαπλές συνδέσεις στο δίκτυο δρομολόγησης.

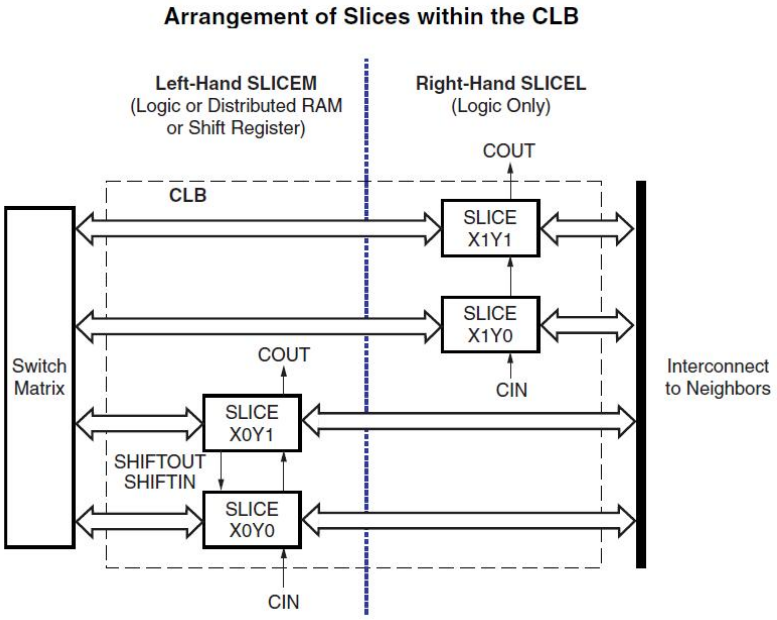


Notes:
1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the far left.

Εικόνα Γ.1: [35] Γενική αρχιτεκτονική δομή των συσκευών Spartan-3

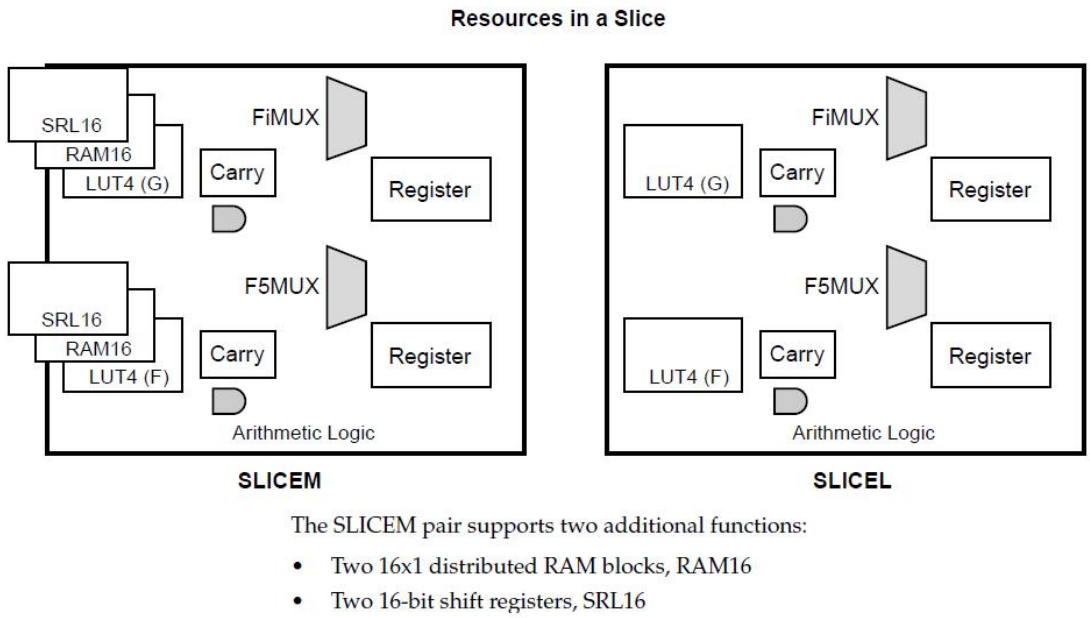
Γ.2 Διαμορφωμένες Λογικές Βαθμίδες (CLBs)

Οι διαμορφωμένες λογικές βαθμίδες (CLBs) αποτελούν την κύρια πηγή λογικής για την εφαρμογή σύγχρονων και συνδυαστικών κυκλωμάτων. Κάθε CLB περιέχει τέσσερα λογικά κυκλώματα τα οποία ονομάζονται ”φέτες” (Slices), Εικόνα Γ.2. Το κάθε CLB περιέχει 2 *SLICEL* τα οποία μπορούν να υλοποιήσουν μόνο λογική και 2 *SLICEM* τα οποία μπορούν να υλοποιήσουν λογική, μνήμη και ολίσθηση. Στην Εικόνα Γ.2 φαίνεται επίσης η μήτρα μεταγωγής (*Switch Matrix*) με την οποία το CLB συνδέεται με τις διάφορες πηγές δρομολόγησης.



Εικόνα Γ.2: [35] Δομή μιας διαμορφωμένης λογικής βαθμίδας (CLBs) της οικογένειας FPGA Spartan-3

Κάθε Slice περιέχει δύο πίνακες αναζήτησης (LUT) των 4 εισόδων (4-Input LUT) για την εφαρμογή οποιασδήποτε λογικής συνάρτησης και δύο ειδικά αποθηκευτικά στοιχεία που μπορούν να χρησιμοποιηθούν ως flip flops ή μανδαλωτές (Latches). Τα LUT είναι δυνατόν να χρησιμοποιηθούν ως 16×1 μνήμη (RAM16) ή ως 16-bit καταχωρητής ολίσθησης (SRL16, Shift Register), αλλά μόνο στα SLICEM και όχι στα SLICEL. Επιπλέον οι πολυπλέκτες και η λογική κρατουμένου (*Carry Logic*), απλοποιούν τις αριθμητικές λειτουργίες. Η λογική διαμόρφωση αντιστοιχίζεται αυτόματα στα περισσότερα από τα γενικής χρήσης σχεδία λογικής. Στην Εικόνα Γ.3, φαίνεται πως μπορεί να διαμορφωθούν τα SLICEM και SLICEL και παράλληλα πώς μπορούν αυτά να χρησιμοποιηθούν σε ένα CLB.



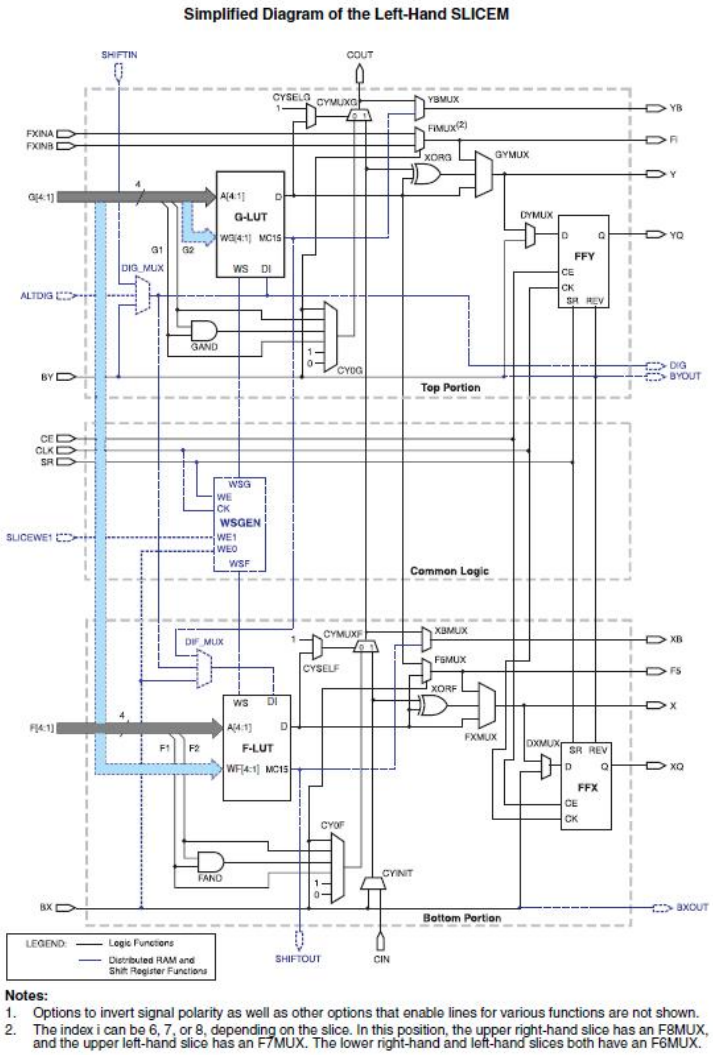
Εικόνα Γ.3: [36] Δομή ενός SLICEM και ενός SLICEL μέσα σε ένα CLB, της οικογένειας FPGA Spartan-3

Στην Εικόνα Γ.4, φαίνεται σε μεγαλύτερη λεπτομέρεια η δομή ενός SLICEM, η δομή ενός SLICEL είναι πανομοιότυπη απλά δεν χρησιμεύουν τα LUT ως μνήμη ή ως καταχωρητής ολίσθησης. Παρέχονται επίσης, γρήγορη αριθμητική λογική, λογική πολλαπλασιαστή, λογική πολυπλέκτη, set ή reset, κανονικές ή αντεστραμμένες είσοδοι. Η μνήμη RAM που μπορεί να υλοποιηθεί από ένα LUT σε ένα SLICEM, μπορεί να είναι μονής ή διπλής θύρας (single and dual port RAM) και συνδέοντας πολλά LUTs αυξάνεται το μέγεθός της. Η εγγραφή είναι μόνο σύγχρονη και η ανάγνωση μπορεί να είναι σύγχρονη ή ασύγχρονη. Υπάρχει η δυνατότητα να δημιουργηθεί ένας μεγάλος καταχωρητής ολίσθησης συνδέοντας πολλά LUTs. Το κάθε CLB περιέχει έως και 64 bits μίας θύρας μνήμη RAM ή 32 bits της διπλής θύρας μνήμη RAM. Αυτές η μνήμες RAM είναι κατανεμημένα σε όλη την αρχιτεκτονική της συσκευής FPGA και ονομάζεται κοινώς

"*κατανεμημένη μνήμη RAM*", ώστε να διαχωρίζεται από το 18Kbit μνήμης RAM (RAM Block). Η κατανεμημένη μνήμη RAM είναι επίσης γνωστή και ως *LUT RAM*.

Γ.3 Λογικό Στοιχείο

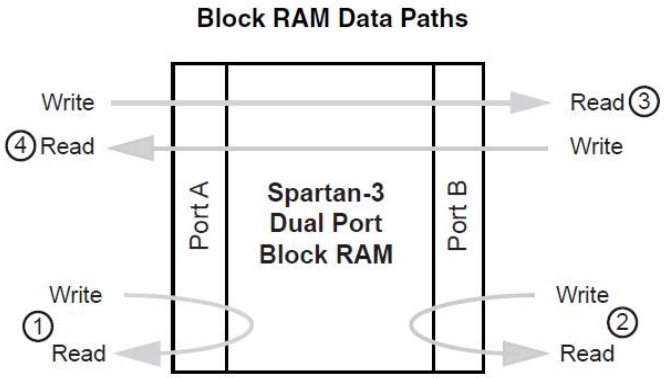
Το λογικό στοιχειό στη Xilinx αποτελεί το λογικό κύτταρο "Logic Cell", το οποίο δεν είναι άλλο από τη λογική που περιβάλει ένα LUT. Ο συνδυασμός ενός LUT και του αποθηκευτικού στοιχείο είναι γνωστό ως «Logic Cell". Τα πρόσθετα χαρακτηριστικά σε ένα slice, όπως είναι ο πολυπλέκτης, η λογική κρατουμένου, αλλά και οι αριθμητικές πύλες προσθέτουν και αυξάνουν τη χωρητικότητα ενός slice. Αν δεν υπήρχαν αυτές οι λογικές μονάδες θα χρειαζόταν επιπλέον πρόσθεση ενός ακόμη LUT.



Εικόνα Γ.4: [35] Λεπτομερέστερη δομή ενός SLICEM, της οικογένειας FPGA Spartan-3

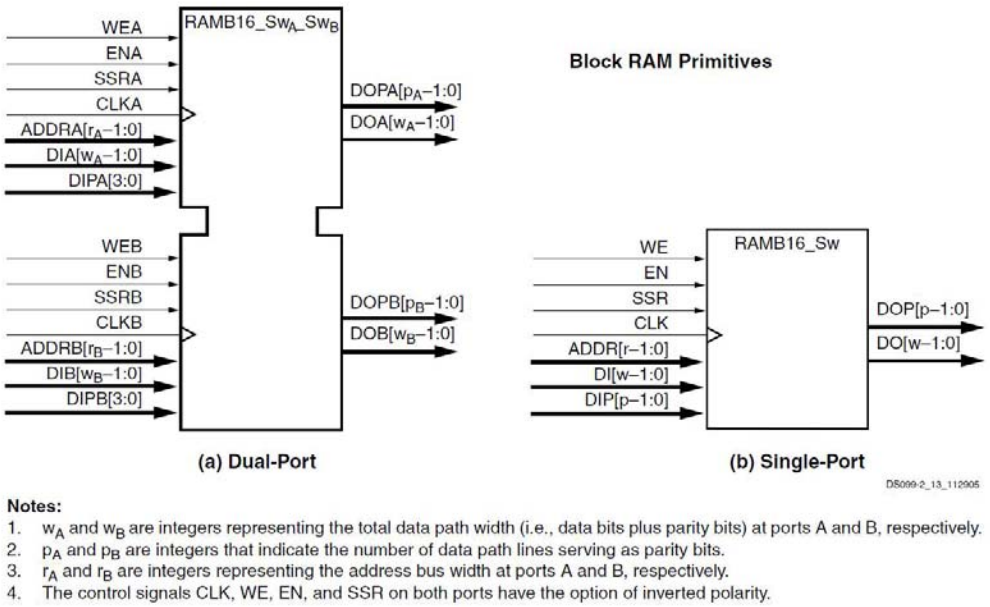
Γ.4 Μνήμη RAM

Οι ενσωματωμένες βαθμίδες μνήμης (RAM Block), οργανώνονται σε μία διαμορφωμένη μνήμη των 18Kbit. Υπάρχουν ειδικές περιπτώσεις όπου κάποιες μνήμες, ανάλογα με τη λειτουργία που εκτελούν, διαθέτουν τα 16K για τα δεδομένα και τα 2K για την ισοτιμία. Η μνήμη RAM αποθηκεύει σχετικά μεγαλύτερες ποσότητες δεδομένων πιο αποτελεσματικά από την κατανεμημένη μνήμη RAM που περιγράψαμε παραπάνω. Είναι ιδανική για υλοποίηση διαφορετικών τύπων μνημών όπως διπλής και μονής θύρας. Στην Εικόνα Γ.5, φαίνεται η χρήση της μνήμης RAM ως διπλής θύρας (Dual Port) προσπέλασης.



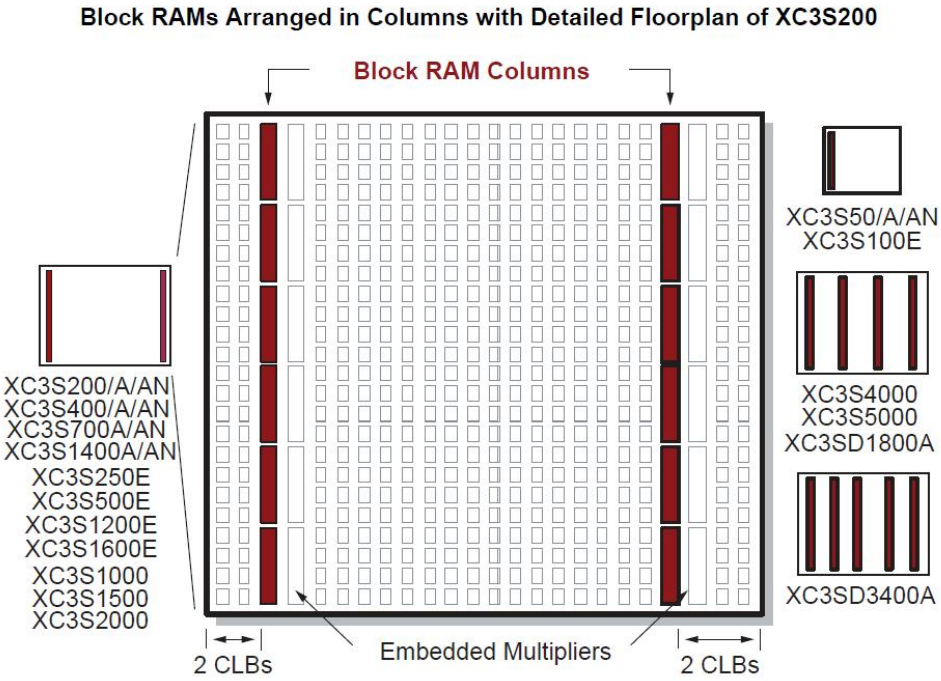
Εικόνα Γ.5: [35] Μνήμη RAM με χρήση διπλής θύρας (Dual Port), της οικογένειας FPGA Spartan-3

Στην Εικόνα Γ.6, φαίνεται η λεπτομερέστερη απεικόνιση των σημάτων ελέγχου και διασύνδεσης με χρήση λειτουργίας διπλής θύρας και μονής θύρας. Όπως μπορεί να παρατηρηθεί υπάρχει ανεξάρτητη πρόσβαση και διευθέτηση των θυρών. Μια διπλής θύρας RAM μπορεί να διαχωριστεί και να λειτουργεί ως δυο ανεξάρτητες μνήμες μονής θύρας, το μειονέκτημα με αυτή την επιλογή είναι ότι η μνήμη πέφτει στο μισό της μέγεθος. Η κάθε θύρα συγχρονίζεται με το δικό της ανεξάρτητο ρολόι. Επιπλέον αυτές οι μνήμες μπορούν αν αρχικοποιηθούν και να χρησιμοποιηθούν ως ROM.



Εικόνα Γ.6: [35] Τα σήματα ελέγχου και διασύνδεσης της μνήμης RAM με χρήση διπλής και μονής θύρας (Dual - Single Port), της οικογένειας FPGA Spartan-3

Στην Εικόνα Γ.7, φαίνεται η δομή που έχει η μνήμη RAM σε κάθε μία από τις συσκευές της οικογένειας συσκευών FPGA Spartan-3, αλλά και των επεκτάσεών της Spartan-3A και Spartan-3E.



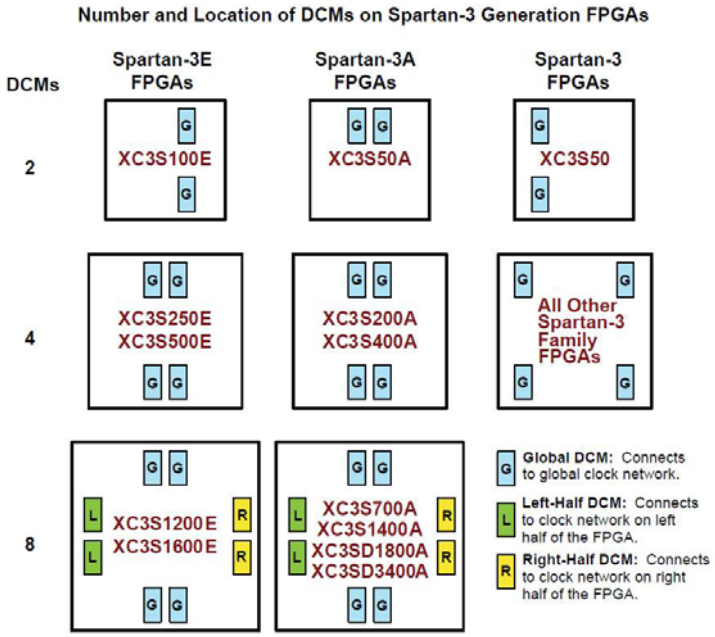
Εικόνα Γ.7: [36] Δομή μνήμης RAM για κάθε μία από τις συσκευές της οικογένειας FPGA Spartan-3 και των επεκτάσεων της, Spartan-3A και Spartan-3E

Από όλα τα παραπάνω χαρακτηριστικά που έχουν αναφερθεί φαίνεται ότι χρησιμοποιώντας διάφορες ρυθμίσεις και επιλογές, μπορεί κανείς στις επιλεγόμενες μνήμες RAM να δημιουργήσει RAM, ROM, FIFOs, μεγάλους πίνακες αναζήτησης LUT, καταχωρητές ολίσθησης και μετατρέποντας το πλάτος των δεδομένων αποθήκευσης να υποστηρίξουμε μία ποικιλία εισαγωγής δεδομένων ανεξάρτητα του μεγέθους τους.

Γ.5 Ρολόι

Κάθε Spartan-3 FPGA συσκευή παρέχει οκτώ υψηλής ταχύτητας πηγές παραγωγής σημάτων ρολογιού, με χαμηλή απόκλιση (low skew) για να βελτιστοποιήσουν την απόδοση. Οι πηγές αυτές χρησιμοποιούνται αυτόματα από τα εργαλεία Xilinx. Ακόμη και αν ο ρυθμός του ρολογιού είναι σχετικά αργός, εξακολουθεί να είναι σημαντικό η χρησιμοποίηση των πηγών αυτών για τη σωστή δρομολόγηση. Με αυτό το τρόπο πετυχαίνεται να εξαλειφθεί ο κάθε πιθανός κινδύνους που μπορεί να υπάρξει σε μία συσκευή όπως η έλλειψη σήματος ρολογιού. Είναι πολύ σημαντικό να καταλάβει κανείς πώς πρέπει να καθορίσει ή καλύτερα πως να επωφεληθεί από αυτούς τους πόρους.

Κάθε συσκευή Spartan-3 έχει περισσότερα από δυο διευθυντές ρολογιού (DCM). Μία λεπτομερή αναπαράσταση της δομής των διευθυντών ρολογιού σε κάθε συσκευή της οικογένειας Spartan-3 και των επεκτάσεων της Spartan-3A και Spartan-3E, φαίνεται στην Εικόνα Γ.8.



Εικόνα Γ.8: [36] Δομή των διευθυντών ρολογιού (DCM) για κάθε μία από τις συσκευές της οικογένειας FPGA Spartan-3 και των επεκτάσεων της, Spartan-3A και Spartan-3E

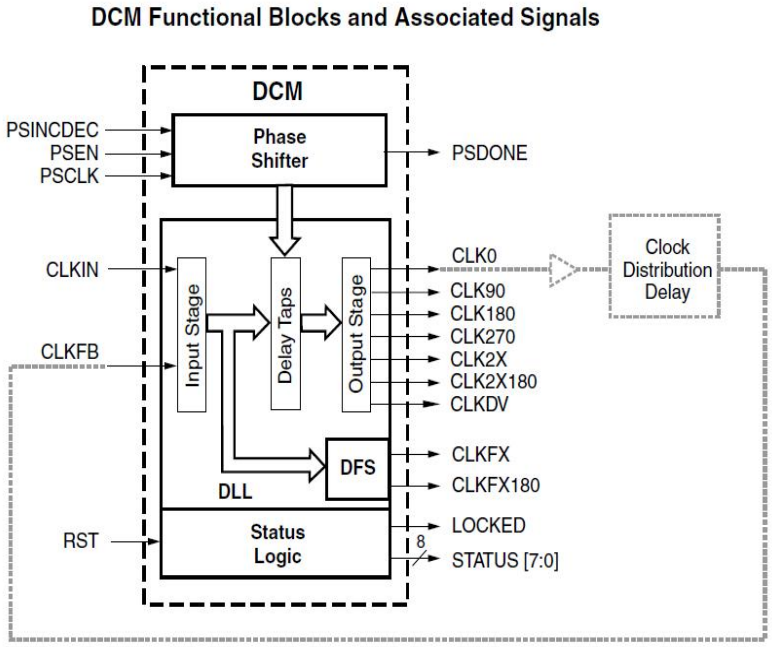
Ο κάθε διευθυντής ρολογιού παρέχει, περιορισμό της απόκλισης ρολογιού (clock skew), ολίσθηση φάσης (DPS, Digital Phase Shift), σύνθεση συχνοτήτων (DFS, Digital Frequency Synthesis), η δομή ενός DCM φαίνεται στην Εικόνα Γ.9. Για να επιτευχθούν τα παραπάνω ο DCM χρησιμοποιεί ένα *βρόχο σταθερής καθυστέρησης* (*DLL, Delay Locked Loop*) το οποίο κλειδώνει στην καθυστέρηση. Είναι ένα πλήρες ψηφιακό σύστημα ελέγχου που χρησιμοποιεί την ανατροφοδότηση του κύριου ρολογιού για να διατηρήσει τα χαρακτηριστικά του σήματος ρολογιού σε υψηλό βαθμό ακρίβειας (Εικόνα Γ.10). Κάποια κύρια χαρακτηριστικά του DLL είναι:

- Εισάγει καθυστέρηση στο δίκτυο του ρολογιού έως ότου η ανοδική ακμή του ρολογιού εισόδου (*CLKIN*) είναι σε φάση με την ανοδική ακμή του ρολογιού ανάδρασης (*CLKFB*). Δηλαδή το εξωτερικό ρολόι και το εσωτερικό να είναι «ευθυγραμμισμένα» (Εικόνα Γ.9).

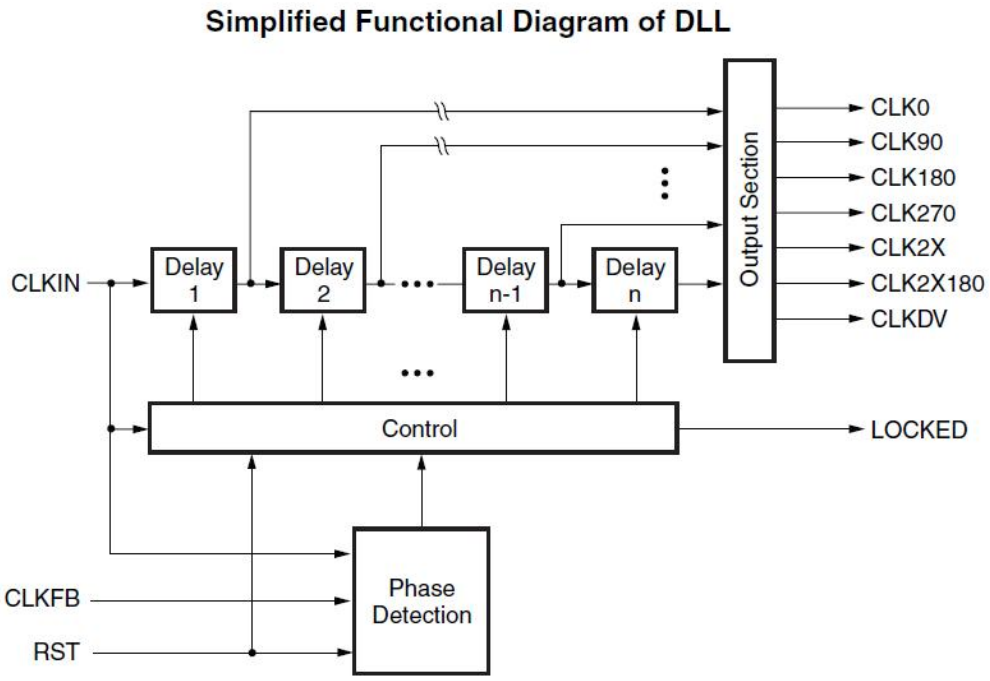
- Με ένα καλοσχεδιασμένο δίκτυο διανομής του ρολογιού, οι ακμές του ρολογιού «φτάνουν» ταυτόχρονα σε όλα τα σημεία της συσκευής μαζί με την άφιξη της ακμής του ρολογιού εισόδου του DLL (Εικόνα Γ.11).

