# ΤΕΙ ΗΠΕΙΡΟΥ

## ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

**¨ Ανάπτυξη μηχανής πεπερασμένων καταστάσεων και αριθμητικής λογικής μονάδας σε γλώσσα VHDL ¨**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

Σιαμλίδης Ηλίας

Κανελλόπουλος Αθανάσιος

**Άρτα, Σεπτέμβριος 2015**

Η εργασία αυτή είναι αφιερωμένη
στους γονείς μας.

# Ευχαριστίες

Αρχικά, θα θέλαμε να ευχαριστήσουμε τον επιβλέπων καθηγητή μας κ. Αλέξανδρο Τζάλλα για τις πολύτιμες συμβουλές του και την υποστήριξη που μας πρόσφερε σε όλη την διάρκεια της εκπόνησης της παρούσας εργασίας, δίνοντάς μας όλες τις γνώσεις του στον συγκεκριμένο τομέα της ηλεκτρονικής για να έχουμε το επιθυμητό αποτέλεσμα, αλλά και την επιμονή και την υπομονή που έδειξε όλους αυτούς τους μήνες απέναντι μας.

Επίσης, ευχαριστούμε τον κ. Χρήστο Κολιοπάνο ο οποίος μέσα απο το μάθημα επιλογής του στο 7ο εξάμηνο με τίτλο " Σχεδίαση Ψηφιακών Τηλεπικοινωνιακών Συστημάτων με VHDL " μας έδωσε το ερέθυσμα την ιδέα αλλά και την έμπνευση να ξεκινήσουμε την παρούσα εργασία. Ακόμα, των ευχαριστούμε για τον τρόπο με τον οποίο μας έφερε σε επαφή με τον «κόσμο» της VHDL και των ηλεκτρονικών γενικότερα, κάνωντάς μας να «αγαπήσουμε» τον συγκεκριμένο κλάδο και να προχωρήσουμε στους «κόλπους» του μόνοι μας.

Τέλος, θα θέλαμε να πούμε ένα ΤΕΡΑΣΤΙΟ ΕΥΧΑΡΙΣΤΩ και στους αγαπημένους μας γονείς που όλα αυτά τα χρόνια μας στηρίζουν και συνεχίζουν να μας στηρίζουν για να πετύχουμε τους στόχους μας για να έχουμε ένα καλύτερο μέλλον.

# Περίληψη

Μέσα από την πτυχιακή εργασία που κάναμε , μας δόθηκε η δυνατότητα να εξερευνήσουμε και να μάθουμε πολλά πράγματα γύρο από τη γλώσσα VHDL και τις πλακέτες FPGA'S. Αρχικά, κάναμε μια εκτενή αναφορά στη γλώσσα VHDL και στης πλακέτες FPGA'S. Έπειτα, αναλύσαμε τι είναι μια μηχανή πεπερασμένων καταστάσεων και αναφέραμε και τις κατηγορίες των μηχανών. Στη συνέχεια, αναφερθήκαμε στο τι είναι μια αριθμητική λογική μονάδα. Επιπροσθέτως, μιλήσαμε για το εργαλείο με το οποίο κάναμε την υλοποίηση και την σχεδίαση των Project μας. Έπειτα, είπαμε λίγα λόγια για την εταιρία ALTERA. Επιπλέον, καταγράψαμε το διάγραμμα μιας μηχανής πεπερασμένων καταστάσεων και το αναλύσαμε. Στη συνέχεια, καταγράψαμε τα βήματα με τα οποία δημιουργούμε ένα Project πάνω στο εργαλείο QUARTUS II. Επιπροσθέτως, δημιουργήσαμε και εξηγήσαμε των κώδικα μιας μηχανής ανίχνευσης αλφαριθμητικών παραστάσεων. Ακόμα, δημιουργήσαμε και καταγράψαμε το κύκλωμα μιας αριθμητικής λογικής μονάδας και τον πίνακα αριθμητικών και λογικών πράξεων της. Έπειτα, δημιουργήσαμε και αναλύσαμε τον κώδικα της αριθμητικής και λογικής μονάδας που δημιουργήσαμε. Τέλος, καταγράψαμε τα συμπεράσματα που βγάλαμε μέσα από την πτυχιακή εργασία που κάναμε.