- Χρήση ρολογιών με διαφορετικές φάσεις 0^o, 90^o, 180^o και 270^o ώστε να αυξηθεί η απόδοση, είναι κατάλληλο για διασύνδεση εξωτερικών μνημών.

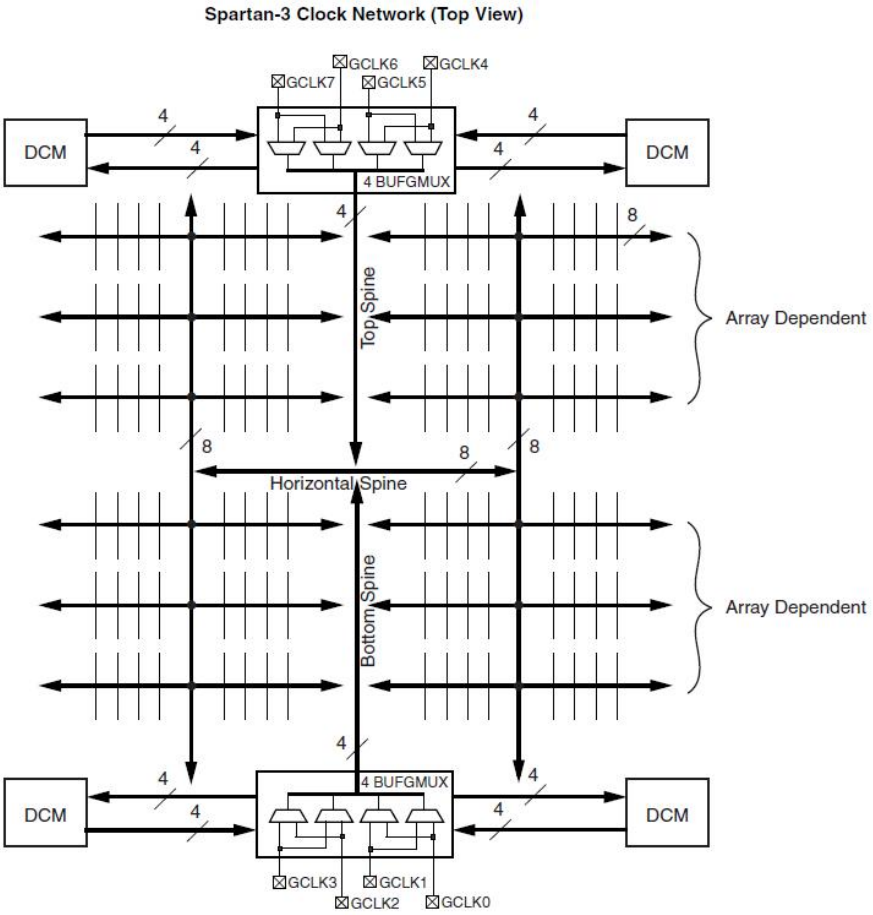
- Πολλαπλασιασμό συχνότητας $\times 2$, διαίρεση συχνότητας από 1.5 έως 16.



Εικόνα Γ.9: [35] Δομή ενός διευθυντή ρολογιού (DCM), της οικογένειας FPGA Spartan-3



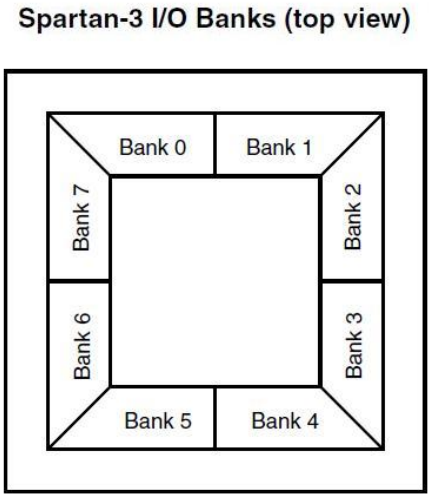
Εικόνα Γ.10: [35] Δομή ενός βρόχου σταθερής καθυστέρησης (DLL, Delay Locked Loop), της οικογένειας FPGA Spartan-3



Εικόνα Γ.11: [35] Δομή δικτύου διανομής του ρολογιού της οικογένειας FPGA Spartan-3

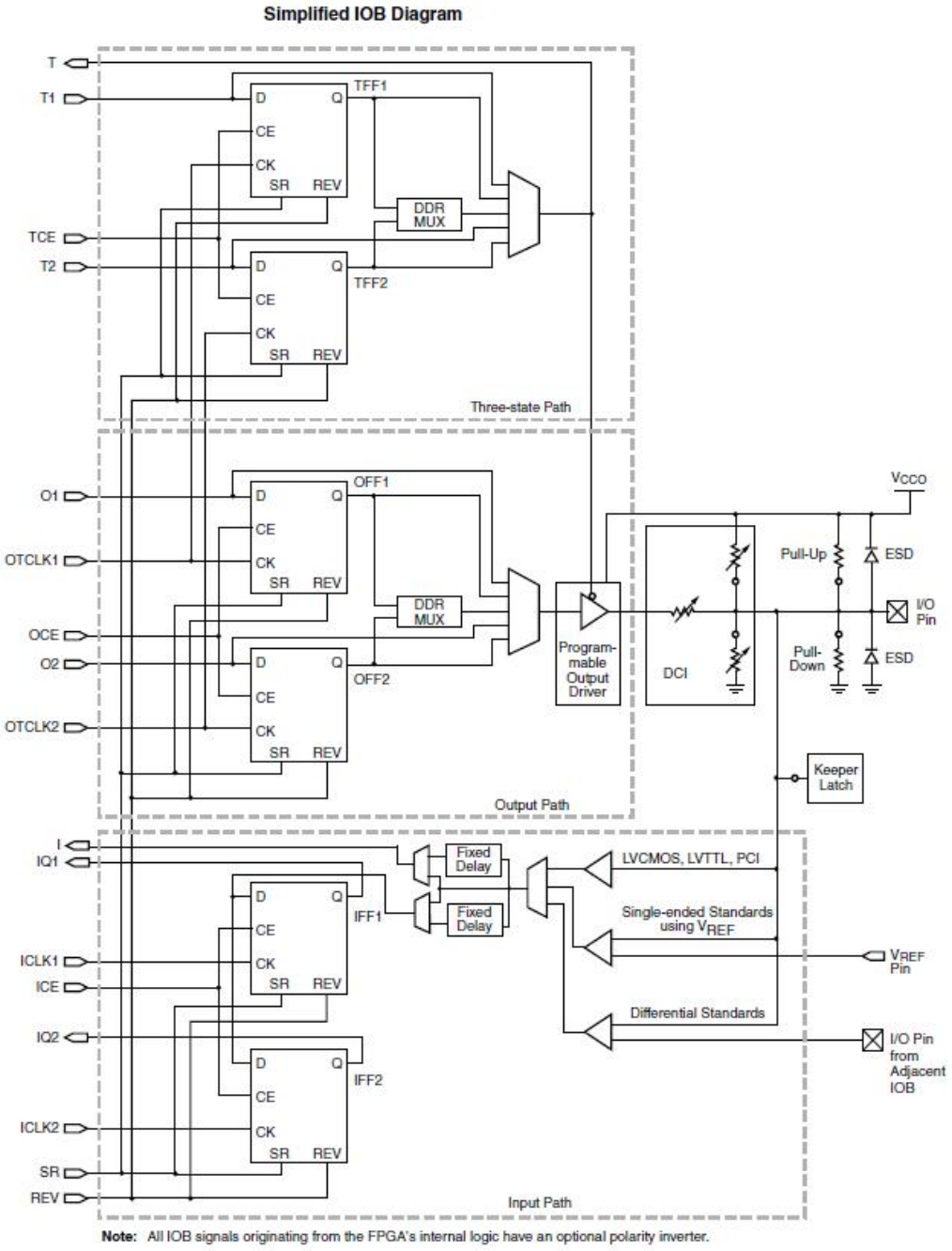
Γ.6 Ακροδέκτες Εισόδου Εξόδου (I/O Bank)

Οι ακροδέκτες εισόδου εξόδου (I/O) στην αρχιτεκτονική της οικογένειας Spartan-3 οργανώνονται σε οκτώ βαθμίδες (I/O Bank), Εικόνα Γ.12. Κάθε βαθμίδα περιέχει ένα σύνολο από ακροδέκτες (IOB, I/O Block) το οποίο εξαρτάται από τη δομή της κάθε συσκευής.



Εικόνα Γ.12: [35] Δομή των βαθμίδων (I/O Bank) των ακροδεκτών στην αρχιτεκτονική της οικογένειας FPGA Spartan-3

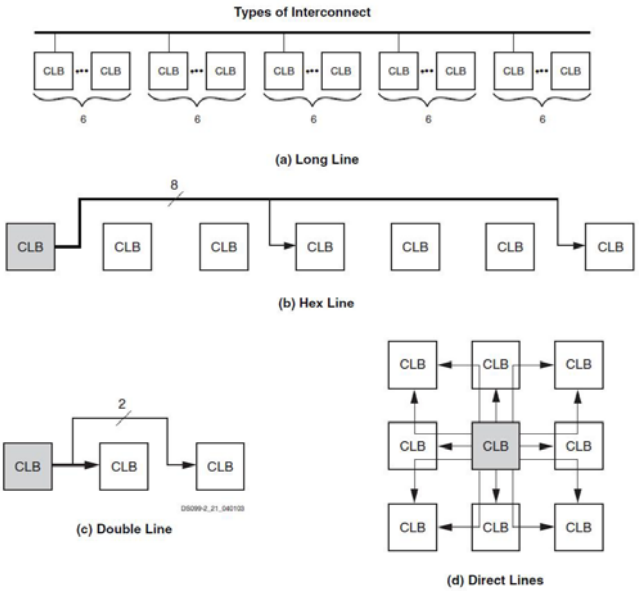
Το σύνολο ακροδεκτών IOB παρέχει μια προγραμματιζόμενη μονής ή διπλής κατεύθυνσης διεπαφή μεταξύ ενός πακέτου ακροδεκτών και της εσωτερικής λογικής του FPGA, υποστηρίζοντας μια μεγάλη ποικιλία τυποποιημένων διεπαφών. Το σύνολο των χαρακτηριστικών περιλαμβάνει προγραμματιζόμενο έλεγχο της εξόδου και της εισόδου, καθυστέρηση της εισόδου, αποθήκευση των αποτελεσμάτων στην έξοδο ή στην είσοδο, με τους ειδικούς καταχωρητές διπλού ρυθμού (DDR, Double Data Rate). Η δομή ενός IOB φαίνεται στην Εικόνα Γ.13. Προαναφέρθηκαν κάποια χαρακτηριστικά όπως, μονοί και διαφορικοί ακροδέκτες, πχ όταν έχει 784 μονούς ακροδέκτες αυτοί μπορούν να γίνουν 344 διαφορικά ζεύγη, 26 πρότυπα εισόδου-εξόδου (I/O) όπου περισσότερα πρότυπα επιτρέπουν περισσότερες δυνατότητες ολοκλήρωσης στο σύστημα [40]. Επιπλέον, σύνδεση Chip με Chip LVDS, LVCMOS, LVTTL, διασύνδεση με πλακέτα βάσης (balkplane) GTL, GTL+, PCI, BLVDS, διασύνδεση με μνήμη υψηλής ταχύτητας HSTL, SSTL, υποστήριξη PCI 32/33 και 64/33.



Εικόνα Γ.13: [35] Δομή ενός συνόλου ακροδεκτών (I/O Block) στην αρχιτεκτονική της οικογένειας FPGA Spartan-3

Γ.7 Διασύνδεση

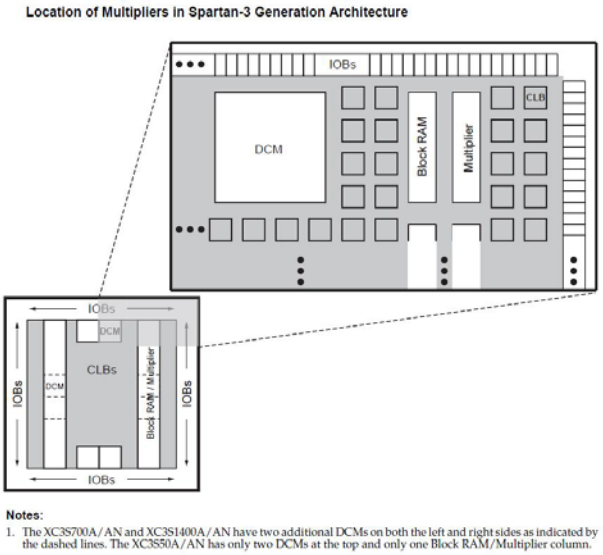
Ο τρόπος διασύνδεσης των CLB μεταξύ τους φαίνεται στην Εικόνα Γ.14, όπου στο a) *Long Line* υπάρχει διασύνδεση κάθε CLB ανά έξι, στο b) *Xex Line* υπάρχει διασύνδεση κάθε CLB ανά 3, στο c) *Double Line* υπάρχει διασύνδεση κάθε CLB ανά 2 και τέλος στο d) *Direct Line* υπάρχει άμεση διασύνδεση κάθε CLB με τα γειτονικά του.



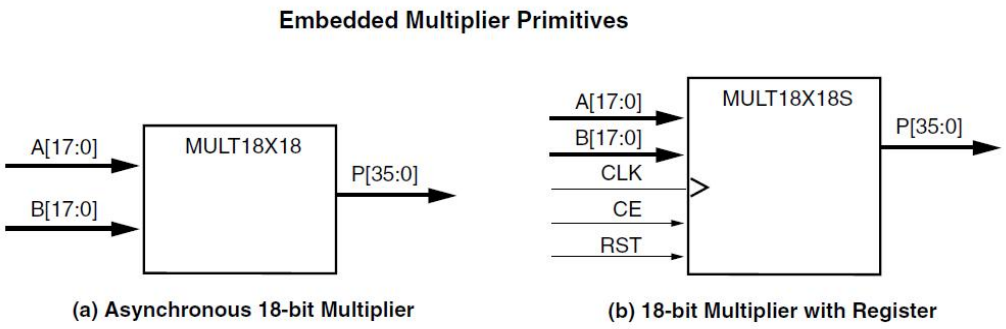
Εικόνα Γ.14: [35] Δομή της διασύνδεσης των CLB στην αρχιτεκτονική της οικογένειας FPGA Spartan-3

Γ.8 Πολλαπλασιαστές

Μία επιπλέον λογική μονάδα που έχει η αρχιτεκτονική δομή της οικογένειας Spartan-3 είναι οι *Πολλαπλασιαστές* (*Multipliers*). Ο κάθε πολλαπλασιαστής βρίσκεται δίπλα σε μια βαθμίδα μνήμης (RAM Block) έτσι ώστε να διευκολύνει τον υπολογισμό των αριθμητικών πράξεων και των δεδομένων, Εικόνα Γ.15. Αυτοί οι πολλαπλασιαστές δέχονται δύο 18bit λέξεις ως είσοδο και παράγουν ως έξοδο μια λέξη των 36bit. Υπάρχουν δύο εκδόσεις, μία με την ονομασία MULT18X18 η οποία είναι ασύγχρονη και μια έκδοση με καταχωρητή, με την ονομασία MULT18X18S, Εικόνα Γ.16 a) και b).



Εικόνα Γ.15: [36] Δομή πολλαπλασιαστών (Multipliers) στην αρχιτεκτονική της οικογένειας FPGA Spartan-3



Εικόνα Γ.16: [35] Οι δύο εκδόσεις των πολλαπλασιαστών (Multipliers) της οικογένειας FPGA Spartan-3

Για περισσότερη λεπτομέρεια για όσα έχουν αναφερθεί παραπάνω σε σχέση με την αρχιτεκτονική δομή της οικογένειας FPGA Spartan-3 της εταιρίας Xilinx, ο αναγνώστης θα παραπεμφθεί στους οδηγούς χρήσης «*Spartan-3 FPGA Family Data Sheet*» [35] και «*Spartan-3 Generation FPGA User Guide*» [36].

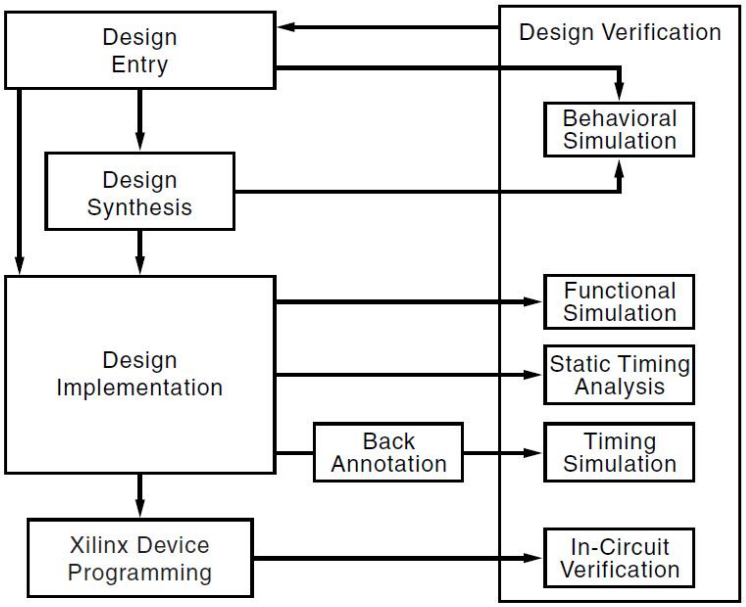
Παράρτημα Δ

Σχεδίαση με Χρήση Λογισμικού Σχεδίασης CAD, Xilinx ISE 10.1

Σε αυτήν την ενότητα θα παρουσιαστεί η διαδικασία σχεδίασης με το εργαλείο ISE 10.1 [32,33] της εταιρίας Xilinx [37]. Για την ανάγκη της παρουσίασης ας επιτραπεί από τον αναγνώστη η χρήση του πρώτου και δεύτερου πληθυντικού.

Δ.1 Εισαγωγή στο ISE

Το εργαλείο σχεδίασης που θα παρουσιαστεί είναι το ISE και συγκεκριμένα η έκδοση 10.1. Η γενική μεθοδολογία σχεδίασης που προτείνεται στο ISE είναι αυτό της Εικόνας Δ.1. Όπως μπορείτε να δείτε έχει ομαδοποιηθεί η υλοποίηση (*Design Implementation*) η οποία περιλαμβάνει τα στάδια της λειτουργικής προσομοίωσης (*Functional Simulation*), της στατικής και χρονικής ανάλυσης (*Static Timing Analysis*) αλλά και αυτή της χρονικής προσομοίωσης (*Timing Simulation*). Να σημειωθεί εδώ ότι η Εικόνα που βλέπετε μπορεί να διαφέρει αρκετά με αυτό της Εικόνας 5.1, αλλά η ακολουθία των βημάτων είναι η ίδια με αυτή που περιγράψαμε στην Ενότητα 5.1.1. Άλλωστε θα το διαπιστώσετε και μόνοι σας από αυτά που θα ακολουθήσουν.



Εικόνα Δ.1: [33] Γενική μεθοδολογία σχεδίασης στο πρόγραμμα CAD ISE

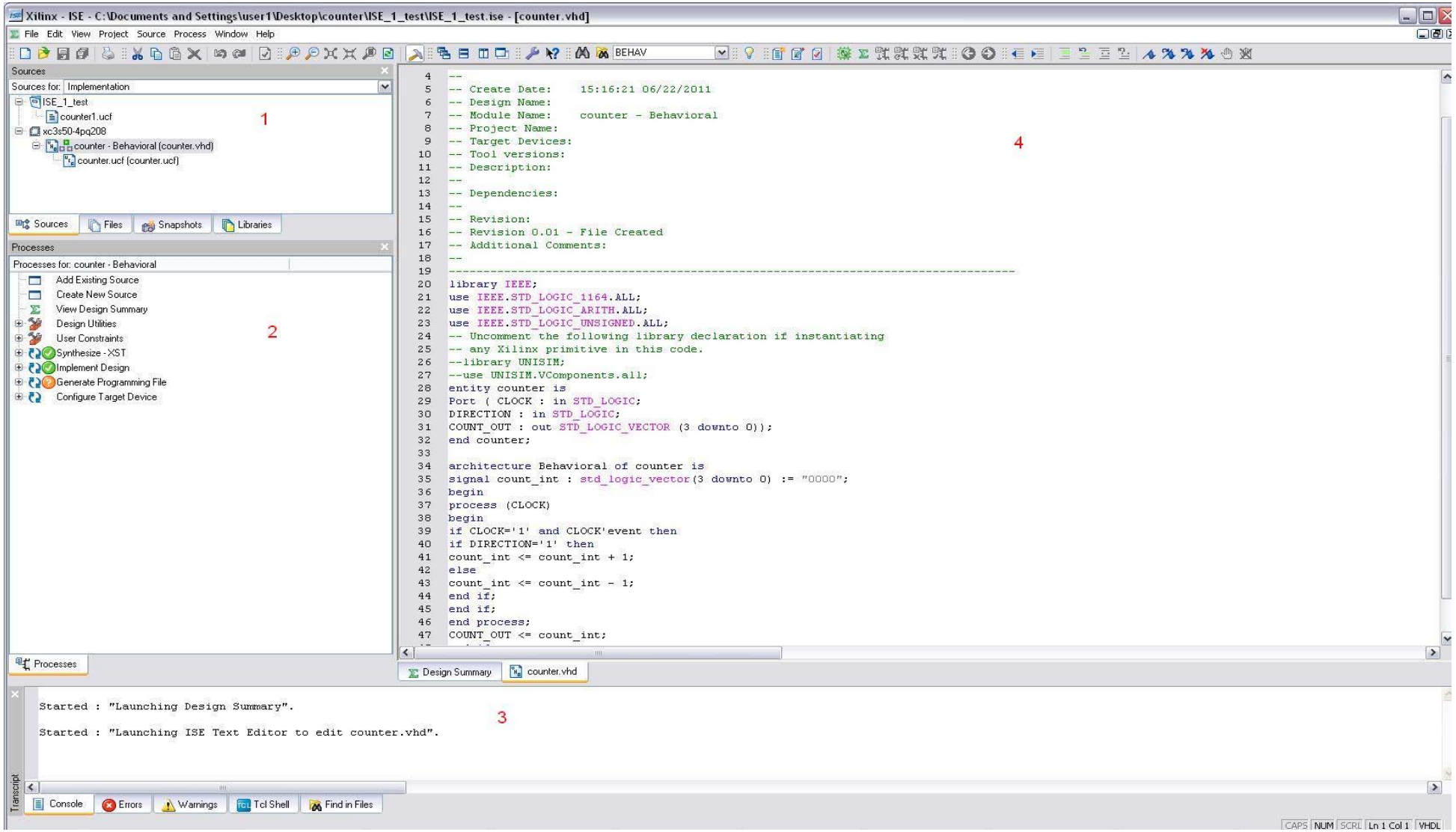
Το περιβάλλον εργασίας του ISE είναι το Project Navigator το οποίο φαίνεται στην Εικόνα Δ.2 το οποίο αποτελείται από τέσσερα υποπαράθυρα τα οποία είναι τα εξής :

- Το παράθυρο *Source in Project*, το οποίο αποτελείται από τρεις ετικέτες που παρέχουν πληροφορίες για το χρηστή σε σχέση με τα αρχεία που χρησιμοποιούνται στο κάθε project.

- Το παράθυρο *Processes for Current Source*, το οποίο έχει την ετικέτα *Process View* η οποίο περιέχει τις διαδικασίες στις οποίες μπορεί να προβεί για εκτέλεση ο χρήστης και αλλάζει το περιεχόμενο της ανάλογα το αρχείο που εκτελείται εκείνη τη στιγμή.

- Το παράθυρο *Console*, το οποίο μας δείχνει τα τυχόν λάθη ή μηνύματα που θα προκύψουν κατά τη διάρκεια της σχεδίασης.

- Και τέλος το τέταρτο παράθυρο είναι αυτό στη μέση το οποίο χρησιμοποιείται για ανάγνωση και εγγραφή σε ποικιλία αρχείων που βρίσκονται μέσα στο project όπως αναφορές, αρχεία VHDL, αρχεία προσομοίωσης συμπεριφοράς κτλ.

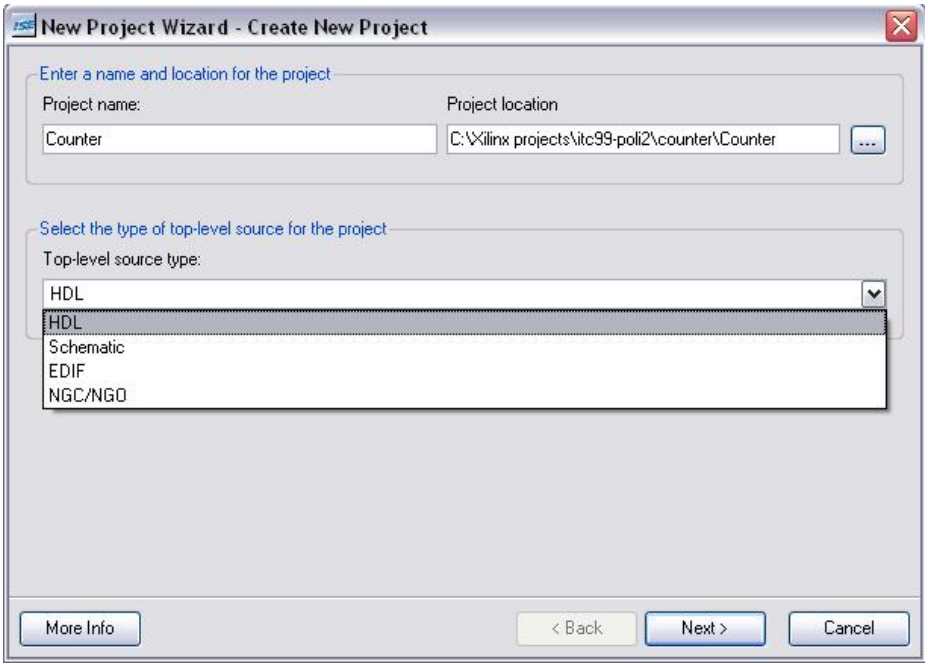


Εικόνα Δ.2: Ένα τυπικό παράθυρο σχεδίασης Project Navigator του προγράμματος CAD ISE

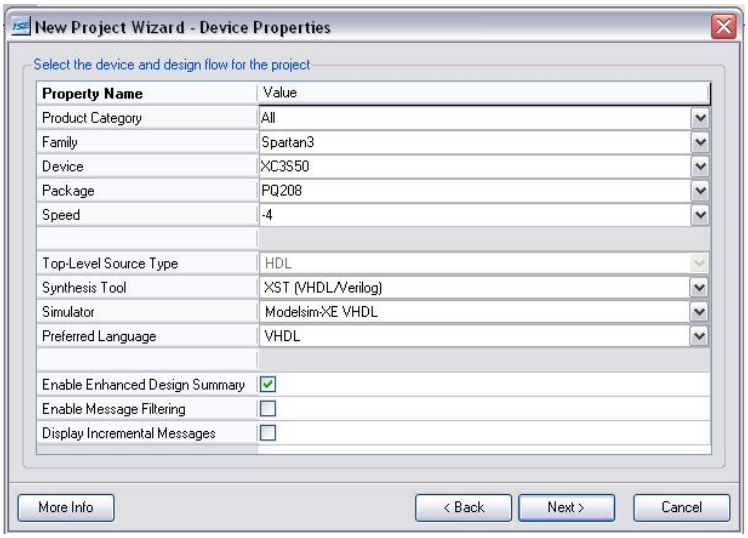
Δ.2 Δημιουργία Νέου Project

Όπως και στην προηγούμενη ενότητα έτσι και τώρα θα δημιουργήσουμε ένα αρχείο VHDL το οποίο και θα εκτελέσουμε. Σε αυτή την ενότητα θα ακολουθήσουμε το προτεινόμενο πρόγραμμα που βρίσκεται στον οδηγό χρήσης του εργαλείου ISE, το οποίο είναι ένας μετρητής *Counter* (Ενότητα Ζ.3 Παράρτημα Ζ).

Το πρώτο που θα κάνουμε είναι να δημιουργήσουμε ένα νέο project, πάμε *File→New Project*. Στο παράθυρο διαλόγου που εμφανίζεται επιλέγουμε το όνομα του νέου project έστω *Counter*, το κατάλογο προορισμού αλλά και τη σχεδίαση θέλουμε να κάνουμε (Εικόνα Δ.3). Κατόπιν πατάμε next, στο νέο παράθυρο διαλόγου επιλέγουμε τη συσκευή της επιθυμίας μας, έστω *Spartan–3 XC3S50* αλλά και τα υπόλοιπα χαρακτηριστικά αν θέλουμε (Εικόνα Δ.4). Αν σε περίπτωση η συσκευή που έχουμε επιλέξει αποδειχθεί λιγότερο ικανή (λόγω χαρακτηριστικών) για το project που θα έχουμε δημιουργήσει, τότε μπορούμε να την αλλάξουμε από τις ιδιότητες του project.

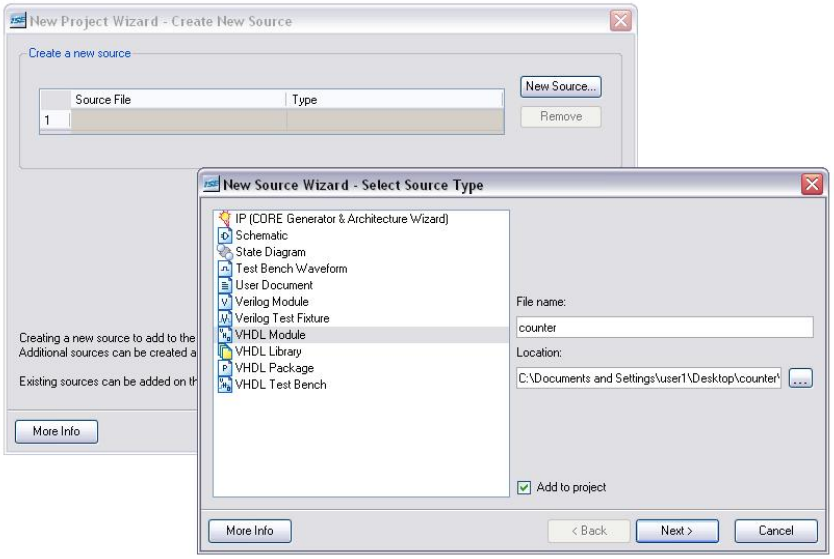


Εικόνα Δ.3: Παράθυρο διαλόγου για τη δημιουργία νέου project, από το πρόγραμμα CAD ISE

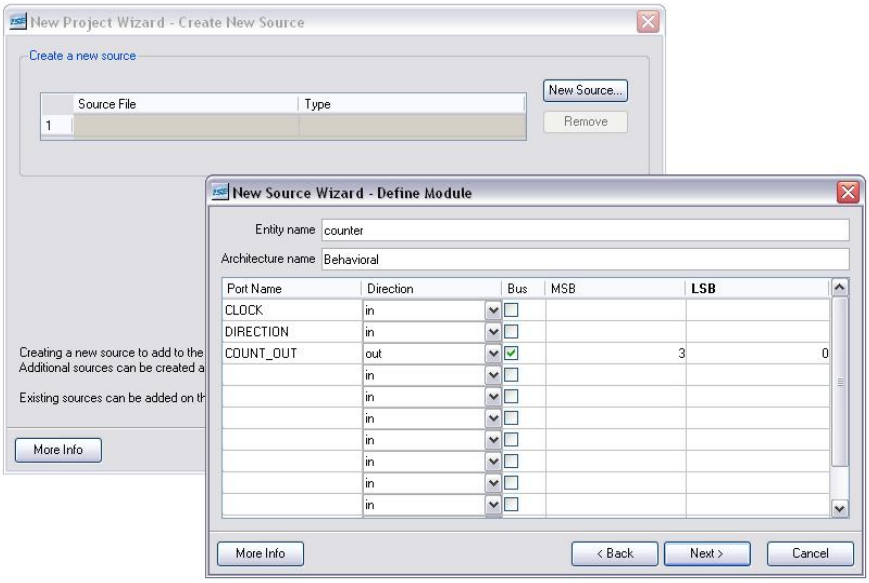


Εικόνα Δ.4: Παράθυρο διαλόγου για την επιλογή συσκευής Spartan-3 XC3S50 στο νέο project, από το πρόγραμμα CAD ISE

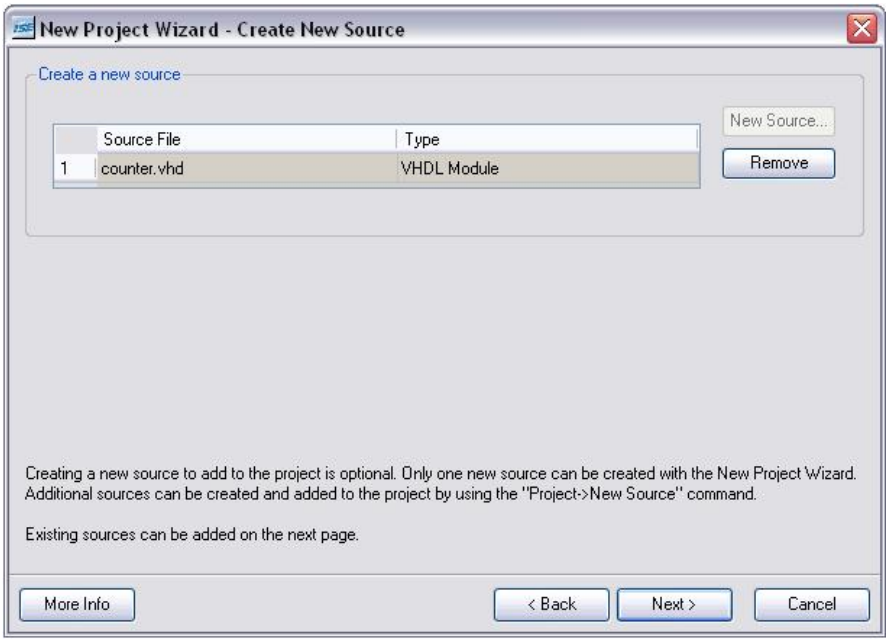
Στη συνέχεια αφού επιλέξουμε τη συσκευή πατάμε Next. Στο νέο παράθυρο διαλόγου θα πρέπει να επιλέξου ένα αρχείο στο οποίο θα είναι η πηγή της εργασίας που θα κάνουμε. Επιλέγουμε *New Source*, από το νέο παράθυρο που θα εμφανιστεί επιλέγουμε το αρχείο στο οποίο θα γράψουμε το κώδικα μας. Επιλέγουμε *VHDL Module* και βάζουμε όνομα αρχείου Counter και μετά Next, Εικόνα Δ.5. Έχουμε ένα νέο παράθυρο στο οποίο θα πρέπει να βάλουμε τις μεταβλητές εισόδου αλλά και εξόδου για τη νέα οντότητα (*Entity*) που δημιουργούμε, γράφουμε και επιλέγουμε ότι φαίνεται στην Εικόνα Δ.6. Αφού γίνει και αυτό πατάμε Next και Finish, αφού επανέλθουμε στο αρχικό παράθυρο θα πρέπει να έχει περιλάβει το νέο αρχείο που δημιουργήσαμε (Εικόνα Δ.7). Μετά από όλα αυτά με δυο Next και Finish έχουμε δημιουργήσει το νέο project.



Εικόνα Δ.5: Παράθυρο διαλόγου για την επιλογή VHDL Module στο νέο project, από το πρόγραμμα CAD ISE



Εικόνα Δ.6: Παράθυρο διαλόγου για τον ορισμό της νέας οντότητας (Entity) που δημιουργούμε στο project, από το πρόγραμμα CAD ISE

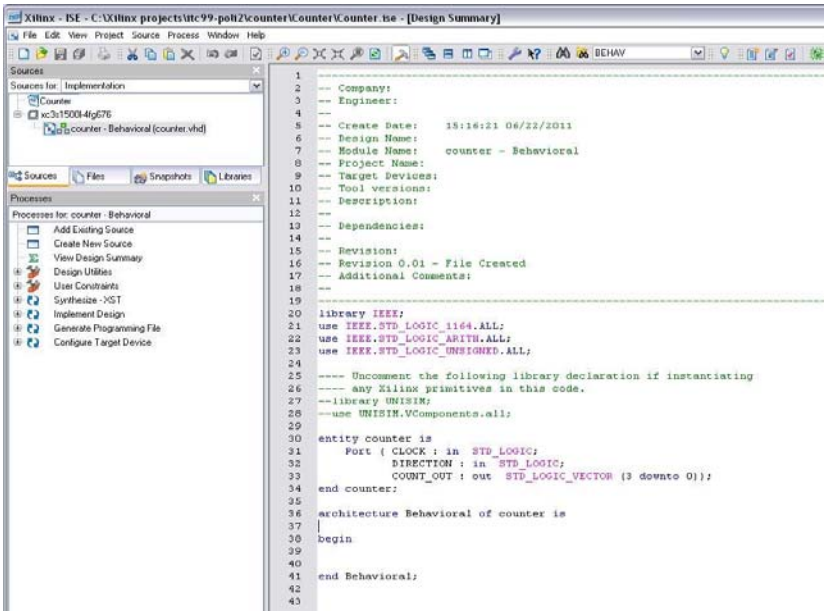


Εικόνα Δ.7: Παράθυρο διαλόγου που περιλαμβάνει τη νέα οντότητα (Entity) στο νέο project, από το πρόγραμμα CAD ISE

Δ.3 Σχεδίαση με Γλώσσα VHDL

Αν όλα πάνε καλά θα πρέπει στο εργαλείο ISE να υπάρχει το νέο project μαζί με το αρχείο VHDL όπως φαίνεται στην Εικόνα Δ.8. Αν τώρα θα θέλαμε να εμπλουτίσουμε το αρχείο μας με νέες και έτοιμες μονάδες λογικής όπως είναι ένας απλός μετρητής, αρκεί να πάμε *Edit→Language Templates→VHDL→ Synthesis Constructs→Coding Examples→Counters→Binary→Up / Down Counters→Simple Counter*. Με αυτό το τρόπο μπορούμε να εισάγουμε έτοιμες μονάδες λογικής

στη σχεδίαση μας. Στις βιβλιοθήκες αυτές υπάρχουν διαφόρων ειδών λογικές μονάδες χωρίς εμείς να χρειαστεί να δημιουργήσουμε τα περισσότερα από αυτά.



Εικόνα Δ.8: Το νέο project μαζί με το αρχείο VHDL, από το πρόγραμμα CAD ISE

Στη συνέχεια, θα πρέπει να διαμορφώσουμε τη σχεδίαση μας σύμφωνα με το κώδικα που φαίνεται στην Εικόνα Δ.9, αν το κάνουμε αυτό αποθηκεύουμε. Το κομμάτι αυτού του κώδικα βρίσκεται στο παράρτημα Β με το κωδικό όνομα *counter.vhd*.

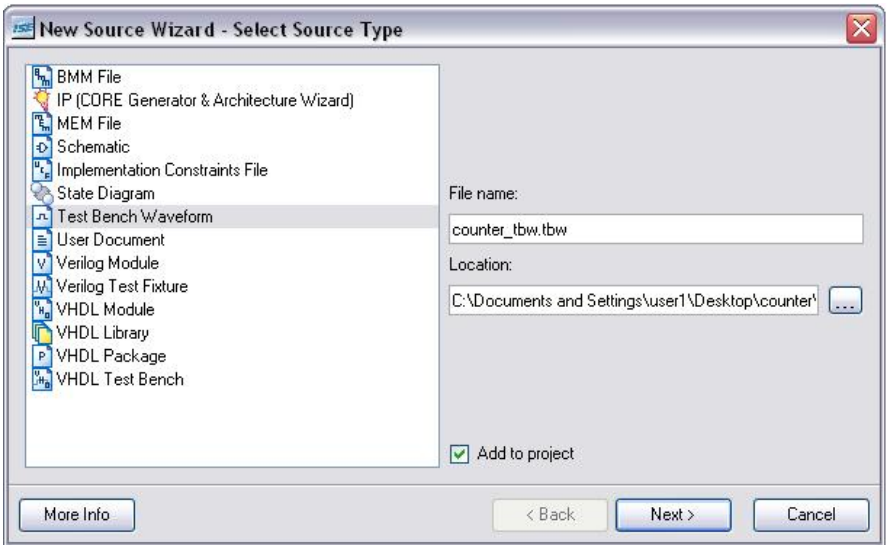
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitive in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter is
  Port ( CLOCK : in  STD_LOGIC;
    DIRECTION : in  STD_LOGIC;
    COUNT_OUT : out  STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
begin
  if CLOCK='1' and CLOCK'event then
    if DIRECTION='1' then
      count_int <= count_int + 1;
    else
      count_int <= count_int - 1;
    end if;
  end if;
end process;
COUNT_OUT <= count_int;
end Behavioral;
```

Εικόνα Δ.9: [32] Κώδικας VHDL για την υλοποίηση απλού μετρητή Counter

Δ.4 Λειτουργική Προσομοίωση

Για να μπορέσουμε να δούμε αν η σχεδίαση μας είναι σωστή πρέπει να κάνουμε σύνθεση και λειτουργική προσομοίωση, όπως είδαμε και στην Ενότητα 5.1.1. Πριν κάνουμε σύνθεση καλό είναι να δούμε τη λειτουργική προσομοίωση. Για να γίνει αυτό στο ISE πρέπει να δημιουργήσουμε έναν «*πάγκο εργασίας*» για την ακρίβεια να δημιουργήσουμε ένα αρχείο *Test Bench*. Υπάρχουν δύο ειδών αρχεία Test Bench που έχει το ISE, το πρώτο *Test Bench Waveform* με το οποίο παράγουμε κυματομορφές με χειροκίνητη εισαγωγή των τιμών εισόδου και το δεύτερο *VHDL Test Be*nch γράφοντας ένα αρχείο με τη γλώσσα VHDL. Στη δεύτερη περίπτωση ο χρήστης μπορεί να γράψει την κυματομορφή εισαγωγής που επιθυμεί με την τοποθέτηση μηδέν και ένα. Μέσω των παραγόμενων κυματομορφών μπορούμε να διαπιστώσουμε αν η σχεδίαση μας έχει τη σωστή συμπεριφορά. Εμείς θα δούμε την πρώτη εφαρμογή και μετά τη δεύτερη.

Για την πρώτη περίπτωση, το πετυχαίνουμε πηγαίνοντας *Project→New Source* και επιλέγοντας τη δημιουργία αρχείου *Test Bench WaveForm*, δηλαδή το αρχείο αυτό θα μας δείξει τις κυματομορφές που θα παράγει η σχεδίαση μας. Σώζουμε με το όνομα *counter_tbw.tbw* και πατάμε Next, Next και Finsh (Εικόνα Δ.10).

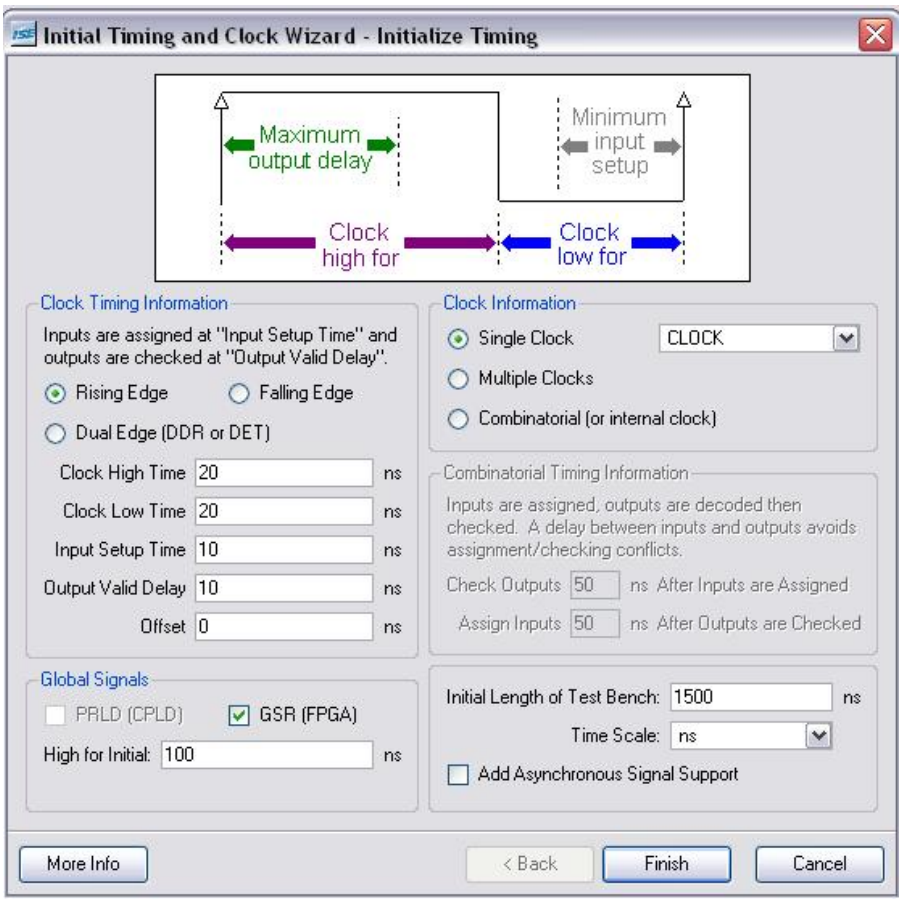


Εικόνα Δ.10: Επιλογή «πάγκου εργασίας»Test Bench από το πρόγραμμα CAD ISE

Κατόπιν θα εμφανίσει ένα παράθυρο το *Initial Timing and Clock Wizard* στο οποίο θα πρέπει να τροποποιήσουμε κάποιες παραμέτρους όπως χρόνος διάρκειας προσομοίωσης, χρόνους ρολογιού (High and Low) κτλ. Αυτές οι παράμετροι χρησιμεύουν για να θέσουμε μερικές προδιαγραφές στο σύστημα μας και μέσω αυτών θα δούμε αν η σχεδίαση μας λειτουργεί σωστά. Έτσι επιλέγουμε να αλλάξουμε:

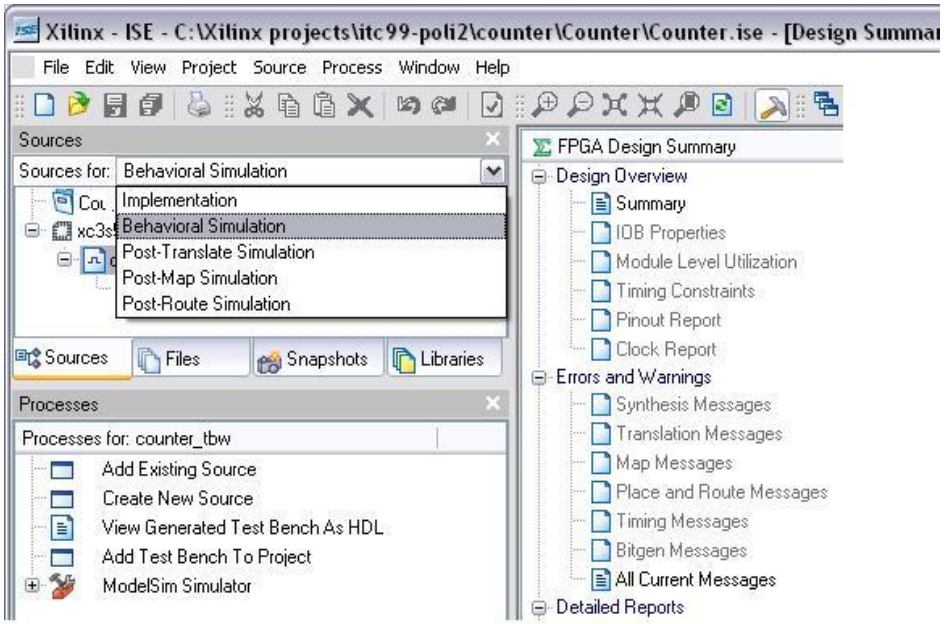
- Clock High Time: **20** ns
- Clock Low Time: **20** ns
- Input Setup Time: **10** ns
- Output Valid Delay: **10** ns
- Offset: **0** ns
- Global Signals: **GSR (FPGA)**
- High for initial: **100** ns
- Initial Length of Test Bench: **1500** ns

Οι νέες ρυθμίσεις που κάναμε θα πρέπει να φαίνονται όπως στην Εικόνα Δ.11.

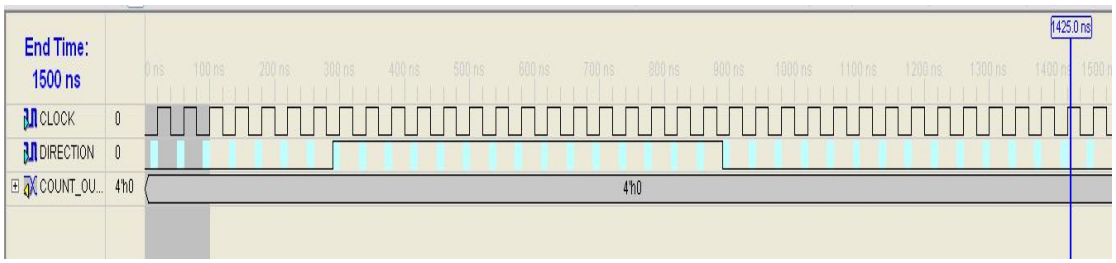


Εικόνα Δ.11: Αρχικοποίηση τιμών χρόνου και ρολογιού

Από το παράθυρο Sources του εργαλείου ISE επιλέγουμε την προσομοίωση συμπεριφοράς *Behaviral Simulation* Εικόνα Δ.12 και με διπλό click στο αρχείο *counter_tbw.tbw* θα εμφανιστεί η κυματομορφή Εικόνα Δ.13, στην οποία μπορούμε να θέσουμε και τιμές εισαγωγής που επιθυμούμε. Εμείς θέσαμε στα 300ns ανοδική ακμή και μετά καθοδική στα 900ns.



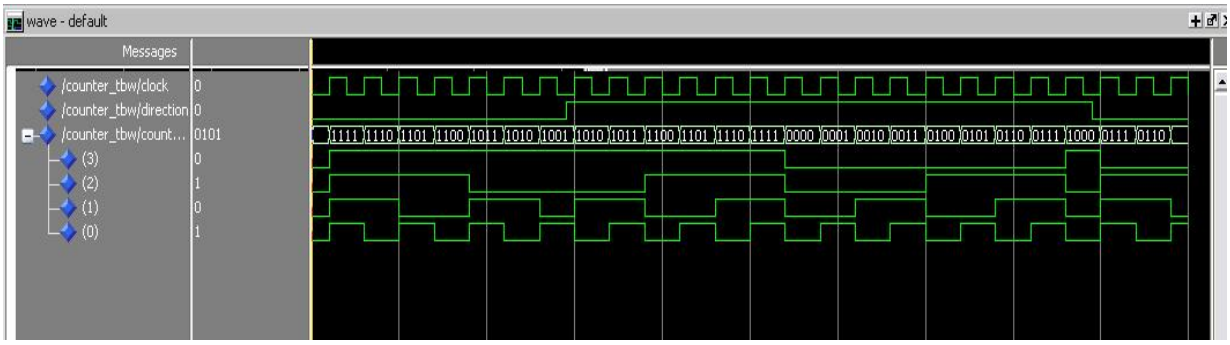
Εικόνα Δ.12: Επιλογή Behavioral Simulation από το πρόγραμμα CAD ISE



Εικόνα Δ.13: Εμφάνιση κυματομορφής από το Test Bench Waveform από το πρόγραμμα CAD ISE

Όμως υπάρχει και μία επιπλέον βοήθεια και αυτή έρχεται από ένα άλλο πρόγραμμα προσομοίωσης το *ModelSim* [15] το οποίο είναι συμβατό με τα αρχεία του ISE. Γενικά κάποιες εταιρίες κατασκευής εργαλείων σχεδίασης CAD το περιλαμβάνουν μέσα στο πακέτο λογισμικού που προσφέρουν και άλλες το προτείνουν ως ένα επιπλέον λογισμικό για ελέγχους και προσομοιώσεις.

Επιλέγοντας από την καρτέλα *Processes* το "**+**" βρίσκεται το *Simulate Behavioral Model*, με διπλό click πάνω, ενεργοποιείται το ModelSim και εμφανίζεται η κυματομορφή εκτελούμενη με την είσοδο που της θέσαμε (Εικόνα Δ.14). Με αυτόν το τρόπο μπορούμε να διαπιστώσουμε τις παραγόμενες κυματομορφές εξόδου αλλά και τα εσωτερικά σήματα που παράγονται.



Εικόνα Δ.14: Εκτέλεση κυματομορφής με το εργαλείο ModelSim

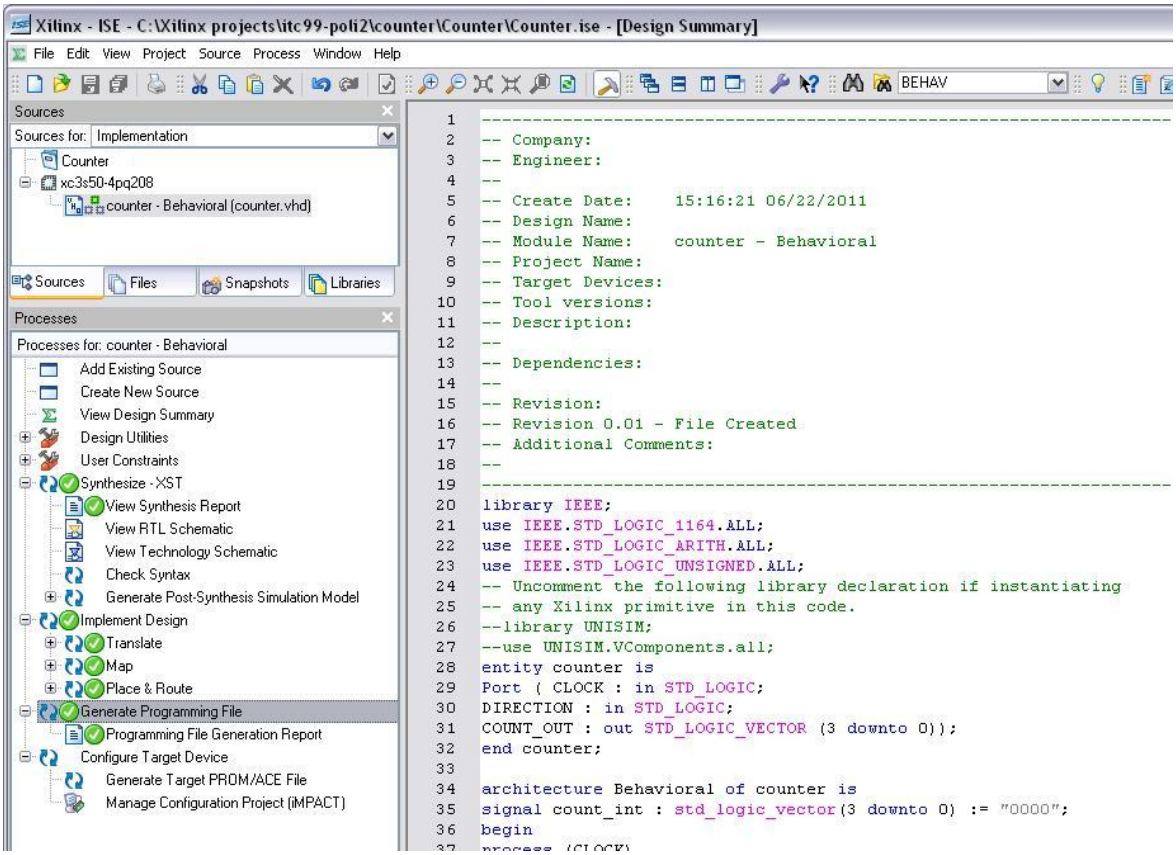
Ο δεύτερος τρόπος υλοποίησης της λειτουργικής προσομοίωσης είναι χρησιμοποιώντας πάγκο εργασίας με χρήση της γλώσσας VHDL. Απλά διαλέγουμε *Project→New Source→VHDL Test Bench*. Ένα ενδεικτικό αρχείο για την κατασκευή κυματομορφών με χρήση της γλώσσας VHDL είναι αυτό στην Ενότητα Z.2 στο Παράρτημα Z με το κωδικό όνομα *file even_par.vhd* [41]. Κατά τα άλλα ακολουθείται η ίδια διαδικασία με τα παραπάνω για την παραγωγή των κυματομορφών εξόδου αλλά και των εσωτερικών σημάτων.

Δ.5 Σύνθεση και Υλοποίηση

Αν η λειτουργική προσομοίωση είναι σωστή τότε προχωράμε στη σύνθεση της σχεδίασης πατώντας με διπλό click την ένδειξη *Synthesize–XST*. Όταν το ISE εκτελέσει τη σύνθεση μπορούμε να προχωρήσουμε και στην υλοποίηση της σχεδίασης πατώντας με διπλό click την επιλογή *Implement Design*. Αν όλα πάνε καλά τότε θα εμφανιστεί μια πράσινη ένδειξη σε όλες τις επιλογές όπως φαίνεται στην Εικόνα Δ.15.

Σημείωση 1 : Αν επιλέξουμε απευθείας να πατήσουμε την υλοποίηση (Implement Design) το ISE αυτόματα θα κάνει πρώτα τη σύνθεση (Synthesize) και μετά θα προχωρήσει στην υλοποίηση.

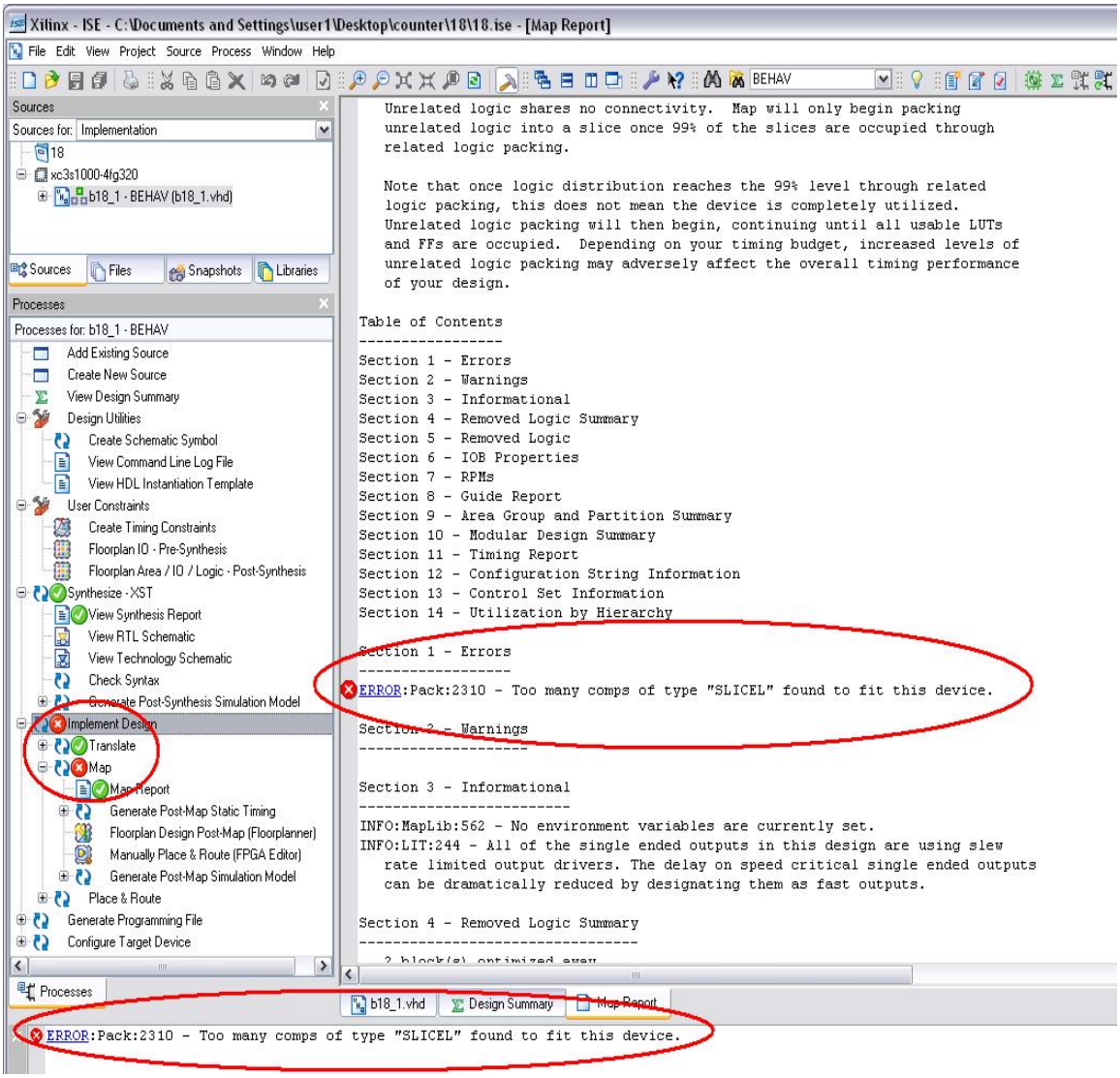
Σημείωση 2 : στη δεύτερη επιλογή Implement Design θα μπορούσε ο χρήστης να εκτελέσει μια μια τις επιλογές *Translate*, *Map* και *Place & Route* με διπλό click. Οι τρεις αυτές ενδείξεις εκτελούν τη χωροθέτηση την τοποθέτηση και τη δρομολόγηση αντίστοιχα όπως έχουμε δει και στην Ενότητα 5.1.1.



Εικόνα Δ.15: Σύνθεση και υλοποίηση μιας σχεδίασης από το πρόγραμμα CAD ISE

Έχοντας ολοκληρώσει σωστά τις παραπάνω ενέργειες μπορούμε να υλοποιήσουμε και το αρχείο BitStream εκτελώντας την επιλογή *Generate Programming File*. Τέλος όταν δημιουργηθεί το αρχείο BitStream, πηγαίνοντας και εκτελώντας την επιλογή *Configure Target Device* φορτώνεται το πρόγραμμα που σχεδιάσαμε στη συσκευή FPGA. Εμείς σε αυτό το σημείο δεν θα προχωρήσουμε μιας και το τελευταίο δεν αποτελεί αντικείμενο μελέτης αυτής της μεταπτυχιακής διατριβής.

Αν σε περίπτωση κάποιο από τα παραπάνω είχε αστοχία στην εκτέλεση, το ISE προειδοποιεί με μηνύματα την ύπαρξη λάθους. Για παράδειγμα βλέπουμε στην Εικόνα Δ.16 ένα σφάλμα στην υλοποίηση, η συγκεκριμένη αιτία είναι ότι η σχεδίαση είναι αρκετά μεγάλη για τη συσκευή που έχει προταθεί. Δεν αρκεί στην υφιστάμενη συσκευή ο χώρος για τη χωροθέτηση των λογικών μονάδων, οπότε θα πρέπει να μεταβούμε σε μια μεγαλύτερη συσκευή. Το σφάλμα και η αιτιολόγηση του φαίνονται στην Εικόνα Δ.16.



Εικόνα Δ.16: Σφάλμα και αιτιολόγηση από το πρόγραμμα CAD ISE

Δ.6 Χρονικοί Περιορισμοί και Αρχείο Υλοποίησης Design Summary

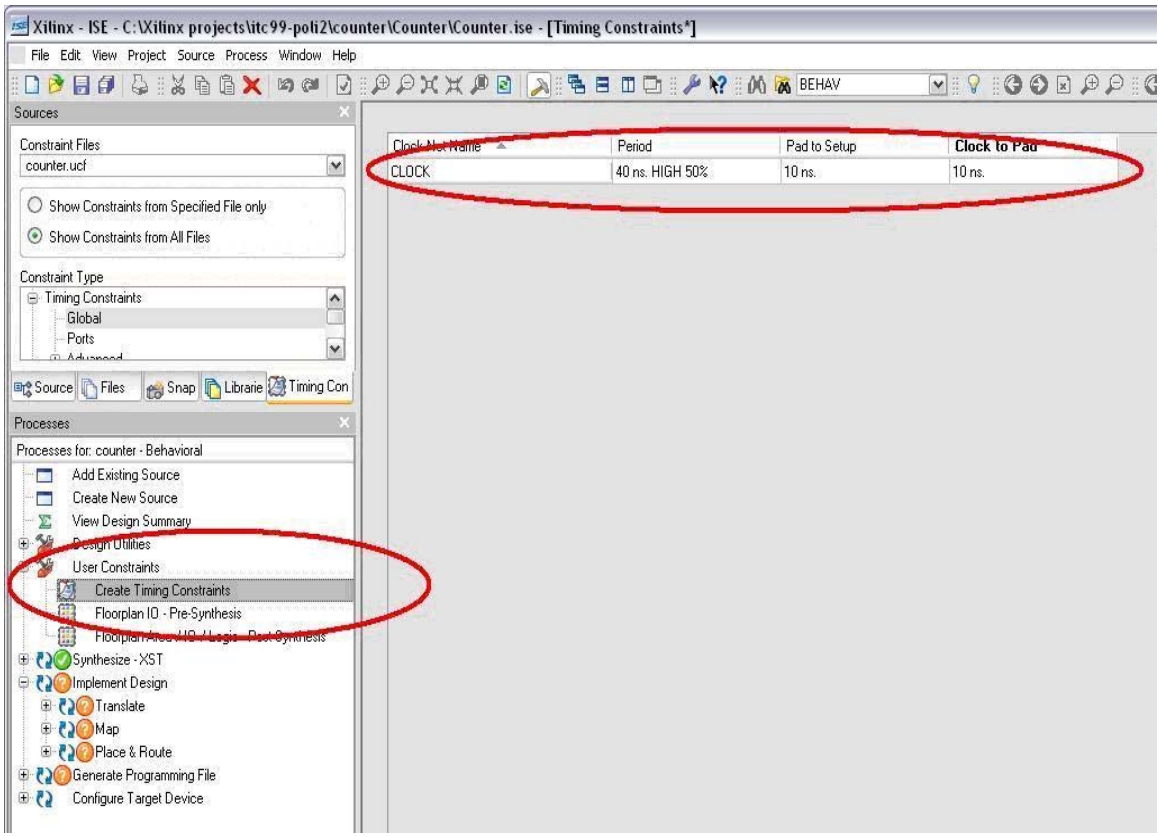
Μία άλλη σημαντική παράμετρος είναι η δημιουργία χρονικών περιορισμών, αυτοί οι περιορισμοί μπορούν υπάρχουν σε κάποιες συγκεκριμένες χρονικές στιγμές όπου σύμφωνα με τη σχεδίαση και τις αρχικές προδιαγραφές του συστήματος, θα πρέπει να λειτουργούν.

Αυτοί είναι :

- να θέσουμε χρονικά όρια στην περίοδο του χρόνου ρολογιού,
- την αρχικοποίηση, δηλαδή το *Setup Time* αλλά και

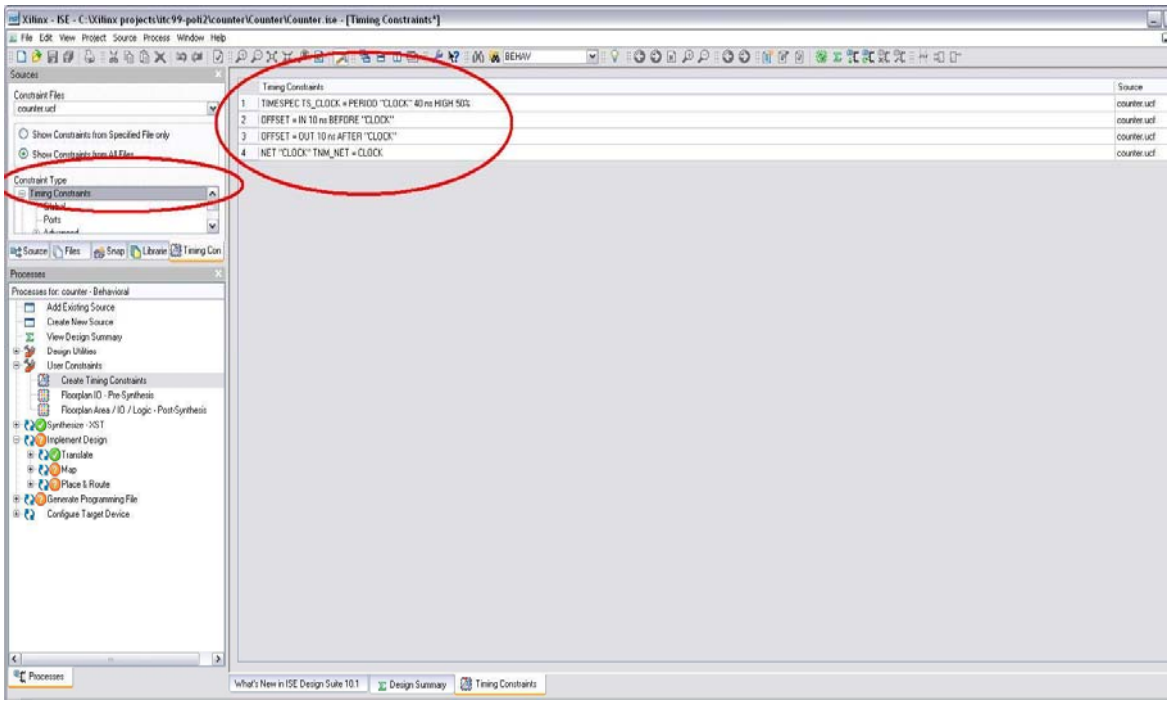
- το χρόνο απόκρισης και διάδοσης των δεδομένων μέσα στους ακροδέκτες, δηλαδή το *Hold Time*.

Έχοντας χρονικούς περιορισμούς θα δούμε αν η σχεδίαση μας ανταποκρίνεται θετικά σε αυτά. Για να το πετύχουμε αυτό θα πρέπει να δημιουργήσουμε ένα αρχείο το *<arxeio>.ucf*, που τα αρχικά σημαίνουν *User Constraints File*. Για να το κάνουμε αυτό πηγαίνουμε στο παράθυρο *Source* επιλέγουμε από την πτυσσόμενη λίστα το *Implementation* και αμέσως μετά το αρχείο *counter.vhdl* (συνέχεια του παραδείγματος μας). Στη συνέχεια πάμε στο παράθυρο *Processes* και επιλέγουμε από το *User Constraints* το *Create Timing Constraints*, αυτόματα το εργαλείο θα ανοίξει το παράθυρο που φαίνεται στην Εικόνα Δ.17 στο οποίο ο χρήστης συμπληρώνει τους χρόνους πού θέλει. Εμείς θα συμπληρώσουμε τους χρόνους όπως αυτοί φαίνονται στην Εικόνα Δ.17 και στη συνέχεια θα πατήσουμε enter. Παράλληλα με το άνοιγμα του παραθύρου για τη συμπλήρωση των χρόνων, το εργαλείο ISE δημιουργεί το αρχείο *counter.ucf* το οποίο και θα συμπεριλάβει στο φάκελο με όλα τα αρχεία του project.



Εικόνα Δ.17: Δημιουργία χρονικών περιορισμών στο Create Timing Constraints από το πρόγραμμα CAD ISE

Όταν τελειώσουμε και θέλουμε να δούμε αυτά τα χαρακτηριστικά πάμε στο *Constraint Type* και επιλέγουμε *Timing Constraints*, θα εμφανιστούν τα χαρακτηριστικά όπως φαίνονται στην Εικόνα Δ.18.



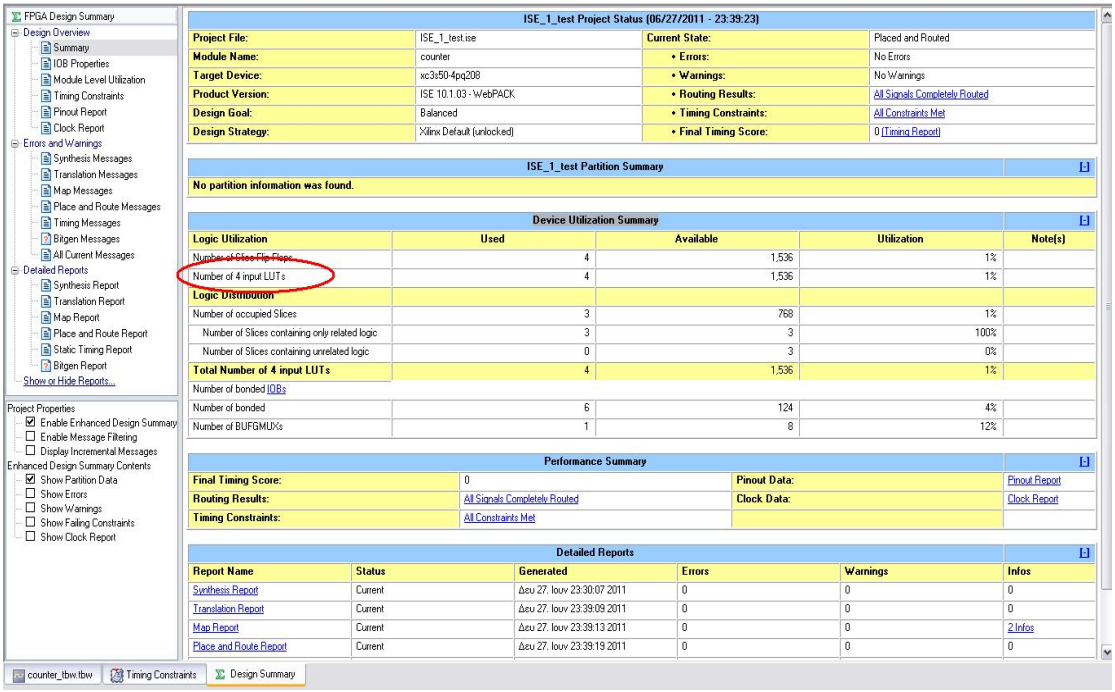
Εικόνα Δ.18: Χαρακτηριστικά χρονικών περιορισμών Timing Constraints από το πρόγραμμα CAD ISE

Αφού θέσουμε τους περιορισμούς θα πρέπει να υλοποιήσουμε τη σχεδίαση μας. Εκτελώντας την όπως μάθαμε από τις παραπάνω υποδείξεις διαπιστώνουμε το αποτέλεσμα αυτής της υλοποίησης.

Το ISE, όταν αρχικά εκτελεί μία σχεδίαση δεν λαμβάνει υπόψη τους χρονικούς περιορισμούς, διότι δεν υπάρχει κάποιο αρχείο χρονικών περιορισμών. Αυτό που κάνει είναι να εκτελεί την αρχική σχεδίαση και μετά να βγάζει αναφορές σχετικά με τις όποιες καθυστερήσεις αλλά και την προτεινόμενη συχνότητα λειτουργίας της σχεδίασης. Το ISE βγάζει αναφορές για την περίοδο του ρολογιού από όπου κανείς μπορεί να βγάλει τη μέγιστη συχνότητα λειτουργίας, εν αντιθέσει του Quartus που το βγάζει απευθείας με το FMax. Επιπλέον το ISE βγάζει τους προτεινόμενους χρόνους Setup Time και Hold Time, κάτι το οποίο δεν κάνει το Quartus και το οποίο τα ενσωματώνει αυτόματα.

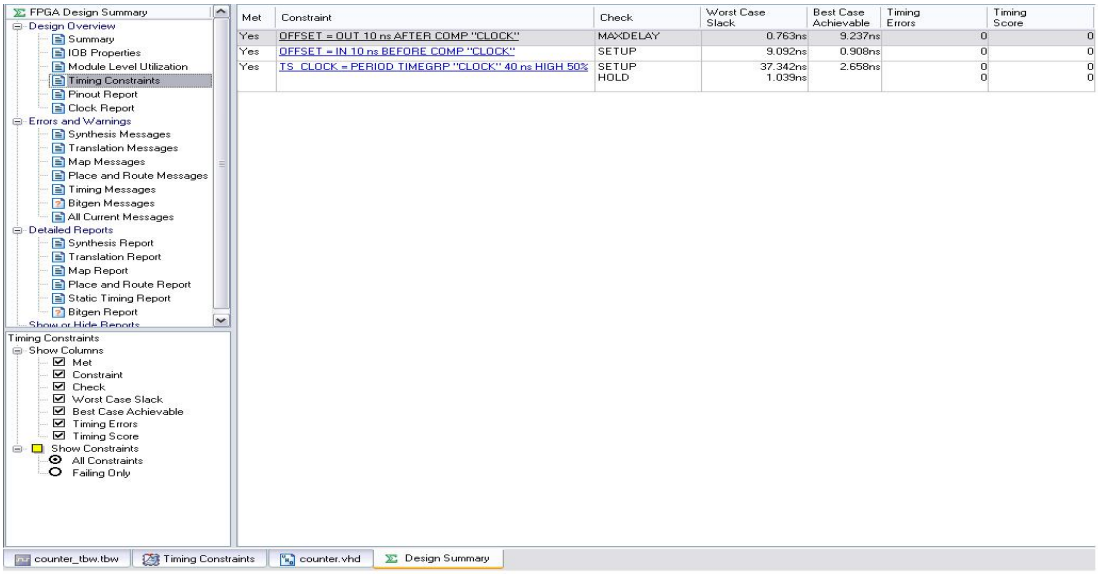
Το ISE σε κάθε υλοποίηση παράγει ένα αρχείο με όλα τα στατιστικά στοιχεία που διέπουν μια υλοποίηση. Για να δούμε το αρχείο αυτό και όλα τα χαρακτηριστικά αρκεί να πάμε στο παράθυρο Processes και να επιλέξουμε το *View Design Summary*. Στο παράθυρο που θα ανοίξει

θα δούμε κάποια χαρακτηριστικά τα οποία έχουν ομαδοποιηθεί. Για παράδειγμα αν θέλουμε να δούμε πόσα LUT έχουν χρησιμοποιηθεί στη σχεδίαση μας αρκεί να δούμε στο *Device Utilization Summary* το *Number of 4-input LUT* Εικόνα Δ.19, να θυμηθούμε ότι κάθε LUT αποτελεί ένα λογικό στοιχείο στην αρχιτεκτονική του Spartan-3. Παράλληλα μπορεί κανείς να δει και άλλα χαρακτηριστικά όπως ενεργοποιημένοι ακροδέκτες είσοδο εξόδου χρησιμοποιημένη μνήμη RAM κτλ κτλ. Πάντα όμως όλα έχουν σχέση με την εκάστοτε σχεδίαση και τη συσκευή που έχει επιλεγεί για υλοποίηση.



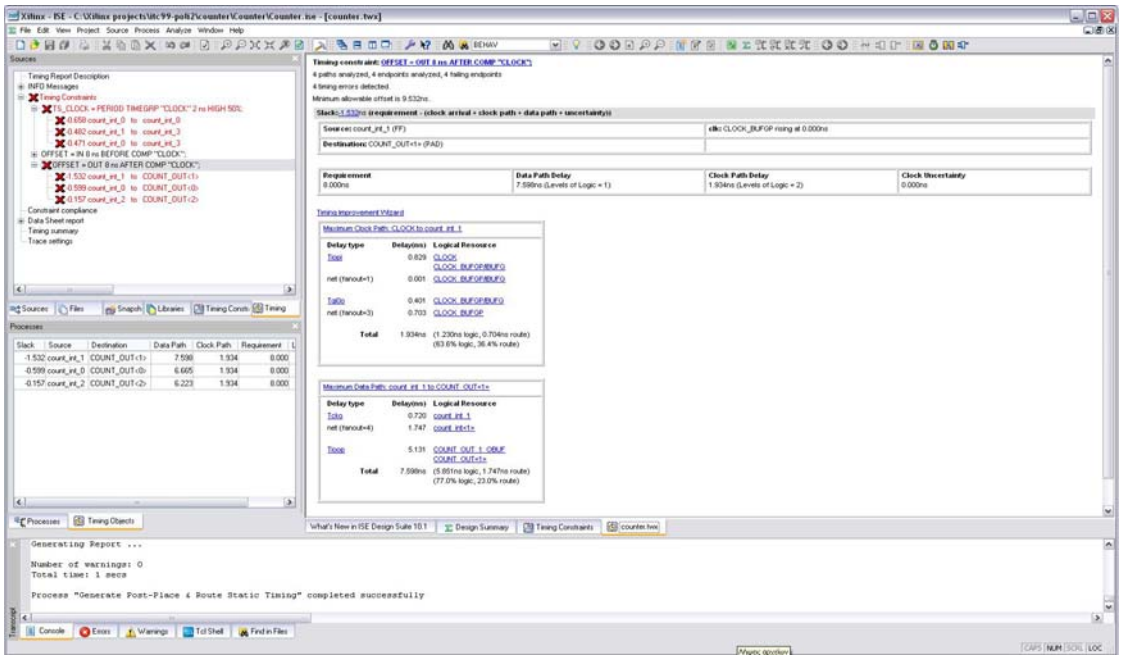
Εικόνα Δ.19: Παραγόμενο αρχείο τεκμηρίωσης της υλοποίησης Design Summary από το πρόγραμμα CAD ISE

Ιδιαίτερη αναφορά πρέπει να γίνει για τους χρονικούς περιορισμούς που έχουμε βάλει από την παραπάνω διαδικασία. Στην επιλογή *Performance Summary* διαλέγοντας το *All Constraints Met* μπορούμε να δούμε τα αποτελέσματα. Στο νέο παράθυρο που θα εμφανιστεί μπορεί κανείς να δει τη μεγαλύτερη καθυστέρηση που σημειώθηκε στην ένδειξη *MAXDELAY*, τα *SETUP* και *HOLD TIME* Εικόνα Δ.20. Η ίδια επιλογή μπορεί να ενεργοποιηθεί και από την επιλογή πάνω αριστερά *Design Overview* πατώντας την ένδειξη *Timing Constraints*.



Εικόνα Δ.20: Αποτελέσματα υλοποίησης Timing Constraints από το Design Summary του προγράμματος CAD ISE

Αν τώρα κάποιος περιορισμός δεν είναι επιτρεπτός αυτό φαίνεται με την αρνητική τιμή που έχουν οι ενδείξεις MAXDELAY, τα SETUP και HOLD TIME. Για λεπτομέρεια μπορούμε να πατήσουμε πάνω σε κάποια ένδειξη και να εμφανιστεί το αρχείο **.twx* το οποίο αποτελεί μια αναφορά σε κάθε ένα από τα προαναφερόμενα. Για αρνητικές τιμές ή χρόνους που δεν καλύπτουν τους χρονικούς περιορισμού της σχεδίασης, απεικονίζονται με κόκκινο (Εικόνα Δ.21).



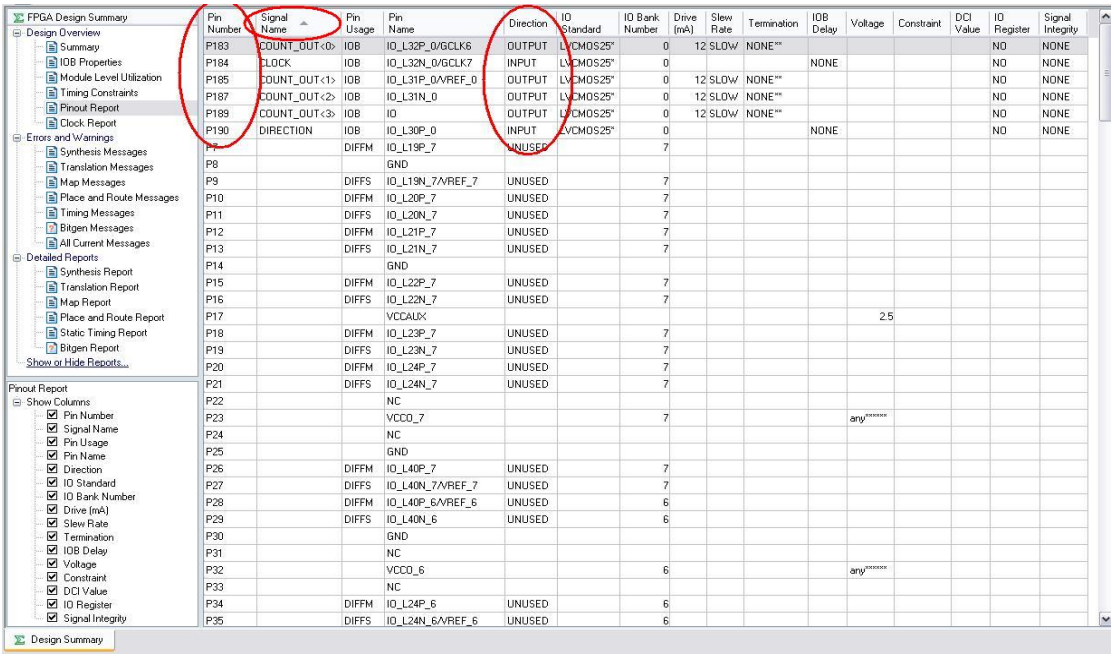
Εικόνα Δ.21: Αποτελέσματα αρνητικής υλοποίησης Timing Constraints από το Design Summary του προγράμματος CAD ISE

Από τους όποιους περιορισμούς μπορούμε να συμπεράνουμε τη συχνότητα λειτουργίας αλλά και τους χρόνους Setup Time και Hold Time. Αυτές οι λειτουργίες μας ενδιαφέρουν πολύ γιατί θα αποτελέσουν αντικείμενο μελέτης και παρακολούθησης στο πειραματικό μέρος.

Δ.7 Ακροδέκτες Εισόδου Εξόδου και Pinout Report

Η χαρτογράφηση των ακροδεκτών εισόδου εξόδου δε θα παρουσιαστεί στη συνέχεια, μιας και το αντικείμενο μελέτης της μεταπτυχιακής διατριβής δεν έχει σχέση με αυτή τη λειτουργία. Ουσιαστικά μπορεί ο χρήστης να επιλέξει από μόνος του τους ακροδέκτες εισόδου και εξόδου αλλά και των άλλων χαρακτηριστικών όπως ρολόι κτλ.

Απλά αυτό που έχει ενδιαφέρον και πρέπει να δούμε είναι η αναφορά που βγάζει το ISE για την ενεργοποίηση των ακροδεκτών. Επιλέγουμε από το Design Summary το *Pinout Report*, το οποίο θα μας δείχνει ποιοι ακροδέκτες χρησιμοποιούνται σε μια σχεδίαση. Σε συνέχεια της σχεδίασης μας θα δούμε το Pinout Report του Counter. Για να δούμε ποιοι ακροδέκτες χρησιμοποιούνται πατάμε την επιλογή *Signal Name* και αυτόματα θα εμφανιστούν στην αρχή, παράλληλα στην επιλογή *Direction* περιγράφουν πια χρήση επιτελούν. Όπως βλέπουμε από το report χρησιμοποιούνται έξη ακροδέκτες δυο για Input και τέσσερις για Output (Εικόνα Δ.22). Μια άλλη παρόμοια αλλά πιο σύντομη διαδικασία είναι να πατήσουμε την επιλογή IOB Properties η οποία εμφανίζει μόνο τους ακροδέκτες που χρησιμοποιούνται.



Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	IO Bank Number	Drive (mA)	Slew Rate	Termination	IOB Delay	Voltage	Constraint	DCI Value	IO Register	Signal Integrity
P183	COUNT_OUT<0>	IOB	IO_L32P_0/GCLK6	OUTPUT	LVCMOS25*	0	12	SLOW	NONE**					NO	NONE
P184	CLOCK	IOB	IO_L32N_0/GCLK7	INPUT	LVCMOS25*	0				NONE				NO	NONE
P185	COUNT_OUT<1>	IOB	IO_L31P_0/VREF_0	OUTPUT	LVCMOS25*	0	12	SLOW	NONE**					NO	NONE
P187	COUNT_OUT<2>	IOB	IO_L31N_0	OUTPUT	LVCMOS25*	0	12	SLOW	NONE**					NO	NONE
P189	COUNT_OUT<3>	IOB	IO	OUTPUT	LVCMOS25*	0	12	SLOW	NONE**					NO	NONE
P190	DIRECTION	IOB	IO_L30P_0	INPUT	LVCMOS25*	0				NONE				NO	NONE
P7		DIFFM	IO_L19P_7	UNUSED		7									
P8			GND												
P9		DIFFS	IO_L19N_7/VREF_7	UNUSED		7									
P10		DIFFM	IO_L20P_7	UNUSED		7									
P11		DIFFS	IO_L20N_7	UNUSED		7									
P12		DIFFM	IO_L21P_7	UNUSED		7									
P13		DIFFS	IO_L21N_7	UNUSED		7									
P14			GND												
P15		DIFFM	IO_L22P_7	UNUSED		7									
P16		DIFFS	IO_L22N_7	UNUSED		7									
P17			VCCAUX								2.5				
P18		DIFFM	IO_L23P_7	UNUSED		7									
P19		DIFFS	IO_L23N_7	UNUSED		7									
P20		DIFFM	IO_L24P_7	UNUSED		7									
P21		DIFFS	IO_L24N_7	UNUSED		7									
P22			NC												
P23			VCCO_7			7					any***				
P24			NC												
P25			GND												
P26		DIFFM	IO_L40P_7	UNUSED		7									
P27		DIFFS	IO_L40N_7/VREF_7	UNUSED		7									
P28		DIFFM	IO_L40P_6/VREF_6	UNUSED		6									
P29		DIFFS	IO_L40N_6	UNUSED		6									
P30			GND												
P31			NC												
P32			VCCO_6			6					any***				
P33			NC												
P34		DIFFM	IO_L24P_6	UNUSED		6									
P35		DIFFS	IO_L24N_6/VREF_6	UNUSED		6									

Εικόνα Δ.22: Αποτελέσματα υλοποίησης Pinout Report από το Design Summary του προγράμματος CAD ISE

Σε αυτή την ενότητα είδαμε πως μπορούμε να σχεδιάσουμε και να δημιουργήσουμε αρχεία VHDL με το εργαλείο σχεδίασης ISE. Συγκεκριμένα είδαμε το τρόπο σχεδίασης με κώδικα VHDL, πως ελέγχουμε τη σχεδίαση μας αλλά και τη διαδικασία υλοποίησης για την παραγωγή του αρχείου Bitstream. Είδαμε το τρόπο που θέτουμε χρονικούς περιορισμούς σε μία σχεδίαση αλλά και την τελική σύνοψη της σχεδίασης μας σε ένα παράθυρο όπως και σε διάφορα αρχεία log.

Για περαιτέρω μελέτη του εργαλείου σχεδίασης ISE της εταιρίας Xilinx δείτε τα αντίστοιχα έγγραφα «*ISE 10.1 Quick Start Tutorial*» [32] και «*ISE Design Suite 10.1 Software Manuals*» [33], τα οποία αναφέρονται στη βιβλιογραφία.

Παράρτημα Ε

Παραγόμενος Κώδικας των Ερωτημάτων SQL, σε VHDL

Σε αυτό το παράρτημα θα ανατεθούν όλοι οι κώδικες VHDL που χρησιμοποιήθηκαν στα παραδείγματα προσομοίωσης των επερωτήσεων Q1, Q2, Q3 και Q4.

Ε.1 Κώδικας Επερώτησης Q1

Ε.1.1 Αρχείο TB_FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TB_FILE_READ is
---port(                                    -- not used for simulation only for execute
        --clk          : in  std_logic;     -- not used for simulation only for execute
        --rst          : in  std_logic;     -- not used for simulation only for execute
```

```
		--s_i_dieythinsi:std_logic_vector(3 downto 0) -- not used for simulation only for execute
--);                                                    -- not used for simulation only for execute
end TB_FILE_READ;
architecture Behavioral of TB_FILE_READ is
component FILE_READ
		generic (stim_file:	string  := "R.txt");
		port(
					CLK		: in  std_logic;
					RST		: in  std_logic;
					Y		: out std_logic_vector(23 downto 0);
					EOG		: out std_logic);
end component;
component FILE_WRITE
		generic (	write_file:	string  := "OUT.TXT");
		port(
			CLK	: in std_logic;
			RST	: in std_logic;
			X: in std_logic_vector(23 downto 0);
			EOG	: in std_logic;
			valid_bit  : in std_logic);
end component;
component Q1_select
		Port (	CLK	: in  std_logic;
			RST	: in  std_logic;
			in_data	: in  STD_LOGIC_VECTOR (23 downto 0);
			valid_bit : out STD_LOGIC;
			out_data  : out  STD_LOGIC_VECTOR (23 downto 0));
end component;
signal rst		:std_logic;
signal clk		:std_logic := '1';
signal eog		:std_logic;
signal s_valid_bit	:std_logic;
--signal for data
signal s_data_in :	std_logic_vector(23 downto 0) := (others => '0');
signal s_data_out:	std_logic_vector(23 downto 0) := (others => '0');
begin
rst <= '0', '1' after 40 ns, '0' after 100 ns;
clk <= not clk after 10 ns;
input_stim: FILE_READ
		port map(  CLK  => clk,RST  => rst,Y    => s_data_in,EOG  => eog);
select_Q1: Q1_select
		port map(  CLK	=> clk,RST	=> rst,in_data	=> s_data_in,valid_bit  => s_valid_bit,out_data	=> s_data_out);
```

```
output1_stim: FILE_WRITE
        port map(  CLK  => clk,RST  => rst,X  => s_data_out,EOG  => eog,valid_bit  => s_valid_bit);
end Behavioral;
```

E.1.2 Αρχείο FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
use std.textio.all;
use work.txt_util.all;
entity FILE_READ is
generic ( stim_file:     string  := "R.txt" );
 port(       CLK          : in  std_logic;
             RST          : in  std_logic;
              Y           : out std_logic_vector(23 downto 0)          := (others => '0');
              EOG          : out std_logic    );
end FILE_READ;
architecture Behavioral of FILE_READ is
function str_to_stdvec(inp: string) return std_logic_vector is
     variable temp: std_logic_vector(inp'range) := (others => '0');
  begin
    for i in inp'range loop
      if (inp(i) = '1') then
        temp(i) := '1';
      elsif (inp(i) = '0') then
        temp(i) := '0';
      end if;
    end loop;
    return temp;
  end function str_to_stdvec;
file stimulus: TEXT open read_mode is stim_file;
begin
-- read data and control information from a file
receive_data: process
                variable l: line;
                variable s: string(24 downto 1);
                begin
                        EOG <= '0';
                        -- wait for Reset to complete
```

```
                                wait until RST='1';
                                wait until RST='0';
                                while not endfile(stimulus) loop
                                -- read digital data from input file
                                readline(stimulus, l);
                                read(l, s);
                                --Y <=  str_to_stdvec(s);
                                Y <= to_std_logic_vector(s);
                                wait until CLK = '1';
                                end loop;
                        print("I@FILE_READ: reached end of "& stim_file);
                        EOG <= '1';
                        wait;
 end process receive_data;
end Behavioral;
```

E.1.3 Αρχείο Q1_select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Q1_select is
Port (                  clk      : in  std_logic;                                    -- clock input
                        rst      : in  std_logic;                                    -- reset input
                        in_data    : in  STD_LOGIC_VECTOR (23 downto 0);       -- data input
                        valid_bit  : out  STD_LOGIC;                    -- valid bit output
                        out_data   : out  STD_LOGIC_VECTOR (23 downto 0)     -- data output);
end Q1_select;
architecture Behavioral of Q1_select is
-- control signals
signal A1               :  std_logic_vector(7 downto 0);                -- the first element A1 of the tuple
signal A2               :  std_logic_vector(7 downto 0);                -- the second element A2 of the tuple
signal A3               :  std_logic_vector(7 downto 0);                -- the third element A3 of the tuple
signal s_in_data        :  STD_LOGIC_VECTOR (23 downto 0); -- the signal from input data
begin
-- signals  initialization
s_in_data <= in_data;
A1 <= s_in_data (23 downto 16);             -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);             -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);             -- initialize third element A3 of the tuple
---------------------------------------------- Q1 ----------------------------------------------------------------------------------------------------------
```

```
Q1: process (clk)                    -- start process Q1
        begin
        if clk'event and clk='1' then                    -- synchronization whit clock
                if (A3 > "00000010") then                -- the control element A3 > 2
                        out_data <= s_in_data;           -- output correct data
                        valid_bit <= '1' ;               -- valid bit = '1'
                else
                        out_data <= (others =>'0'); -- output null data
                        valid_bit <= '0' ;       -- valid bit = '0'
                        end if;
        end if;
end process Q1;                              --end process Q1
end Behavioral;
```

E.1.4 Αρχείο FILE_WRITE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE is
generic (       write_file:     string  := "OUT.TXT"        );
 port(
        CLK   : in std_logic;
        RST   : in std_logic;
        X   : in std_logic_vector(23 downto 0);
        EOG         : in std_logic;
        valid_bit   : in std_logic);
end FILE_WRITE;
architecture Behavioral of FILE_WRITE is
file out_file: TEXT open write_mode is write_file;
signal ok : std_logic :='1';
begin
delay: process (clk)
 begin
        if(clk'event and clk = '1')then
                if(eog = '1')then
                        ok <= '0';
                elsif(eog = '0')then
                        ok <= '1';
```

```
                        end if;
                end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                        if (ok  = '1') and (valid_bit = '1')then
                        --if (valid_bit = '1')then
                                -- write digital data into log file
                                write(OUTLINE, str(X));          --add variable to outline variable for output.
                                writeline(out_file, OUTLINE);    --writes outline variable to file.
                                --print(l_file, str(x1)& " "& hstr(x2)& "h");
                        end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.2 Κώδικας Επερώτησης Q2

E.2.1 Αρχείο TB_FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TB_FILE_READ is
---port(                -- not used for simulation only for execute
        --clk      : in  std_logic;          -- not used for simulation only for execute
        --rst      : in  std_logic;          -- not used for simulation only for execute
        --s_i_dieythinsi      :std_logic_vector(3 downto 0)    -- not used for simulation only for execute
        --);                                 -- not used for simulation only for execute
end TB_FILE_READ;
architecture Behavioral of TB_FILE_READ is
component FILE_READ
        generic (stim_file:     string  := "S.txt");
        port(                   CLK       : in  std_logic;
```

```
                                RST          : in  std_logic;
                                Y            : out std_logic_vector(23 downto 0);
                                EOG          : out std_logic                      );
end component;
component FILE_WRITE
        generic (   write_file:     string  := "OUT.TXT");
        port(                   CLK   : in std_logic;
                                RST   : in std_logic;
                                X: in std_logic_vector(23 downto 0);
                                EOG          : in std_logic;
                                 valid_bit : in std_logic                      );
end component;
component Q2_select
Port (      clk        : in  std_logic;                          -- clock input
            rst        : in  std_logic;                          -- reset input
            in_data         : in  STD_LOGIC_VECTOR (23 downto 0);   -- data input
            valid_bit   : out  STD_LOGIC;                       -- valid bit A output
            out_data    : out  STD_LOGIC_VECTOR (23 downto 0)   -- data output);
end component;
signal rst                  :std_logic;
signal clk                  :std_logic := '1';
signal eog                  :std_logic;
signal s_valid_bit          :std_logic;
--signal y:                         std_logic_vector(31 downto 0);
signal data_in:             std_logic_vector(23 downto 0) := (others => '0');
signal data_out:            std_logic_vector(23 downto 0) := (others => '0');
begin
rst <= '0', '1' after 40 ns, '0' after 100 ns;
clk <= not clk after 10 ns;
input_stim: FILE_READ
        port map(  CLK  => clk,RST  => rst,Y    => data_in,EOG  => eog);
select_Q2: Q2_select
        port map(  clk  => clk,  rst    => rst,in_data       => data_in,valid_bit  => s_valid_bit,out_data => data_out);
output_stim: FILE_WRITE
        port map(  CLK  => clk,RST  => rst,X  => data_out,EOG  => eog, valid_bit => s_valid_bit);
end Behavioral;
```

E.2.2 Αρχείο FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
use std.textio.all;
use work.txt_util.all;
entity FILE_READ is
generic (  stim_file:      string  := "S.txt"  );
 port(
     CLK            : in  std_logic;
     RST            : in  std_logic;
     Y              : out std_logic_vector(23 downto 0)        := (others => '0');
     EOG            : out std_logic    );
end FILE_READ;
architecture Behavioral of FILE_READ is
function str_to_stdvec(inp: string) return std_logic_vector is
      variable temp: std_logic_vector(inp'range) := (others => '0');
   begin
      for i in inp'range loop
         if (inp(i) = '1') then
            temp(i) := '1';
         elsif (inp(i) = '0') then
           temp(i) := '0';
         end if;
      end loop;
      return temp;
   end function str_to_stdvec;
 file stimulus: TEXT open read_mode is stim_file;
begin
-- read data and control information from a file
receive_data: process
                        variable l: line;
                        variable s: string(24 downto 1);
                        begin
                                EOG <= '0';
                                -- wait for Reset to complete
                                wait until RST='1';
                                wait until RST='0';
                                while not endfile(stimulus) loop
                                -- read digital data from input file
                                readline(stimulus, l);
                                read(l, s);
                                --Y <=  str_to_stdvec(s);
                                Y <= to_std_logic_vector(s);
```

```
                        wait until CLK = '1';
                        end loop;
                print("I@FILE_READ: reached end of "& stim_file);
                EOG <= '1';
                wait;
 end process receive_data;
end Behavioral;
```

E.2.3 Αρχείο Q2_select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Q2_select is
Port (          clk     : in  std_logic;                                -- clock input
                rst     : in  std_logic;                                -- reset input
                in_data     : in  STD_LOGIC_VECTOR (23 downto 0);   -- data input
                valid_bit   : out  STD_LOGIC;                           -- valid bit A output
                out_data    : out  STD_LOGIC_VECTOR (23 downto 0) );   -- data output
end Q2_select;
architecture Behavioral of Q2_select is
-- control signals
signal B1   :       std_logic_vector(7 downto 0);                       -- the first element A1 of the tuple
signal B2   :       std_logic_vector(7 downto 0);                       -- the second element A2 of the tuple
signal B3   : std_logic_vector(7 downto 0);                             -- the third element A3 of the tuple
signal s_in_data    : STD_LOGIC_VECTOR (23 downto 0);       -- the signal from input data
signal s_in_data_B  : STD_LOGIC_VECTOR (23 downto 0);       -- the signal from input data
signal valid_bit_A  :  STD_LOGIC :='0';                 -- the first valid bit output
signal valid_bit_B  :  STD_LOGIC :='0';                 -- the second valid bit output
begin
-- signals  initialization
s_in_data <= in_data;
B1 <= s_in_data (23 downto 16);             -- initialize first element B1 of the tuple
B2 <= s_in_data (15 downto  8);             -- initialize second element B2 of the tuple
B3 <= s_in_data ( 7 downto  0);             -- initialize third element B3 of the tuple
---------------------------------------------- Q2_A ----------------------------------------------------------------------------------------------
Q2_A: process (clk)                                 -- start process Q2_A
        begin
        if clk'event and clk='1' then               -- synchronization whit clock
                if (B1 = "00000001") then               -- the control element B1 = '1'
                        valid_bit_A <= '1' ;            -- valid bit A = '1'
```

```
                        s_in_data_B <= in_data;          -- correct data
                else
                        valid_bit_A <= '0' ;             -- valid bit A = '0'
                        end if;
        end if;
end process Q2_A;                       -- end process Q2_A
------------------------------------------- Q2_B ----------------------------------------------------------------------------------------------------
Q2_B: process (clk)             -- start process Q2_B
        begin
        if clk'event and clk='1' then           -- synchronization whit clock
                if (B2 > "00000001") then       -- the control element B2 > 1
                        valid_bit_B <= '1' ;             -- valid bit B = '1'
                        s_in_data_B <= in_data;          -- correct data
                else
                        valid_bit_B <= '0' ;     -- valid bit B = '0'
                        end if;
        end if;
end process Q2_B;                               -- end process Q2_B
------------------------------------------- Q2_C ----------------------------------------------------------------------------------------------------
Q2_C:process (clk,valid_bit_A,valid_bit_B)   -- start process Q2_C
        begin
        --      if clk'event and clk='1' then       -- synchronization whit clock
                        if (valid_bit_A = '1' and valid_bit_B = '1') then -- if the comparisons are correct
                                out_data <= s_in_data_B;-- output correct data
                                valid_bit <= '1' ;               -- valid bit = '1'
                        else
                                out_data <= (others =>'0');      -- output null data
                                valid_bit <= '0' ;       -- valid bit = '0'
                        end if;
        -- end if;
end process Q2_C;                       -- end process Q2_C
end Behavioral;
```

E.2.4 Αρχείο FILE_WRITE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE is
```

```
generic (       write_file:      string  := "OUT.TXT"      );
 port(                      CLK   : in std_logic;
                            RST   : in std_logic;
                            X : in std_logic_vector(23 downto 0);
                            EOG: in std_logic;
                            valid_bit   : in std_logic);
end FILE_WRITE;
architecture Behavioral of FILE_WRITE is
file out_file: TEXT open write_mode is write_file;
signal ok : std_logic :='1';
begin
delay: process (clk)
 begin
            if(clk'event and clk = '1')then
                        if(eog = '1')then
                                    ok <= '0';
                        elsif(eog = '0')then
                                    ok <= '1';
                        end if;
            end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                        if (ok  = '1') and (valid_bit = '1')then
                        --if (valid_bit = '1')then
                                    -- write digital data into log file
                                    write(OUTLINE, str(X));              --add variable to outline variable for output.
                                    writeline(out_file, OUTLINE);      --writes outline variable to file.
                                    --print(l_file, str(x1)& " "& hstr(x2)& "h");
                        end if;
 wait until CLK = '1';
 end loop;
 end process receive_data;
end Behavioral;
```

Ε.3 Κώδικας Επερώτησης Q3

Ε.3.1 Αρχείο TB_FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity TB_FILE_READ is
---port(	--clk		: in  std_logic;
		--rst		: in  std_logic;
		--s_i_dieythinsi:std_logic_vector(3 downto 0)
		--);
end TB_FILE_READ;
architecture Behavioral of TB_FILE_READ is
component FILE_READ
		generic (stim_file:	string  := "R.txt");
		port(
					CLK		: in  std_logic;
					RST		: in  std_logic;
					Y		: out std_logic_vector(23 downto 0);
					EOG		: out std_logic);
end component;
component FILE_WRITE
		generic (	write_file:	string  := "OUT1.TXT");
		port(
					CLK	: in std_logic;
					RST	: in std_logic;
					X_A: in std_logic_vector(47 downto 0);
					EOG: in std_logic;
		valid_bit_A	: in std_logic);
end component;
component FILE_WRITE_B
		generic (	write_file:	string  := "OUT2.TXT");
		port(		CLK	: in std_logic;
					RST	: in std_logic;
					X_B			: in std_logic_vector(47 downto 0);
					EOG		: in std_logic;
		valid_bit_B	: in std_logic);
```

```
end component;
component element_select
        port (      clk     : in  std_logic;                                    -- clock input
                    rst     : in  std_logic;                                    -- reset input
                    in_data   : in  STD_LOGIC_VECTOR (23 downto 0);         -- data input
                    out_data  : out  STD_LOGIC_VECTOR (23 downto 0);    -- data output
                    element     : out  STD_LOGIC_VECTOR (7 downto 0) );  -- element output
end component;
component Q3_select
        port (      clk       : in  std_logic;                  -- clock input
                    rst       : in  std_logic;   -- reset input
                    in_valid_bit : in  std_logic;          -- in valid bit
                    tuple_A_and_B   : in  STD_LOGIC_VECTOR (47 downto 0); -- A data input
                    valid_bit   : out  STD_LOGIC;                    -- valid bit A output
                    out_data    : out  STD_LOGIC_VECTOR (47 downto 0) );   -- data output
end component;
component find_tuple
  Port (        clk : in  STD_LOGIC;
                rst : in  STD_LOGIC;
                A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
                all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
                valid_bit_tuple : out  STD_LOGIC );
end component;
component find_tuple_B
  Port (        clk : in  STD_LOGIC;
                rst : in  STD_LOGIC;
                A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
                all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
                valid_bit_tuple : out  STD_LOGIC );
end component;
signal rst          :std_logic;
signal clk          :std_logic := '1';
signal eog          :std_logic;
signal data_in  :   std_logic_vector(23 downto 0) := (others => '0');
signal data_out_A :  std_logic_vector(47 downto 0) := (others => '0');
signal data_out_B :  std_logic_vector(47 downto 0) := (others => '0');
signal s_tuple_A_B  : std_logic_vector(23 downto 0) := (others => '0');
signal s_tuple_B_B  : std_logic_vector(23 downto 0) := (others => '0');
signal s_all_tuple_A  : std_logic_vector(47 downto 0) := (others => '0');
signal s_all_tuple_B  : std_logic_vector(47 downto 0) := (others => '0');
```

```
signal s_element_A :  std_logic_vector( 7 downto 0) := (others => '0');

signal s_valid_bit_A : std_logic :='0';

signal s_valid_bit_B : std_logic :='0';

signal s_valid_bit_tuple_A : std_logic :='0';

signal s_valid_bit_tuple_B : std_logic :='0';

begin

rst <= '0', '1' after 40 ns, '0' after 100 ns;

clk <= not clk after 10 ns;

input_stim: FILE_READ

        port map(  CLK  => clk,RST  => rst,Y    => data_in,EOG  => eog);

find_A_tuple: find_tuple

        port map(  clk => clk,   rst => rst,   A_tuple     => data_in,B_tuple     => s_tuple_A_B,all_tuple => s_all_tuple_A,valid_bit_tuple=>
s_valid_bit_tuple_A  );

find_B_tuple: find_tuple_B

        port map(  clk => clk,   rst => rst,   A_tuple     => data_in,B_tuple     => s_tuple_B_B,all_tuple => s_all_tuple_B,valid_bit_tuple=>
s_valid_bit_tuple_B  );

select1: Q3_select

        port map(  clk=> clk,rst  => rst, in_valid_bit   => s_valid_bit_tuple_A,tuple_A_and_B  => s_all_tuple_A,
        valid_bit    => s_valid_bit_A,out_data => data_out_A );

select2: Q3_select

        port map(  clk => clk,   rst => rst,   in_valid_bit => s_valid_bit_tuple_B,tuple_A_and_B=> s_all_tuple_B,
        valid_bit   => s_valid_bit_B,out_data     => data_out_B    );

output_stim: FILE_WRITE

        port map(  CLK  => clk,RST  => rst,X_A => data_out_A,    EOG  => eog,valid_bit_A  => s_valid_bit_A);

output_stim_2: FILE_WRITE_B

        port map(  CLK  => clk,RST  => rst,X_B => data_out_B,EOG  => eog, valid_bit_B  => s_valid_bit_B);

end Behavioral;
```

E.3.2 Αρχείο FILE_READ

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use ieee.numeric_std.all;

use std.textio.all;

use work.txt_util.all;

entity FILE_READ is

generic ( stim_file:      string  := "R.txt"  );

 port(

                        CLK           : in  std_logic;

                         RST           : in  std_logic;

                        Y             : out std_logic_vector(23 downto 0)             := (others => '0');
```

```
                        EOG          : out std_logic );
end FILE_READ;
architecture Behavioral of FILE_READ is
function str_to_stdvec(inp: string) return std_logic_vector is
      variable temp: std_logic_vector(inp'range) := (others => '0');
   begin
      for i in inp'range loop
         if (inp(i) = '1') then
            temp(i) := '1';
         elsif (inp(i) = '0') then
           temp(i) := '0';
         end if;
      end loop;
      return temp;
   end function str_to_stdvec;
file stimulus: TEXT open read_mode is stim_file;
begin
-- read data and control information from a file
receive_data: process
                        variable l: line;
                        variable s: string(24 downto 1);
                        begin
                                EOG <= '0';
                                -- wait for Reset to complete
                                wait until RST='1';
                                wait until RST='0';
                                while not endfile(stimulus) loop
                                -- read digital data from input file
                                readline(stimulus, l);
                                read(l, s);
                                --Y <=  str_to_stdvec(s);
                                Y <= to_std_logic_vector(s);
                                wait until CLK = '1';
                                end loop;
                        print("I@FILE_READ: reached end of "& stim_file);
                        EOG <= '1';
                        wait;
 end process receive_data;
end Behavioral;
```

Ε.3.3 Αρχείο find_tuple

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple is
   Port (                  clk : in  STD_LOGIC;
                           rst : in  STD_LOGIC;
                           A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                           B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
                           all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
                           valid_bit_tuple : out  STD_LOGIC  );
end find_tuple;
architecture Behavioral of find_tuple is
component camA
            port (
                           clk: IN std_logic;
                           din: IN std_logic_VECTOR(7 downto 0);
                           busy: OUT std_logic;
                           match: OUT std_logic;
                           match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port (     clk : in  STD_LOGIC;
              rst : in  STD_LOGIC;
              in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
              match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
              match : out  STD_LOGIC  );
end component;
component romA IS
              port (
              clka: IN std_logic;
              addra: IN std_logic_VECTOR(3 downto 0);
              douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        data_in : in  STD_LOGIC_VECTOR (23 downto 0);
        data_out : out  STD_LOGIC_VECTOR (23 downto 0)  );
end component;
```

```
----------signals ------------------------
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1                : std_logic_vector(7 downto 0);    -- the first element A1 of the tuple
signal A2                : std_logic_vector(7 downto 0);                -- the second element A2 of the tuple
signal A3                : std_logic_vector(7 downto 0);                -- the third element A3 of the tuple
signal s_in_data         : STD_LOGIC_VECTOR (23 downto 0); -- the signal from input data
signal tuple             : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
---- signals for delay
signal data_in2 :        std_logic_vector(23 downto 0) := (others => '0');
signal data_in3 :        std_logic_vector(23 downto 0) := (others => '0');
signal data_in4 :        std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);                -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);                -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);                -- initialize third element A3 of the tuple
CAMA1: camA -- find the correct address with element in memory CAM
            port map(  clk=> clk, din=> A2, busy   => s_busy,match   => s_match_CAM,match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,RST  => rst,data_in => s_in_data,data_out => data_in2);
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
            port map(  clk => clk,rst => rst,in_addres => s_cam_address,match_adrress => s_ram_address,match => s_match_find_address);
de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst,data_in => data_in2,data_out => data_in3 );
ROMA1:romA  -- find tuple from ROM
                                    port map ( clka => clk, addra => s_ram_address,douta => tuple);
de3: delay -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst,data_in => data_in3,data_out => data_in4  );
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
            if(clk'event and clk = '1')then
                        valid_bit_tuple <= s_match_find_address;
                        all_tuple <= data_in4 & tuple;
                        B_tuple <= tuple;
            end if;
end process twenty_ns_delay;
```

```
end Behavioral;
```

E.3.4 Αρχείο find_tuple_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple_B is
   Port (      clk : in  STD_LOGIC;
               rst : in  STD_LOGIC;
               A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
               B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
               all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
               valid_bit_tuple : out  STD_LOGIC );
end find_tuple_B;
architecture Behavioral of find_tuple_B is
component camB
               port (       clk: IN std_logic;
                            din: IN std_logic_VECTOR(7 downto 0);
                            busy: OUT std_logic;
                            match: OUT std_logic;
                            match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
          match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
                                        match : out  STD_LOGIC  );
end component;
component romB
               port (       clka: IN std_logic;
                            addra: IN std_logic_VECTOR(3 downto 0);
                            douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (23 downto 0);
          data_out : out  STD_LOGIC_VECTOR (23 downto 0)
                                        );
```

```
end component;
--------------signals ------------------------
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1   :          std_logic_vector(7 downto 0);                    -- the first element A1 of the tuple
signal A2   :          std_logic_vector(7 downto 0);                    -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                             -- the third element A3 of the tuple
signal s_in_data:std_logic_vector(23 downto 0);

signal tuple           : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
---- signals for delay
signal data_in2  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in3  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in4  :     std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);          -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);          -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);          -- initialize third element A3 of the tuple
CAMB1: camB -- find the correct address with element in memory CAM
          port map( clk => clk,din => A2,busy=> s_busy,match=> s_match_CAM,match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,
          RST  => rst,
          data_in => s_in_data,
           data_out => data_in2);
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
port map(  clk => clk,  rst => rst,in_addres => s_cam_address,match_adrress => s_ram_address,match => s_match_find_address  );
de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst,data_in => data_in2, data_out => data_in3 );
ROMB1:romB -- find tuple from ROM
          port map ( clka => clk,addra => s_ram_address,douta => tuple);
de3: delay  -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in3, data_out => data_in4 );
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
          if(clk'event and clk = '1')then
```

```
                valid_bit_tuple <= s_match_find_address;
                all_tuple <= data_in4 & tuple;
                B_tuple <= tuple;
        end if;
end process twenty_ns_delay;
end Behavioral;
```

E.3.5 Αρχείο Q3_select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Q3_select is
Port (  clk        : in  std_logic;                                    -- clock input
        rst        : in  std_logic;                                    -- reset input
        in_valid_bit : in  std_logic;                                  -- in valid bit
        tuple_A_and_B    : in  STD_LOGIC_VECTOR (47 downto 0);         -- A data input
        valid_bit    : out  STD_LOGIC;                                 -- valid bit A output
        out_data     : out  STD_LOGIC_VECTOR (47 downto 0)  );  -- data output
end Q3_select;
architecture Behavioral of Q3_select is
component delay47 is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (47 downto 0);
          data_out : out  STD_LOGIC_VECTOR (47 downto 0));
end component;
-- control signals
signal A1   :        std_logic_vector(7 downto 0);          -- the first element A1 of the tuple
signal A2   :        std_logic_vector(7 downto 0);          -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                 -- the third element A3 of the tuple
signal B1   :        std_logic_vector(7 downto 0);          -- the first element B1 of the tuple
signal B2   :        std_logic_vector(7 downto 0);          -- the second element B2 of the tuple
signal B3   : std_logic_vector(7 downto 0);                 -- the third element B3 of the tuple
signal s_in_data      : STD_LOGIC_VECTOR (47 downto 0);     -- the signal from input data
signal valid_bit_A :  STD_LOGIC :='0';                      -- the first valid bit output
signal valid_bit_B :  STD_LOGIC :='0';                      -- the second valid bit output
signal data_in2 :     std_logic_vector(47 downto 0) := (others => '0');
begin
-- signals  initialization
s_in_data <= tuple_A_and_B;
```

```
A1 <= s_in_data (47 downto 40);                -- initialize first element A1 of the tuple
A2 <= s_in_data (39 downto 32);                -- initialize second element A2 of the tuple
A3 <= s_in_data (31 downto 24);                -- initialize third element A3 of the tuple
B1 <= s_in_data (23 downto 16);                -- initialize first element B1 of the tuple
B2 <= s_in_data (15 downto  8);                -- initialize second element B2 of the tuple
B3 <= s_in_data ( 7 downto  0);                -- initialize third element B3 of the tuple
---------------------------------------------- delay47 ----------------------------------------------------------------------------------------------------------
de47: delay47                          -- delay for synchronization
   port map ( CLK  => clk,RST  => rst,data_in => s_in_data,data_out => data_in2 );
---------------------------------------------- Q3_A -------------------------------------------------------------------------------------------------------------
Q3_A: process (clk)                            -- start process Q3_A
            begin
            if clk'event and clk='1' then                  -- synchronization whit clock
                        if (A1 > "00000010") then          -- the control element A2 > 2
                                    valid_bit_A <= '1' ;   -- valid bit A = '1'
                        else
                                    valid_bit_A <= '0' ;   -- valid bit = '0'
                                    end if;
            end if;
end process Q3_A;                              -- end process Q3_A
---------------------------------------------- Q3_B -------------------------------------------------------------------------------------------------------------
Q3_B: process (clk)                                                    -- start process Q3_B
            begin
            if (clk'event and clk='1') then    -- synchronization whit clock
                        if (A2 = B2) then              -- the control element A2 = B2
                                    valid_bit_B <= '1' ;   -- valid bit B = '1'
                        else
                                    valid_bit_B <= '0' ;   -- valid bit B = '0'
                                    end if;
            end if;
end process Q3_B;                              -- end process Q3_B
---------------------------------------------- Q3_C -------------------------------------------------------------------------------------------------------------
Q3_C:process (clk,valid_bit_A,valid_bit_B)   -- start process Q3_C
            begin
                        if clk'event and clk='1' then                                                                                                  --
synchronization whit clock
                                    if (valid_bit_A = '1' and valid_bit_B = '1') then          -- if the comparisons are correct
                                                out_data <= data_in2;          -- output correct data
                                                valid_bit <= '1' ;         -- valid bit = '1'
                                    else
                                                out_data <= (others =>'0');         -- output null data
                                                valid_bit <= '0' ;         -- valid bit = '0'
                                    end if;
```

```
                end if;
end process Q3_C;                                   -- end process Q3_C
end Behavioral;
```

E.3.6 Αρχείο FILE_WRITE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE is
generic (write_file:      string  := "OUT1.TXT");
 port(       CLK   : in std_logic;
             RST   : in std_logic;
             X_A          : in std_logic_vector(47 downto 0);
             EOG          : in std_logic;
             valid_bit_A   : in std_logic);
end FILE_WRITE;
architecture Behavioral of FILE_WRITE is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 3 :=0;
signal ok_A : std_logic;
begin
delay: process (clk)
 begin
             if(clk'event and clk = '1')then
                          if(eog = '1')then
                                       if count =3 then
                                                    ok_A <= '0';
                                       else
                                                    count <= count +1;
                                                    ok_A  <= '1';
                                       end if;
                          elsif(eog = '0')then
                                       ok_A <= '1';
                                       count <= 0;
                          end if;
             end if;
end process delay;
receive_data: process
```

```
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                        if (ok_A  = '1') and (valid_bit_A = '1')then
                        --if (valid_bit = '1')then
                                -- write digital data into log file
                                write(OUTLINE, str(X_A));          --add variable to outline variable for output.
                                writeline(out_file, OUTLINE);      --writes outline variable to file.
                                --print(l_file, str(x1)& " "& hstr(x2)& "h");
                        end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.3.7 Αρχείο FILE_WRITE_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE_B is
generic (  write_file:      string  := "OUT2.TXT" );
  port(       CLK   : in std_logic;
              RST   : in std_logic;
              X_B           : in std_logic_vector(47 downto 0);
              EOG           : in std_logic;
              valid_bit_B   : in std_logic);
end FILE_WRITE_B;
architecture Behavioral of FILE_WRITE_B is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 10 :=0;
signal ok_B : std_logic;
 begin
delay: process (clk)
 begin
              if(clk'event and clk = '1')then
                        if(eog = '1')then
```

```
                        if count =3 then
                                ok_B <= '0';
                        else
                                count <= count +1;
                                ok_B  <= '1';
                        end if;
                elsif(eog = '0')then
                        ok_B <= '1';
                        count <= 0;
                end if;
        end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                if (ok_B  = '1') and (valid_bit_B = '1')then
                --if (valid_bit = '1')then
                        -- write digital data into log file
                        write(OUTLINE, str(X_B));          --add variable to outline variable for output.
                        writeline(out_file, OUTLINE);      --writes outline variable to file.
                        --print(l_file, str(x1)& " "& hstr(x2)& "h");
                end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.4 Κώδικας Επερώτησης Q4

E.4.1 Αρχείο TB_FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
```

```
entity TB_FILE_READ is
---port(    --clk          : in  std_logic;
            --rst          : in  std_logic;
            --s_i_dieythinsi:std_logic_vector(3 downto 0)
            --);
end TB_FILE_READ;
architecture Behavioral of TB_FILE_READ is
component FILE_READ
            generic (stim_file:     string  := "R.txt");
            port(       CLK          : in  std_logic;
                        RST          : in  std_logic;
                        Y           : out std_logic_vector(23 downto 0);
                        EOG          : out std_logic);
end component;
component FILE_WRITE
            generic (   write_file:     string  := "OUT1.TXT");
            port(       CLK   : in std_logic;
                        RST   : in std_logic;
                        X_A          : in std_logic_vector(39 downto 0);
                        EOG          : in std_logic;
                        valid_bit_A : in std_logic);
end component;
component FILE_WRITE_B
            generic (write_file:     string  := "OUT2.TXT" );
            port(       CLK   : in std_logic;
                        RST   : in std_logic;
                        X_B          : in std_logic_vector(39 downto 0);
                        EOG          : in std_logic;
                        valid_bit_B : in std_logic);
end component;
component FILE_WRITE_C
            generic (write_file:     string  := "OUT3.TXT");
            port(       CLK   : in std_logic;
                        RST   : in std_logic;
                        X_C          : in std_logic_vector(39 downto 0);
                        EOG          : in std_logic;
                        valid_bit_C : in std_logic);
end component;
component FILE_WRITE_D
            generic (   write_file:     string  := "OUT4.TXT");
            port(       CLK   : in std_logic;
                        RST   : in std_logic;
```

```
                        X_D          : in std_logic_vector(39 downto 0);
                        EOG          : in std_logic;
                        valid_bit_D : in std_logic);
end component;
component Q4_select
            port (      clk        : in  std_logic;                                    -- clock input
                        rst        : in  std_logic;                                    -- reset input
                        in_valid_bit : in  std_logic;                                  -- in valid bit
                        tuple_A_and_B    : in  STD_LOGIC_VECTOR (47 downto 0);         -- A data input
                        valid_bit    : out  STD_LOGIC;                                 -- valid bit A output
                        out_data     : out  STD_LOGIC_VECTOR (39 downto 0) );  -- data output
end component;
component find_tuple
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
              B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
        all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
        valid_bit_tuple : out  STD_LOGIC );
end component;
component find_tuple_B
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                                    B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
        all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
        valid_bit_tuple : out  STD_LOGIC );
end component;
component find_tuple_C
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
 B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
        all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
        valid_bit_tuple : out  STD_LOGIC );
end component;
component find_tuple_D
   Port ( clk : in  STD_LOGIC;
        rst : in  STD_LOGIC;
        A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
             B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
        all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
```

```
        valid_bit_tuple : out  STD_LOGIC );
end component;
signal rst                  :std_logic;
signal clk                  :std_logic := '1';
signal eog                  :std_logic;
signal data_in  :           std_logic_vector(23 downto 0) := (others => '0');
signal data_out_A :         std_logic_vector(39 downto 0) := (others => '0');
signal data_out_B :         std_logic_vector(39 downto 0) := (others => '0');
signal data_out_C :         std_logic_vector(39 downto 0) := (others => '0');
signal data_out_D :         std_logic_vector(39 downto 0) := (others => '0');
signal s_tuple_A_B  :       std_logic_vector(23 downto 0) := (others => '0');
signal s_tuple_B_B  :       std_logic_vector(23 downto 0) := (others => '0');
signal s_tuple_C_B  :       std_logic_vector(23 downto 0) := (others => '0');
signal s_tuple_D_B  :       std_logic_vector(23 downto 0) := (others => '0');
signal s_all_tuple_A  :     std_logic_vector(47 downto 0) := (others => '0');
signal s_all_tuple_B  :     std_logic_vector(47 downto 0) := (others => '0');
signal s_all_tuple_C  :     std_logic_vector(47 downto 0) := (others => '0');
signal s_all_tuple_D  :     std_logic_vector(47 downto 0) := (others => '0');
signal s_element_A :        std_logic_vector( 7 downto 0) := (others => '0');
signal s_valid_bit_A : std_logic :='0';
signal s_valid_bit_B : std_logic :='0';
signal s_valid_bit_C : std_logic :='0';
signal s_valid_bit_D : std_logic :='0';
signal s_valid_bit_tuple_A : std_logic :='0';
signal s_valid_bit_tuple_B : std_logic :='0';
signal s_valid_bit_tuple_C : std_logic :='0';
signal s_valid_bit_tuple_D : std_logic :='0';
begin
rst <= '0', '1' after 40 ns, '0' after 100 ns;
clk <= not clk after 10 ns;
input_stim: FILE_READ
            port map(CLK  => clk,RST  => rst,Y=> data_in,EOG  => eog);
find_A_tuple: find_tuple
            port map(  clk => clk,   rst => rst,   A_tuple => data_in,B_tuple => s_tuple_A_B,all_tuple => s_all_tuple_A,
                        valid_bit_tuple=> s_valid_bit_tuple_A  );
find_B_tuple: find_tuple_B
            port map(  clk => clk,   rst => rst,   A_tuple => data_in,B_tuple => s_tuple_B_B, all_tuple => s_all_tuple_B,
                        valid_bit_tuple=> s_valid_bit_tuple_B  );

find_C_tuple: find_tuple_C
            port map(  clk => clk,   rst => rst,A_tuple => data_in,B_tuple => s_tuple_C_B,all_tuple => s_all_tuple_C,
                        valid_bit_tuple=> s_valid_bit_tuple_C  );
```

```
find_D_tuple: find_tuple_D
        port map(  clk => clk,rst => rst,A_tuple => data_in,B_tuple => s_tuple_D_B,all_tuple => s_all_tuple_D,valid_bit_tuple=>
s_valid_bit_tuple_D  );
select1: Q4_select
        port map(  clk  => clk,  rst => rst,in_valid_bit => s_valid_bit_tuple_A,tuple_A_and_B  => s_all_tuple_A,
                        valid_bit    => s_valid_bit_A,out_data     => data_out_A );
select2: Q4_select
        port map(  clk => clk,rst => rst,in_valid_bit => s_valid_bit_tuple_B, tuple_A_and_B=> s_all_tuple_B,
                valid_bit    => s_valid_bit_B,out_data     => data_out_B   );
select3: Q4_select
        port map(  clk  => clk,  rst => rst,   in_valid_bit => s_valid_bit_tuple_C,tuple_A_and_B  => s_all_tuple_C,
                        valid_bit    => s_valid_bit_C,out_data     => data_out_C    );
select4: Q4_select
        port map(  clk   => clk, rst  => rst,  in_valid_bit => s_valid_bit_tuple_D,tuple_A_and_B=> s_all_tuple_D,
                valid_bit    => s_valid_bit_D,out_data   => data_out_D );
output_stim: FILE_WRITE
        port map(  CLK  => clk,RST  => rst,X_A  => data_out_A,EOG  => eog, valid_bit_A  => s_valid_bit_A);
output_stim_2: FILE_WRITE_B
        port map(  CLK  => clk,RST  => rst,X_B  => data_out_B,EOG  => eog,valid_bit_B => s_valid_bit_B);
output_stim_3: FILE_WRITE_C
        port map(  CLK  => clk,RST  => rst,X_C  => data_out_C,EOG  => eog,  valid_bit_C => s_valid_bit_C);
output_stim_4: FILE_WRITE_D
        port map(  CLK  => clk,RST  => rst,X_D  => data_out_D,EOG  => eog,valid_bit_D  => s_valid_bit_D);
end Behavioral;
```

E.4.2 Αρχείο FILE_READ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
use std.textio.all;
use work.txt_util.all;
entity FILE_READ is
generic (  stim_file:      string  := "R.txt"    );
 port(
    CLK           : in  std_logic;
    RST           : in  std_logic;
    Y             : out std_logic_vector(23 downto 0)        := (others => '0');
    EOG           : out std_logic   );
end FILE_READ;
```

```
architecture Behavioral of FILE_READ is
function str_to_stdvec(inp: string) return std_logic_vector is
      variable temp: std_logic_vector(inp'range) := (others => '0');
   begin
      for i in inp'range loop
        if (inp(i) = '1') then
          temp(i) := '1';
        elsif (inp(i) = '0') then
          temp(i) := '0';
        end if;
      end loop;
      return temp;
   end function str_to_stdvec;
file stimulus: TEXT open read_mode is stim_file;
begin
-- read data and control information from a file
receive_data: process
                        variable l: line;
                        variable s: string(24 downto 1);
                        begin
                                EOG <= '0';
                                -- wait for Reset to complete
                                wait until RST='1';
                                wait until RST='0';
                                while not endfile(stimulus) loop
                                -- read digital data from input file
                                readline(stimulus, l);
                                read(l, s);
                                --Y <=  str_to_stdvec(s);
                                Y <= to_std_logic_vector(s);
                                wait until CLK = '1';
                                end loop;
                        print("l@FILE_READ: reached end of "& stim_file);
                        EOG <= '1';
                        wait;
 end process receive_data;
end Behavioral;
```

E.4.3 Αρχείο find_tuple

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
         B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
          all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
          valid_bit_tuple : out  STD_LOGIC );
end find_tuple;
architecture Behavioral of find_tuple is
component camA
                port (          clk: IN std_logic;
                                din: IN std_logic_VECTOR(7 downto 0);
                                busy: OUT std_logic;
                                match: OUT std_logic;
                                match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
          match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
                 match : out  STD_LOGIC  );
end component;
component romA IS
                port (          clka: IN std_logic;
                                addra: IN std_logic_VECTOR(3 downto 0);
                                douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (23 downto 0);
          data_out : out  STD_LOGIC_VECTOR (23 downto 0)  );
end component;
----------signals -------------------------
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
```

```
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1                : std_logic_vector(7 downto 0);    -- the first element A1 of the tuple
signal A2                : std_logic_vector(7 downto 0);                -- the second element A2 of the tuple
signal A3                : std_logic_vector(7 downto 0);                -- the third element A3 of the tuple
signal s_in_data         : STD_LOGIC_VECTOR (23 downto 0); -- the signal from input data
signal tuple             : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
---- signals for delay
signal data_in2  :       std_logic_vector(23 downto 0) := (others => '0');
signal data_in3  :       std_logic_vector(23 downto 0) := (others => '0');
signal data_in4  :       std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);              -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);              -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);              -- initialize third element A3 of the tuple
CAMA1: camA -- find the correct address with element in memory CAM
            port map(  clk  => clk,din => A1, busy   => s_busy, match=> s_match_CAM, match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,RST  => rst, data_in => s_in_data, data_out => data_in2 );
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
            port map(  clk => clk,rst => rst,in_addres => s_cam_address,match_adrress => s_ram_address,match => s_match_find_address );
de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst,data_in => data_in2, data_out => data_in3  );
ROMA1:romA  -- find tuple from ROM
            port map ( clka => clk, addra => s_ram_address,douta    => tuple);
de3: delay -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in3, data_out => data_in4);
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
            if(clk'event and clk = '1')then
                        valid_bit_tuple <= s_match_find_address;
                        all_tuple <= data_in4 & tuple;
                        B_tuple <= tuple;
            end if;
end process twenty_ns_delay;
end Behavioral;
```

Ε.4.4 Αρχείο find_tuple_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple_B is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
          B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
          all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
          valid_bit_tuple : out  STD_LOGIC );
end find_tuple_B;
architecture Behavioral of find_tuple_B is
component camB
                port (          clk: IN std_logic;
                                din: IN std_logic_VECTOR(7 downto 0);
                                busy: OUT std_logic;
                                match: OUT std_logic;
                                match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
          match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
                 match : out  STD_LOGIC  );
end component;
component romB
                port (          clka: IN std_logic;
                                addra: IN std_logic_VECTOR(3 downto 0);
                                douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (23 downto 0);
          data_out : out  STD_LOGIC_VECTOR (23 downto 0)  );
end component;
----------signals -------------------------
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
```

```
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1   :          std_logic_vector(7 downto 0);                    -- the first element A1 of the tuple
signal A2   :          std_logic_vector(7 downto 0);                              -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                             -- the third element A3 of the tuple
signal s_in_data:std_logic_vector(23 downto 0);
signal tuple          : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
---- signals for delay
signal data_in2  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in3  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in4  :     std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);          -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);          -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);          -- initialize third element A3 of the tuple
CAMB1: camB -- find the correct address with element in memory CAM
        port map( clk  => clk, din => A1, busy=> s_busy,   match => s_match_CAM,match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,RST  => rst, data_in => s_in_data,data_out => data_in2 );
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
        port map(  clk => clk,   rst => rst,   in_addres => s_cam_address,match_adrress => s_ram_address,match =>
s_match_find_address );
de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in2, data_out => data_in3 );
ROMB1:romB -- find tuple from ROM
        port map ( clka => clk, addra => s_ram_address,douta=> tuple);
de3: delay  -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in3, data_out => data_in4 );
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
        if(clk'event and clk = '1')then
                valid_bit_tuple <= s_match_find_address;
                all_tuple <= data_in4 & tuple;
                B_tuple <= tuple;
        end if;
end process twenty_ns_delay;
end Behavioral;
```

E.4.5 Αρχείο find_tuple_C

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple_C is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                                        B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
          all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
          valid_bit_tuple : out  STD_LOGIC );
end find_tuple_C;
architecture Behavioral of find_tuple_C is
component camC
                port (          clk: IN std_logic;
                                din: IN std_logic_VECTOR(7 downto 0);
                                busy: OUT std_logic;
                                match: OUT std_logic;
                                match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
          match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
                                        match : out  STD_LOGIC );
end component;
component romC
                port (          clka: IN std_logic;
                                addra: IN std_logic_VECTOR(3 downto 0);
                                douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (23 downto 0);
          data_out : out  STD_LOGIC_VECTOR (23 downto 0)  );
end component;

----------signals ------------------------
```

```
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1   :          std_logic_vector(7 downto 0);                    -- the first element A1 of the tuple
signal A2   :          std_logic_vector(7 downto 0);                              -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                             -- the third element A3 of the tuple
signal s_in_data:std_logic_vector(23 downto 0);
signal tuple           : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
---- signals for delay
signal data_in2  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in3  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in4  :     std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);          -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);          -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);          -- initialize third element A3 of the tuple
CAMC1: camC -- find the correct address with element in memory CAM
        port map(clk  => clk,din => A1,busy   => s_busy,atch=> s_match_CAM,match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,RST  => rst, data_in => s_in_data,  data_out => data_in2 );
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
        port map(  clk => clk,rst => rst,in_addres => s_cam_address,match_adrress => s_ram_address,match => s_match_find_address );

de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in2,  data_out => data_in3 );
ROMC1:romC -- find tuple from ROM
        port map ( clka => clk, addra => s_ram_address,douta => tuple);
de3: delay  -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in3,  data_out => data_in4 );
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
        if(clk'event and clk = '1')then
                valid_bit_tuple <= s_match_find_address;
                all_tuple <= data_in4 & tuple;
                B_tuple <= tuple;
        end if;
end process twenty_ns_delay;
end Behavioral;
```

E.4.6 Αρχείο find_tuple_D

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity find_tuple_D is
   Port (                  clk : in  STD_LOGIC;
                            rst : in  STD_LOGIC;
                            A_tuple : in  STD_LOGIC_VECTOR (23 downto 0);
                           B_tuple : out  STD_LOGIC_VECTOR (23 downto 0);
                           all_tuple : out  STD_LOGIC_VECTOR (47 downto 0);
                           valid_bit_tuple : out  STD_LOGIC
               );
end find_tuple_D;
architecture Behavioral of find_tuple_D is
component camD
               port (      clk: IN std_logic;
                           din: IN std_logic_VECTOR(7 downto 0);
                           busy: OUT std_logic;
                           match: OUT std_logic;
                           match_addr: OUT std_logic_VECTOR(15 downto 0));
end component;
component find_address_from_CAM
   port (                  clk : in  STD_LOGIC;
                           rst : in  STD_LOGIC;
                            in_addres : in  STD_LOGIC_VECTOR (15 downto 0);
                           match_adrress : out  STD_LOGIC_VECTOR (3 downto 0);
                             match : out  STD_LOGIC  );
end component;
component romD
               port (
                           clka: IN std_logic;
                           addra: IN std_logic_VECTOR(3 downto 0);
                           douta: OUT std_logic_VECTOR(23 downto 0));
end component;
component delay
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          data_in : in  STD_LOGIC_VECTOR (23 downto 0);
          data_out : out  STD_LOGIC_VECTOR (23 downto 0) );
end component;
```

```
----------signals ------------------------
signal s_busy: std_logic;
signal s_match_CAM : std_logic;
signal s_match_find_address : std_logic;
signal s_cam_address:std_logic_vector(15 downto 0);
signal s_ram_address:std_logic_vector(3 downto 0);
signal s_element:std_logic_vector(7 downto 0);
--  signals element from A tuple
signal A1   :          std_logic_vector(7 downto 0);          -- the first element A1 of the tuple
signal A2   :          std_logic_vector(7 downto 0);          -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                   -- the third element A3 of the tuple
signal s_in_data:std_logic_vector(23 downto 0);
signal tuple           : STD_LOGIC_VECTOR (23 downto 0); -- the signal from tuple data
signal data_in2  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in3  :     std_logic_vector(23 downto 0) := (others => '0');
signal data_in4  :     std_logic_vector(23 downto 0) := (others => '0');
begin
s_in_data <= A_tuple;
A1 <= s_in_data (23 downto 16);          -- initialize first element A1 of the tuple
A2 <= s_in_data (15 downto  8);          -- initialize second element A2 of the tuple
A3 <= s_in_data ( 7 downto  0);          -- initialize third element A3 of the tuple
CAMD1: camD -- find the correct address with element in memory CAM
        port map(clk  => clk, din => A1,busy=> s_busy,match=> s_match_CAM,match_addr  => s_cam_address);
de: delay -- FIRST DELAY
   port map ( CLK  => clk,RST  => rst, data_in => s_in_data, data_out => data_in2 );
FIND: find_address_from_CAM -- code the address from CAM memory for ROM memory
        port map(  clk => clk,rst => rst,in_addres => s_cam_address,match_adrress => s_ram_address,match => s_match_find_address );
de2: delay -- SECOND DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in2, data_out => data_in3 );
ROMD1:romD -- find tuple from ROM
port map ( clka => clk, addra => s_ram_address,douta => tuple);
de3: delay  -- THIRD DELAY
   port map ( CLK  => clk,RST  => rst, data_in => data_in3, data_out => data_in4  );
twenty_ns_delay: process (clk) -- Turns out  the correct tuple and the valid bit
 begin
        if(clk'event and clk = '1')then
                valid_bit_tuple <= s_match_find_address;
                all_tuple <= data_in4 & tuple;
                B_tuple <= tuple;
        end if;
end process twenty_ns_delay;
end Behavioral;
```

E.4.7 Αρχείο Q4_select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Q4_select is
Port (      clk        : in  std_logic;                          -- clock input
            rst        : in  std_logic;                          -- reset input
            in_valid_bit : in  std_logic;                        -- in valid bit
            tuple_A_and_B   : in  STD_LOGIC_VECTOR (47 downto 0);          -- A data input
            valid_bit   : out  STD_LOGIC;                                  -- valid bit A output
            out_data    : out  STD_LOGIC_VECTOR (39 downto 0) );   -- data output
end Q4_select;
architecture Behavioral of Q4_select is
-- control signals
signal A1   :           std_logic_vector(7 downto 0);                  -- the first element A1 of the tuple
signal A2   :           std_logic_vector(7 downto 0);                  -- the second element A2 of the tuple
signal A3   : std_logic_vector(7 downto 0);                            -- the third element A3 of the tuple
signal B1   :           std_logic_vector(7 downto 0);                  -- the first element B1 of the tuple
signal B2   :           std_logic_vector(7 downto 0);                  -- the second element B2 of the tuple
signal B3   : std_logic_vector(7 downto 0);                            -- the third element B3 of the tuple
signal s_in_data        : STD_LOGIC_VECTOR (47 downto 0);          -- the signal from input data
signal data_out_A1_A2_A3_B2_B3 :        std_logic_vector(39 downto 0) := (others => '0');
begin
-- signals  initialization
s_in_data <= tuple_A_and_B;
A1 <= s_in_data (47 downto 40);             -- initialize first element A1 of the tuple
A2 <= s_in_data (39 downto  32);            -- initialize second element A2 of the tuple
A3 <= s_in_data (31 downto  24);            -- initialize third element A3 of the tuple
B1 <= s_in_data (23 downto 16);             -- initialize first element B1 of the tuple
B2 <= s_in_data (15 downto  8);             -- initialize second element B2 of the tuple
B3 <= s_in_data ( 7 downto  0);             -- initialize third element B3 of the tuple
-- initialize the correct data A1, A2, A3, B2, B3
data_out_A1_A2_A3_B2_B3 <= A1 & A2 & A3 & B2 & B3;
------------------------------------------------ Q4-----------------------------------------------------------------------------------------------------------
Q4: process (clk)                                                      -- start process Q4
            begin
            if (clk'event and clk='1') then                 -- synchronization whit clock
                        if (A1 = B1) then                                      -- the control element A1 = B1
                                    out_data <= data_out_A1_A2_A3_B2_B3;   -- output correct data
                                                valid_bit <= '1' ;         -- valid bit = '1'
```

```
                else
                        out_data <= (others =>'0');         -- output null data
                                valid_bit <= '0' ;          -- valid bit = '0'
                        end if;
        end if;
end process Q4;         -- end process Q4
end Behavioral;
```

E.4.8 Αρχείο FILE_WRITE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE is
generic (   write_file:      string  := "OUT1.TXT" );
  port(       CLK   : in std_logic;
              RST   : in std_logic;
              X_A   : in std_logic_vector(39 downto 0);
              EOG         : in std_logic;
              valid_bit_A : in std_logic);
end FILE_WRITE;
architecture Behavioral of FILE_WRITE is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 5 :=0;
signal ok_A : std_logic;
begin
delay: process (clk)
 begin
        if(clk'event and clk = '1')then
                if(eog = '1')then
                        if count =4 then
                                ok_A <= '0';
                        else
                                count <= count +1;
                                ok_A  <= '1';
                        end if;
                elsif(eog = '0')then
                        ok_A <= '1';
                        count <= 0;
```

```
                        end if;
                end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                        if (ok_A  = '1') and (valid_bit_A = '1')then
                        --if (valid_bit = '1')then
                                -- write digital data into log file
                                write(OUTLINE, str(X_A));          --add variable to outline variable for output.
                                writeline(out_file, OUTLINE);      --writes outline variable to file.
                                --print(l_file, str(x1)& " "& hstr(x2)& "h");
                        end if;

    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.4.9 Αρχείο FILE_WRITE_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE_B is
generic ( write_file:      string  := "OUT2.TXT" );
  port(       CLK   : in std_logic;
              RST   : in std_logic;
              X_B           : in std_logic_vector(39 downto 0);
              EOG           : in std_logic;
              valid_bit_B : in std_logic);
end FILE_WRITE_B;
architecture Behavioral of FILE_WRITE_B is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 5 :=0;
signal ok_B : std_logic;
```

```
begin
delay: process (clk)
 begin
        if(clk'event and clk = '1')then
                if(eog = '1')then
                        if count =4 then
                                ok_B <= '0';
                        else
                                count <= count +1;
                                ok_B  <= '1';
                        end if;
                elsif(eog = '0')then
                        ok_B <= '1';
                        count <= 0;
                end if;
        end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                if (ok_B  = '1') and (valid_bit_B = '1')then
                --if (valid_bit = '1')then
                        -- write digital data into log file
                        write(OUTLINE, str(X_B));                       --add variable to outline variable for output.
                        writeline(out_file, OUTLINE);       --writes outline variable to file.
                        --print(l_file, str(x1)& " "& hstr(x2)& "h");
                end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.4.10 Αρχείο FILE_WRITE_C

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
```

```
use work.txt_util.all;
entity FILE_WRITE_C is
generic (  write_file:      string  := "OUT3.TXT"  );
 port(       CLK   : in std_logic;
             RST   : in std_logic;
             X_C   : in std_logic_vector(39 downto 0);
             EOG          : in std_logic;
             valid_bit_C : in std_logic);
end FILE_WRITE_C;
architecture Behavioral of FILE_WRITE_C is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 5 :=0;
signal ok_C : std_logic;
begin
delay: process (clk)
 begin
             if(clk'event and clk = '1')then
                          if(eog = '1')then
                                       if count =4 then
                                                    ok_C <= '0';
                                       else
                                                    count <= count +1;
                                                    ok_C  <= '1';
                                       end if;
                          elsif(eog = '0')then
                                       ok_C <= '1';
                                       count <= 0;
                          end if;
             end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                          if (ok_C  = '1') and (valid_bit_C = '1')then
                          --if (valid_bit = '1')then
                                       -- write digital data into log file
                                       write(OUTLINE, str(X_C));          --add variable to outline variable for output.
                                       writeline(out_file, OUTLINE);      --writes outline variable to file.
                                       --print(l_file, str(x1)& " "& hstr(x2)& "h");
```

```
                        end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

E.4.11 Αρχείο FILE_WRITE_D

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE_D is
generic (write_file:     string  := "OUT4.TXT" );
  port(                  CLK   : in std_logic;
                         RST   : in std_logic;
                         X_D           : in std_logic_vector(39 downto 0);
                         EOG           : in std_logic;
                         valid_bit_D : in std_logic);
end FILE_WRITE_D;
architecture Behavioral of FILE_WRITE_D is
file out_file: TEXT open write_mode is write_file;
signal count   : integer range 0 to 5 :=0;
signal ok_D : std_logic;
begin
delay: process (clk)
 begin
            if(clk'event and clk = '1')then
                        if(eog = '1')then
                                    if count =4 then
                                                ok_D <= '0';
                                    else
                                                count <= count +1;
                                                ok_D  <= '1';
                                    end if;
                        elsif(eog = '0')then
                                    ok_D <= '1';
                                    count <= 0;
                        end if;
            end if;
```

```
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                        if (ok_D  = '1') and (valid_bit_D = '1')then
                        --if (valid_bit = '1')then
                                -- write digital data into log file
                                write(OUTLINE, str(X_D));          --add variable to outline variable for output.
                                writeline(out_file, OUTLINE);      --writes outline variable to file.
                                --print(l_file, str(x1)& " "& hstr(x2)& "h");
                        end if;
    wait until CLK = '1';
  end loop;
 end process receive_data;
end Behavioral;
```

Παράρτημα Z

Βοηθητικοί Κώδικες Εκτέλεσης και Προσομοίωσης

Z.1 Ανάγνωση Εγγραφή Αρχείων *.txt Κατά την Προσομοίωση

Z.1.1 Αρχείο std.textio.all

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use std.textio.all;
package txt_util is
   -- prints a message to the screen
   procedure print(text: string);
   -- prints the message when active
   -- useful for debug switches
   procedure print(active: boolean; text: string);
   -- converts std_logic into a character
   function chr(sl: std_logic) return character;
```

```
    -- converts std_logic into a string (1 to 1)
    function str(sl: std_logic) return string;
    -- converts std_logic_vector into a string (binary base)
    function str(slv: std_logic_vector) return string;
    -- converts boolean into a string
    function str(b: boolean) return string;
    -- converts an integer into a single character
    -- (can also be used for hex conversion and other bases)
    function chr(int: integer) return character;
    -- converts integer into string using specified base
    function str(int: integer; base: integer) return string;
    -- converts integer to string, using base 10
    function str(int: integer) return string;
    -- convert std_logic_vector into a string in hex format
    function hstr(slv: std_logic_vector) return string;
    -- functions to manipulate strings
    ------------------------------------
    -- convert a character to upper case
    function to_upper(c: character) return character;
    -- convert a character to lower case
    function to_lower(c: character) return character;
    -- convert a string to upper case
    function to_upper(s: string) return string;
    -- convert a string to lower case
    function to_lower(s: string) return string;
    -- functions to convert strings into other formats
    ------------------------------------------------
    -- converts a character into std_logic
    function to_std_logic(c: character) return std_logic;
    -- converts a string into std_logic_vector
    function to_std_logic_vector(s: string) return std_logic_vector;
    -- file I/O
    -----------
    -- read variable length string from input file
    procedure str_read(file in_file: TEXT; res_string: out string);
    -- print string to a file and start new line
    procedure print(file out_file: TEXT;  new_string: in  string);
    -- print character to a file and start new line
    procedure print(file out_file: TEXT;  char:  in  character);
end txt_util;
package body txt_util is
```

```
-- prints text to the screen
procedure print(text: string) is
 variable msg_line: line;
 begin
  write(msg_line, text);
  writeline(output, msg_line);
end print;
-- prints text to the screen when active
procedure print(active: boolean; text: string)  is
 begin
  if active then
    print(text);
  end if;
end print;
-- converts std_logic into a character
function chr(sl: std_logic) return character is
 variable c: character;
 begin
  case sl is
    when 'U' => c:= 'U';
    when 'X' => c:= 'X';
    when '0' => c:= '0';
    when '1' => c:= '1';
    when 'Z' => c:= 'Z';
    when 'W' => c:= 'W';
    when 'L' => c:= 'L';
    when 'H' => c:= 'H';
    when '-' => c:= '-';
  end case;
 return c;
end chr;
-- converts std_logic into a string (1 to 1)
function str(sl: std_logic) return string is
 variable s: string(1 to 1);
 begin
   s(1) := chr(sl);
   return s;
end str;
-- converts std_logic_vector into a string (binary base)
-- (this also takes care of the fact that the range of
--  a string is natural while a std_logic_vector may
--  have an integer range)
```

```
function str(slv: std_logic_vector) return string is
  variable result : string (1 to slv'length);
  variable r : integer;
begin
  r := 1;
  for i in slv'range loop
    result(r) := chr(slv(i));
    r := r + 1;
  end loop;
  return result;
end str;
function str(b: boolean) return string is
 begin
   if b then
     return "true";
  else
    return "false";
   end if;
 end str;
-- converts an integer into a character
-- for 0 to 9 the obvious mapping is used, higher
-- values are mapped to the characters A-Z
-- (this is usefull for systems with base > 10)
-- (adapted from Steve Vogwell's posting in comp.lang.vhdl)
function chr(int: integer) return character is
 variable c: character;
begin
    case int is
      when  0 => c := '0';
      when  1 => c := '1';
      when  2 => c := '2';
      when  3 => c := '3';
      when  4 => c := '4';
      when  5 => c := '5';
      when  6 => c := '6';
      when  7 => c := '7';
      when  8 => c := '8';
      when  9 => c := '9';
      when 10 => c := 'A';
      when 11 => c := 'B';
      when 12 => c := 'C';
      when 13 => c := 'D';
```

```
      when 14 => c := 'E';
      when 15 => c := 'F';
      when 16 => c := 'G';
      when 17 => c := 'H';
      when 18 => c := 'I';
      when 19 => c := 'J';
      when 20 => c := 'K';
      when 21 => c := 'L';
      when 22 => c := 'M';
      when 23 => c := 'N';
      when 24 => c := 'O';
      when 25 => c := 'P';
      when 26 => c := 'Q';
      when 27 => c := 'R';
      when 28 => c := 'S';
      when 29 => c := 'T';
      when 30 => c := 'U';
      when 31 => c := 'V';
      when 32 => c := 'W';
      when 33 => c := 'X';
      when 34 => c := 'Y';
      when 35 => c := 'Z';
      when others => c := '?';
    end case;
    return c;
 end chr;
-- convert integer to string using specified base
-- (adapted from Steve Vogwell's posting in comp.lang.vhdl)
function str(int: integer; base: integer) return string is
 variable temp:     string(1 to 10);
 variable num:      integer;
 variable abs_int:  integer;
 variable len:      integer := 1;
 variable power:    integer := 1;
begin
 -- bug fix for negative numbers
 abs_int := abs(int);
 num     := abs_int;
 while num >= base loop                -- Determine how many
  len := len + 1;                      -- characters required
  num := num / base;                   -- to represent the
 end loop ;                            -- number.
```

```
  for i in len downto 1 loop              -- Convert the number to
    temp(i) := chr(abs_int/power mod base);  -- a string starting
    power := power * base;                -- with the right hand
  end loop ;                              -- side.
  -- return result and add sign if required
  if int < 0 then
    return '-'& temp(1 to len);
   else
    return temp(1 to len);
  end if;
 end str;
-- convert integer to string, using base 10
function str(int: integer) return string is
 begin
  return str(int, 10) ;
 end str;
 -- converts a std_logic_vector into a hex string.
 function hstr(slv: std_logic_vector) return string is
    variable hexlen: integer;
    variable longslv : std_logic_vector(67 downto 0) := (others => '0');
    variable hex : string(1 to 16);
    variable fourbit : std_logic_vector(3 downto 0);
  begin
    hexlen := (slv'left+1)/4;
    if (slv'left+1) mod 4 /= 0 then
      hexlen := hexlen + 1;
    end if;
    longslv(slv'left downto 0) := slv;
    for i in (hexlen -1) downto 0 loop
      fourbit := longslv(((i*4)+3) downto (i*4));
      case fourbit is
        when "0000" => hex(hexlen -I) := '0';
        when "0001" => hex(hexlen -I) := '1';
        when "0010" => hex(hexlen -I) := '2';
        when "0011" => hex(hexlen -I) := '3';
        when "0100" => hex(hexlen -I) := '4';
        when "0101" => hex(hexlen -I) := '5';
        when "0110" => hex(hexlen -I) := '6';
        when "0111" => hex(hexlen -I) := '7';
        when "1000" => hex(hexlen -I) := '8';
        when "1001" => hex(hexlen -I) := '9';
        when "1010" => hex(hexlen -I) := 'A';
```

```
        when "1011" => hex(hexlen -I) := 'B';
        when "1100" => hex(hexlen -I) := 'C';
        when "1101" => hex(hexlen -I) := 'D';
        when "1110" => hex(hexlen -I) := 'E';
        when "1111" => hex(hexlen -I) := 'F';
        when "ZZZZ" => hex(hexlen -I) := 'z';
        when "UUUU" => hex(hexlen -I) := 'u';
        when "XXXX" => hex(hexlen -I) := 'x';
        when others => hex(hexlen -I) := '?';
       end case;
     end loop;
     return hex(1 to hexlen);
   end hstr;
 -- functions to manipulate strings
 ------------------------------------
 -- convert a character to upper case
 function to_upper(c: character) return character is
    variable u: character;
  begin
     case c is
      when 'a' => u := 'A';
      when 'b' => u := 'B';
      when 'c' => u := 'C';
      when 'd' => u := 'D';
      when 'e' => u := 'E';
      when 'f' => u := 'F';
      when 'g' => u := 'G';
      when 'h' => u := 'H';
      when 'i' => u := 'I';
      when 'j' => u := 'J';
      when 'k' => u := 'K';
      when 'l' => u := 'L';
      when 'm' => u := 'M';
      when 'n' => u := 'N';
      when 'o' => u := 'O';
      when 'p' => u := 'P';
      when 'q' => u := 'Q';
      when 'r' => u := 'R';
      when 's' => u := 'S';
      when 't' => u := 'T';
      when 'u' => u := 'U';
      when 'v' => u := 'V';
```

```
    when 'w' => u := 'W';
    when 'x' => u := 'X';
    when 'y' => u := 'Y';
    when 'z' => u := 'Z';
    when others => u := c;
  end case;
   return u;
 end to_upper;
 -- convert a character to lower case
 function to_lower(c: character) return character is
   variable l: character;
 begin
   case c is
    when 'A' => l := 'a';
    when 'B' => l := 'b';
    when 'C' => l := 'c';
    when 'D' => l := 'd';
    when 'E' => l := 'e';
    when 'F' => l := 'f';
    when 'G' => l := 'g';
    when 'H' => l := 'h';
    when 'I' => l := 'i';
    when 'J' => l := 'j';
    when 'K' => l := 'k';
    when 'L' => l := 'l';
    when 'M' => l := 'm';
    when 'N' => l := 'n';
    when 'O' => l := 'o';
    when 'P' => l := 'p';
    when 'Q' => l := 'q';
    when 'R' => l := 'r';
    when 'S' => l := 's';
    when 'T' => l := 't';
    when 'U' => l := 'u';
    when 'V' => l := 'v';
    when 'W' => l := 'w';
    when 'X' => l := 'x';
    when 'Y' => l := 'y';
    when 'Z' => l := 'z';
    when others => l := c;
  end case;
   return l;
```

```
  end to_lower;
  -- convert a string to upper case
  function to_upper(s: string) return string is
   variable uppercase: string (s'range);
  begin
   for i in s'range loop
     uppercase(i):= to_upper(s(i));
   end loop;
   return uppercase;
  end to_upper;
  -- convert a string to lower case
  function to_lower(s: string) return string is
   variable lowercase: string (s'range);
  begin
   for i in s'range loop
     lowercase(i):= to_lower(s(i));
   end loop;
   return lowercase;
  end to_lower;
-- functions to convert strings into other types
-- converts a character into a std_logic
function to_std_logic(c: character) return std_logic is
   variable sl: std_logic;
   begin
    case c is
      when 'U' =>
        sl := 'U';
      when 'X' =>
        sl := 'X';
      when '0' =>
        sl := '0';
      when '1' =>
        sl := '1';
      when 'Z' =>
        sl := 'Z';
      when 'W' =>
        sl := 'W';
      when 'L' =>
        sl := 'L';
      when 'H' =>
        sl := 'H';
      when '-' =>
```

```
        sl := '-';
      when others =>
        sl := 'X';
   end case;
  return sl;
 end to_std_logic;
-- converts a string into std_logic_vector
function to_std_logic_vector(s: string) return std_logic_vector is
 variable slv: std_logic_vector(s'high-s'low downto 0);
 variable k: integer;
begin
  k := s'high-s'low;
 for i in s'range loop
   slv(k) := to_std_logic(s(i));
   k     := k - 1;
 end loop;
 return slv;
end to_std_logic_vector;
----------------
--  file I/O  --
----------------
-- read variable length string from input file
procedure str_read(file in_file: TEXT;
                res_string: out string) is
     variable l:       line;
     variable c:       character;
     variable is_string: boolean;
  begin
   readline(in_file, l);
   -- clear the contents of the result string
   for i in res_string'range loop
      res_string(i) := ' ';
   end loop;
   -- read all characters of the line, up to the length
   -- of the results string
   for i in res_string'range loop
     read(l, c, is_string);
     res_string(i) := c;
     if not is_string then -- found end of line
       exit;
     end if;
   end loop;
```

```
end str_read;
-- print string to a file
procedure print(file out_file: TEXT;
                new_string: in  string) is
      variable l: line;
  begin
    write(l, new_string);
    writeline(out_file, l);
end print;
-- print character to a file and start new line
procedure print(file out_file: TEXT;
                char: in  character) is
      variable l: line;
  begin
    write(l, char);
    writeline(out_file, l);
end print;
-- appends contents of a string to a file until line feed occurs
-- (LF is considered to be the end of the string)
procedure str_write(file out_file: TEXT;
                    new_string: in  string) is
 begin
  for i in new_string'range loop
     print(out_file, new_string(i));
     if new_string(i) = LF then -- end of string
        exit;
     end if;
  end loop;
end str_write;
end txt_util;
```

Z.1.2 Αρχείο FILE_READ.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
use std.textio.all;
use work.txt_util.all;
entity FILE_READ is
generic (    stim_file:      string  := "S.txt"   );
```

```
 port(        CLK          : in  std_logic;
              RST          : in  std_logic;
               Y           : out std_logic_vector(23 downto 0)          := (others => '0');
               EOG          : out std_logic);
end FILE_READ;
architecture Behavioral of FILE_READ is
function str_to_stdvec(inp: string) return std_logic_vector is
      variable temp: std_logic_vector(inp'range) := (others => '0');
   begin
      for i in inp'range loop
         if (inp(i) = '1') then
            temp(i) := '1';
         elsif (inp(i) = '0') then
           temp(i) := '0';
         end if;
      end loop;
      return temp;
   end function str_to_stdvec;
file stimulus: TEXT open read_mode is stim_file;
begin
-- read data and control information from a file
receive_data: process
                        variable l: line;
                        variable s: string(24 downto 1);
                        begin
                                EOG <= '0';
                                -- wait for Reset to complete
                                wait until RST='1';
                                wait until RST='0';
                                while not endfile(stimulus) loop
                                -- read digital data from input file
                                readline(stimulus, l);
                                read(l, s);
                                --Y <=  str_to_stdvec(s);
                                Y <= to_std_logic_vector(s);
                                wait until CLK = '1';
                                end loop;
                        print("I@FILE_READ: reached end of "& stim_file);
                        EOG <= '1';
                        wait;
 end process receive_data;
end Behavioral;
```

Ζ.1.3 Αρχείο FILE_WRITE.vhd

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use std.textio.all;
use work.txt_util.all;
entity FILE_WRITE is
generic (  write_file:      string  := "OUT.TXT"        );
 port(       CLK   : in std_logic;
             RST   : in std_logic;
             X                : in std_logic_vector(23 downto 0);
             EOG          : in std_logic;
             valid_bit   : in std_logic);
end FILE_WRITE;
architecture Behavioral of FILE_WRITE is
file out_file: TEXT open write_mode is write_file;
signal ok : std_logic :='1';
begin
delay: process (clk)
 begin
             if(clk'event and clk = '1')then
                          if(eog = '1')then
                                       ok <= '0';
                          elsif(eog = '0')then
                                       ok <= '1';
                          end if;
             end if;
end process delay;
receive_data: process
variable OUTLINE: line;
begin
  wait until RST='1';
  wait until RST='0';
  while true loop
                          if (ok  = '1') and (valid_bit = '1')then
                                       write(OUTLINE, str(X));              --add variable to outline variable for output.
                                       writeline(out_file, OUTLINE);      --writes outline variable to file.
                          end if;
   wait until CLK = '1';
```

```
  end loop;
 end process receive_data;
end Behavioral;
```

Ζ.1.4 Αρχείο TB_FILE_READ.vhd

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TB_FILE_READ is
---port(     -- not used for simulation only for execute
             --clk          : in  std_logic;                          -- not used for simulation only for execute
             --rst          : in  std_logic;                          -- not used for simulation only for execute
             --s_i_dieythinsi:std_logic_vector(3 downto 0)            -- not used for simulation only for execute
             --);                                                     -- not used for simulation only for execute
end TB_FILE_READ;
architecture Behavioral of TB_FILE_READ is
component FILE_READ
             generic (stim_file:     string  := "S.txt");
             port(       CLK          : in  std_logic;
                         RST          : in  std_logic;
                         Y            : out std_logic_vector(23 downto 0);
                         EOG          : out std_logic);
end component;
component FILE_WRITE
             generic (write_file:     string  := "OUT.TXT");
             Port(       CLK          : in std_logic;
                         RST  :       in std_logic;
                         X            : in std_logic_vector(23 downto 0);
                         EOG          : in std_logic;
                          valid_bit   : in std_logic);
end component;
component Q2_select
Port (       clk       : in  std_logic;                              -- clock input
             rst       : in  std_logic;                              -- reset input
             in_data       : in  STD_LOGIC_VECTOR (23 downto 0);     -- data input
             valid_bit   : out  STD_LOGIC;                           -- valid bit A output
             out_data    : out  STD_LOGIC_VECTOR (23 downto 0) );   -- data output
end component;
signal rst                 :std_logic;
```

```
signal clk              :std_logic := '1';
signal eog              :std_logic;
signal s_valid_bit      :std_logic;
--signal y:             std_logic_vector(31 downto 0);
signal data_in:         std_logic_vector(23 downto 0) := (others => '0');
signal data_out:        std_logic_vector(23 downto 0) := (others => '0');
begin
rst <= '0', '1' after 40 ns, '0' after 100 ns;
clk <= not clk after 10 ns;
input_stim: FILE_READ
        port map(CLK  => clk,RST  => rst,Y    => data_in,EOG  => eog);
select_Q2: Q2_select
        port map(  clk => clk,rst => rst,in_data   => data_in,valid_bit  => s_valid_bit,out_data => data_out);
output_stim: FILE_WRITE
        port map(  CLK  => clk,RST  => rst,  X   => data_out,EOG  => eog,valid_bit => s_valid_bit);
end Behavioral;
```

Z.2 Πρωτόκολλο Επικοινωνίας UART

Z.2.1 Entity uart

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity uart is
        generic (   --Default setting :
                    -- 19200 baud , 8 data bits , 1 stop bit , 2^2 FIFO
                    DBIT: integer :=8;                          -- # data bits
                    SB_TICK : integer :=16;             -- # ticks for stop bits , 16 / 24 / 32
                                                        -- for 1 / 1.5 / 2 stop bits
                    DVSR : integer := 163;              -- baud rate divisor
                                                        -- DVSR = 50 M / ( 16 * baudrate )
                    DVSR_BIT : integer :=8;             -- # bits of DVSR
                    FIFO_W : integer:=4                 -- # addr bits of FIFO
                                                        -- # words in FIFO=2^FIFO_W
                                        );
        port (      clk, reset: in std_logic;
                    rd_uart , wr_uart : in std_logic ;
                    rx: in std_logic;
```

```
                w_data: in std_logic_vector ( 7 downto 0 ) ;
                tx_empty2,tx_full, rx_empty: out std_logic;
                r_data: out std_logic_vector ( 7 downto 0) ;
                tx: out std_logic);
end uart;
architecture str_arch of uart is
signal tick: std_logic ;
signal rx_done_tick: std_logic;
signal tx_fifo_out : std_logic_vector (7 downto 0);
signal rx_data_out : std_logic_vector (7 downto 0);
signal tx_empty, tx_fifo_not_empty : std_logic;
signal tx_done_tick : std_logic;
begin
        baud_gen_unit2: entity work.mod_m_counter (arch)
                generic map(M=>DVSR , N=>DVSR_BIT)
                port map(clk=>clk, reset=>reset, q=>open, max_tick=>tick) ;
        uart_rx_unit: entity work.uart_rx(arch)
                generic map(DBIT=>DBIT, SB_TICK=>SB_TICK)
                port map(clk=>clk, reset=>reset, rx=>rx, s_tick=>tick, rx_done_tick=>rx_done_tick, dout=>rx_data_out);
        fifo_rx_unit: entity work.fifo(arch)
                generic map(B=>DBIT , W=>FIFO_W)
                port map(clk=>clk, reset=>reset, rd=>rd_uart, wr=>rx_done_tick, w_data=>rx_data_out, empty=>rx_empty,
full=>open, r_data=>r_data);
fifo_tx_unit: entity work.fifo(arch)
generic map(B=>DBIT , W=>FIFO_W)
port map(clk=>clk, reset=>reset, rd=>tx_done_tick, wr=>wr_uart, w_data=>w_data, empty=>tx_empty, full=>tx_full, r_data=>tx_fifo_out);
        uart_tx_unit : entity work.uart_tx(arch)
                generic map ( DBIT => DBIT , SB_TICK => SB_TICK )
                port map(clk=>clk, reset=>reset, tx_start=>tx_fifo_not_empty, s_tick=>tick, din=>tx_fifo_out, tx_done_tick=>
tx_done_tick, tx=>tx);
        tx_fifo_not_empty <= not tx_empty;
        tx_empty2 <=tx_empty;
end str_arch;
```

Z.2.2 Entity mod_m_counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all ;
entity mod_m_counter is
```

```
generic (   N: integer := 8; -- number of bits
            M: integer := 163 -- mod 4
            );
port (      clk, reset: in std_logic;
            max_tick: out std_logic;
            q: out std_logic_vector (N-1 downto 0));
end mod_m_counter ;
architecture arch of mod_m_counter is
signal r_reg : unsigned (N-1 downto 0 ) ;
signal r_next : unsigned (N-1 downto 0 ) ;
begin
            -- register
            process (clk, reset)
                        begin
                                    if (reset='1') then
                                                r_reg <= (others =>'0') ;
                                    elsif (clk'event and clk='1') then
                                                r_reg <= r_next;
                                    end if ;
            end process ;
            --next - state logic
            r_next <=(others =>'0') when r_reg=(M-1) else
                                                r_reg + 1;
            --output logic
            q <= std_logic_vector(r_reg);
            max_tick <= '1' when r_reg=(M-1) else '0';
end arch;
```

Z.2.3 Entity uart_rx

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all ;
entity uart_rx is
            generic (DBIT: integer := 8 ;      -- data bits
                        SB_TICK: integer  := 16  -- ticks for stop bits
                        );
port (      clk , reset : in std_logic ;
            rx : in std_logic ;
            s_tick : in std_logic ;
```

```
        rx_done_tick : out std_logic ;
        dout : out std_logic_vector ( 7 downto 0)) ;
end uart_rx ;
architecture arch of uart_rx is
type state_type is ( idle , start , data , stop ) ;
signal state_reg , state_next : state_type ;
signal s_reg , s_next  : unsigned ( 3 downto 0) ;
signal  n_reg , n_next  : unsigned ( 2 downto 0) ;
signal  b_reg , b_next  : std_logic_vector (7 downto 0) ;

begin
-- FSMD state & data registers
process ( clk , reset )
        begin
                if (reset = '1') then
                        state_reg <= idle ;
                        s_reg <= (others =>'0');
                        n_reg <= (others =>'0');
                        b_reg <= (others =>'0');
                elsif clk'event and clk='1' then
                        state_reg <= state_next ;
                        s_reg <= s_next  ;
                        n_reg <= n_next  ;
                        b_reg <= b_next  ;
                end if ;
end process ;
-- next-state logic & data path functional units / routing
process ( state_reg, s_reg, n_reg, b_reg, s_tick, rx )
        begin
                state_next <= state_reg ;
                s_next  <= s_reg  ;
                n_next  <= n_reg  ;
                b_next  <= b_reg  ;
                rx_done_tick <= '0' ;
                case state_reg is
                        when idle =>
                                        if rx = '0' then
                                        state_next <= start ;
                                        s_next <=(others =>'0') ;
                                        end if ;
                        when start =>
                                        if  (s_tick = '1') then
```

```
                                        if  s_reg = 7 then
                                        state_next <= data ;
                                        s_next <= (others =>'0') ;
                                        n_next <= (others =>'0') ;
                        else
                                s_next  <= s_reg  + 1 ;
                        end if ;
                end if ;
                when data =>
                if  (s_tick = '1') then
                        if  s_reg = 15 then
                                s_next  <= (others => '0') ;
                                b_next  <= rx & b_reg  (7 downto 1) ;
                                if  n_reg = (DBIT - 1) then
                                        state_next <= stop ;
                                else
                                        n_next  <= n_reg + 1 ;
                                end if ;
                        else
                                s_next  <= s_reg + 1;
                        end if ;
                end if ;
                when stop =>
                if  (s_tick = '1') then
                        if  s_reg=(SB_TICK - 1) then
                                state_next <= idle ;
                                rx_done_tick <= '1';
                        else
                                s_next  <= s_reg+1;
                        end if ;
                end if ;
        end case ;
end process ;
dout <= b_reg  ;
end arch ;
```

Z.2.4 Entity uart_tx

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
use ieee.numeric_std.all ;
entity uart_tx is
	generic (	DBIT: integer :=8; -- # data bits
			SB_TICK: integer :=16 -- # ticks for stop bits
			);
port (		clk, reset: in std_logic;
		tx_start : in std_logic;
		s_tick : in std_logic ;
		din: in std_logic_vector (7 downto 0) ;
		tx_done_tick: out std_logic;
		tx: out std_logic);
end uart_tx ;
architecture arch of uart_tx is
type state_type is (idle, start, data, stop);
signal state_reg, state_next : state_type;
signal s_reg , s_next : unsigned (3 downto 0) ;
signal n_reg , n_next : unsigned (2 downto 0) ;
signal b_reg , b_next : std_logic_vector (7 downto 0 ) ;
signal tx_reg , tx_next : std_logic ;
begin
-- FSMD state & data registers
process (clk, reset)
	begin
		if reset='1' then
			state_reg <= idle;
			s_reg <= (others =>'0') ;
			n_reg <= (others =>'0') ;
			b_reg <= (others =>'0') ;
			tx_reg <='1';
		elsif ( clk'event and clk='1') then
			state_reg <= state_next;
			s_reg <= s_next;
			n_reg <= n_next;
			b_reg <= b_next;
			tx_reg <= tx_next;
		end if ;
end process ;
-- next - state logic & data path functional units / routing
process (state_reg, s_reg, n_reg, b_reg, s_tick, tx_reg, tx_start, din)
	begin
		state_next <= state_reg;
		s_next <= s_reg ;
```

```
n_next <= n_reg ;
b_next <= b_reg ;
tx_next <= tx_reg ;
tx_done_tick <='0';
case state_reg is
        when idle =>
                tx_next <='1';
                if  tx_start ='1' then
                        state_next <= start;
                        s_next <=( others =>'0') ;
                        b_next <= din ;
                end if ;
        when start =>
                tx_next <='0';
                if (s_tick ='1') then
                        if s_reg =15 then
                                state_next <= data ;
                                s_next <=(others =>'0');
                                n_next <=(others =>'0');
                        else
                                s_next <= s_reg + 1 ;
                        end if ;
                end if ;
        when data =>
                tx_next <= b_reg(0) ;
                if  (s_tick ='1') then
                                if s_reg = 15 then
                                        s_next <=(others =>'0');
                                        b_next <='0' & b_reg (7 downto 1);
                                        if n_reg = (DBIT -1) then
                                                state_next <= stop;
                                        else
                                                n_next <= n_reg + 1;
                                        end if ;
                                else
                                        s_next <= s_reg  + 1;
                                end if ;
                end if ;
        when stop =>
                tx_next <= '1';
                if (s_tick = '1') then
                        if s_reg  = (SB_TICK -1) then
```

```
                                        state_next <= idle;
                                        tx_done_tick <= '1';
                                    else
                                        s_next <= s_reg + 1;
                                    end if;
                                end if;
                    end case;
end process;
tx <= tx_reg;
end arch ;
```

Z.2.5 Entity fifo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
entity fifo is
        generic (   B: natural:=8; -- number of bits
                    W: natural:=4 -- number of address bits
                    );
        port (      clk, reset: in std_logic;
                    rd, wr: in std_logic;
                    w_data: in std_logic_vector (B-1 downto 0) ;
                    empty, full : out std_logic;
                    r_data: out std_logic_vector (B-1 downto 0));
end fifo;
architecture arch of fifo is
type reg_file_type is array ((2**W)-1 downto 0) of std_logic_vector (B-1 downto 0) ;
signal array_reg : reg_file_type;
signal w_ptr_reg, w_ptr_next, w_ptr_succ : std_logic_vector (W-1 downto 0) ;
signal r_ptr_reg, r_ptr_next, r_ptr_succ : std_logic_vector (W-1 downto 0) ;
signal full_reg, empty_reg, full_next, empty_next :std_logic;
signal wr_op : std_logic_vector (1 downto 0) ;
signal wr_en : std_logic ;
signal un1,un2,un3,un4 : unsigned(W-1 downto 0);
signal to_int1 : integer ;
begin
-- ============================================================================
-- register file
```

```
-- ============================================================================
process (clk, reset)
	begin
		if (reset='1') then
			array_reg <= (others=>(others=>'0'));
		elsif (clk'event and clk='1') then
			if wr_en='1' then

				--un3<= unsigned(w_ptr_reg);
				--to_int1 <= conv_integer(un3);

				--array_reg(to_integer(to_int1))<= w_data;
				array_reg(conv_integer(unsigned(w_ptr_reg(W-1 downto 0))))<= w_data;
				--array_reg(to_integer(unsigned(w_ptr_reg(W-1 downto 0))))<= w_data;
			end if ;
		end if ;
end process ;
-- read port
r_data <= array_reg(conv_integer(unsigned(r_ptr_reg(W-1 downto 0))));
--write enabled only when FIFO is not full
wr_en <= wr and (not full_reg);
-- ============================================================================
-- fifo control logic
-- ============================================================================
-- register for read and write pointers
process (clk, reset)
	begin
		if (reset='1') then
			w_ptr_reg <=(others =>'0');
			r_ptr_reg <=(others =>'0');
			full_reg <='0';
			empty_reg <='1';
		elsif (clk'event and clk='1') then
			w_ptr_reg <= w_ptr_next;
			r_ptr_reg <= r_ptr_next;
			full_reg <= full_next;
			empty_reg <= empty_next ;
		end if ;
end process ;
-- successive pointer values
-- now you can do conversions
--un1 <= unsigned(w_ptr_reg);
```

```
--un2 <= unsigned(r_ptr_succ);
un1 <= unsigned(w_ptr_reg) + 1;
un2 <= unsigned(r_ptr_reg) + 1;
w_ptr_succ <= std_logic_vector(un1);
r_ptr_succ <= std_logic_vector(un2);
--w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
--r_ptr_succ <= std_logic_vector((unsigned(r_ptr_reg)+1));
-- next-state  logic for read and write pointers
wr_op <= wr & rd;
process (w_ptr_reg, w_ptr_succ, r_ptr_reg, r_ptr_succ, wr_op, empty_reg, full_reg)
begin
        w_ptr_next <= w_ptr_reg;
        r_ptr_next <= r_ptr_reg;
        full_next <= full_reg;
        empty_next <= empty_reg;
        case wr_op is
                when "00" => -- no op
                when "01" => -- read
                        if (empty_reg /= '1') then -- not empty
                                r_ptr_next <= r_ptr_succ;
                                full_next <='0';
                                if (r_ptr_succ = w_ptr_reg ) then
                                        empty_next <='1';
                                end if ;
                        end if ;
                when "10" => -- write
                        if (full_reg /='1') then -- not full
                                w_ptr_next <= w_ptr_succ;
                                empty_next <='0';
                                if (w_ptr_succ = r_ptr_reg) then
                                        full_next <='1';
                                end if;
                        end if;
                when others => -- write/read ;
                        w_ptr_next <= w_ptr_succ;
                        r_ptr_next <= r_ptr_succ;
        end case;
end process;
-- output
full <= full_reg;
empty <= empty_reg;
end arch;
```

Ζ.3 Κώδικας File even_par.vhd [41]

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY even_par_testbench_vhd_tb IS
END even_par_testbench_vhd_tb;
ARCHITECTURE behavior OF even_par_testbench_vhd_tb IS
COMPONENT even_par
PORT(	a : IN std_logic;
		b : IN std_logic;
		c : IN std_logic;
		d : IN std_logic;
		ep : OUT std_logic);
END COMPONENT;
SIGNAL a : std_logic;
SIGNAL b : std_logic;
SIGNAL c : std_logic;
SIGNAL d : std_logic;
SIGNAL ep : std_logic;
BEGIN
uut: even_par PORT MAP(a => a,b => b,c => c,d => d,ep => ep);
-- *** Test Bench - User Defined Section ***
tb : PROCESS
BEGIN
-- All possible input combinations
		a <= '0'; b <= '0'; c <= '0'; d <= '0';
		wait for 20 ns;
		a <= '0'; b <= '0'; c <= '0'; d <= '1';
		wait for 20 ns;
		a <= '0'; b <= '0'; c <= '1'; d <= '0';
		wait for 20 ns;
		a <= '0'; b <= '0'; c <= '1'; d <= '1';
		wait for 20 ns;
		a <= '0'; b <= '1'; c <= '0'; d <= '0';
		wait for 20 ns;
		a <= '0'; b <= '1'; c <= '0'; d <= '1';
		wait for 20 ns;
		a <= '0'; b <= '1'; c <= '1'; d <= '0';
		wait for 20 ns;
		a <= '0'; b <= '1'; c <= '1'; d <= '1';
		wait for 20 ns;
```

```
		a <= '1'; b <= '0'; c <= '0'; d <= '0';
		wait for 20 ns;
		a <= '1'; b <= '0'; c <= '0'; d <= '1';
		wait for 20 ns;
		a <= '1'; b <= '0'; c <= '1'; d <= '0';
		wait for 20 ns;
		a <= '1'; b <= '0'; c <= '1'; d <= '1';
		wait for 20 ns;
		a <= '1'; b <= '1'; c <= '0'; d <= '0';
		wait for 20 ns;
		a <= '1'; b <= '1'; c <= '0'; d <= '1';
		wait for 20 ns;
		a <= '1'; b <= '1'; c <= '1'; d <= '0';
		wait for 20 ns;
		a <= '1'; b <= '1'; c <= '1'; d <= '1';
		wait; -- will wait forever
END PROCESS;
-- *** End Test Bench - User Defined Section ***
END;
```

Z.4 Κώδικας VHDL Counter.vhd[32]

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter is
		Port (	CLOCK : in STD_LOGIC;
			DIRECTION : in STD_LOGIC;
			COUNT_OUT : out STD_LOGIC_VECTOR (3 downto 0));
end counter;
architecture Behavioral of counter is
signal count_int : std_logic_vector(3 downto 0) := "0000";
begin
process (CLOCK)
		begin
		if CLOCK='1' and CLOCK'event then
			if DIRECTION='1' then
				count_int <= count_int + 1;
			else
				count_int <= count_int - 1;
```

```
                    end if;
            end if;
end process;
COUNT_OUT <= count_int;
end Behavioral;
```