# Περιεχόμενα

# Λίστα Εικόνων

# ΚΕΦΑΛΑΙΟ 1ο

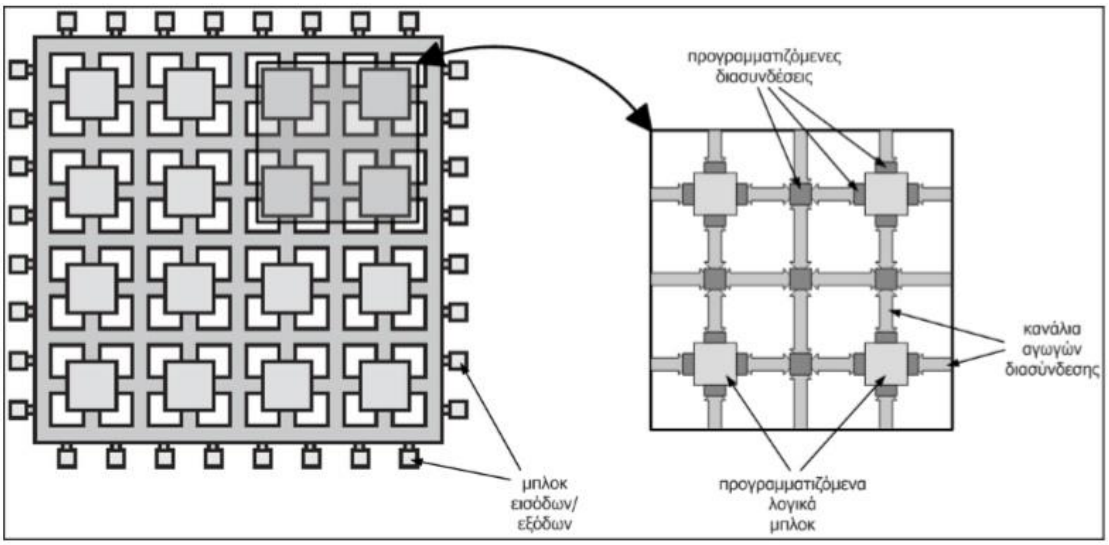
## 1.1 ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΓΛΩΣΣΑ VHDL

Η γλώσσα VHDL είναι μία γλώσσα περιγραφής υλικού (Hardware Description Language) η οποία χρησιμοποιείται για την δημιουργία ηλεκτρονικών σχεδιάσεων και την περιγραφή ψηφιακών και μεικτών συστημάτων,καθως και για την ανάπτυξη ολοκληρωμένων κυκλωμάτων.Ο όρος VHDL προέρχεται απο την λέξη VHSIC-HDL που σημαίνει Very High Speed Integrated Circuit-Hardware Description Language(Γλώσσα υψηλής ταχύτητας ολοκλήρωσης κυκλωμάτων-περιγραφής υλικού)και αποτελεί συντόμευσή της. Αρχικά,η γλώσσα ξεκίνησε απο το Υπουργείο Άμυνας των Η.Π.Α.στη δεκαετία του 1980 ώς μία καλύτερη λύση για την υλοποίηση των ολοκληρωμένων κυκλωμάτων,αντί των πολύπλοκων οδηγών της γλώσσας που κυκλοφορούσαν τότε.Το 1987 έγινε πρότυπη γλώσσα υλοποίησης απο το ινστιτούτο IEEE ώς IEEE 1076.Στην συνέχεια ακολούθησαν κάποιες αναβαθμίσεις του προτύπου οι οποίες είναι οι εξής:IEEE 1164 το 1963,IEEE 1076 το 2002,IEEE 1076 το 2008. Βασικά πλεονεκτήματα της γλώσσας VHDL είναι οτι όταν πρόκειται για σχεδίαση συστημάτων επιτρέπει την μοντελοποίηση και προσομοίωση πρίν τα εργαλεία σχεδίασης και σύνθεσης μετατρέψουν την σχεδίαση σε πύλες και καλώδια.Επίσης,η VHDL είναι μια γλώσσα ροής δεδομένων σε αντίθεση με τις γνωστές γλώσσες προγραμματισμού όπως είναι η PASCAL η BASIC και η C,αλλά η φιλοσοφία της είναι εντελώς διαφορετική.Η βασική διαφορά της γλώσσας,πέραν οτι είναι μια γλώσσα για το hardware όπως αναφέραμε και παραπάνω,είναι οτι οι εντολές της εκτελούνται παράλληλα (concurrent statement) ενω ταυτόχρονα υπάρχει και η δυνατότητα της σειριακής εκτέλεσης,σε αντίθεση με τις γλώσσες software όπου υπάρχει μόνο η δυνατότητα της σειριακής εκτέλεσης.Επίσης,οι γλώσσες PASCAL και C «προσαρμόζονται» σε μια CPU,ενω η γλώσσα VHDL «προσαρμόζεται» σε γενικές δομές του Hardware όπου η δομή του είναι παράλληλη.

Οι δύο βασικές εφαρμογές της VHDL είναι στον τομέα των προγραμματιζόμενων λογικών στοιχείων (PLA),το οποίο διαθέτει τα CPLD (Complex Programmable Logic Devices-Πολύπλοκα Προγραμματιζόμενα Λογκά Στοιχεία) και τα FPGA (Field Programmable Gate Arrays-Επιτόπου Προγραμματιζόμενοι Πίνακες Πυλών) και στον τομέα των κυκλωμάτων ASIC (Application Specific Integrated Circuits-Ολοκληρωμένα Κυκλώματα Ειδικού Σκοπού).Σήμερα,τα πιο ευρέως διαδεδομένα ολοκληρωμένα είναι τα FPGAs (Field Programmable Gate Arrays) διότι διαθέτουν περισσότερες απο 100.000 λογικές πύλες,αλλά παρουσιάζουν υψηλό κόστος αγοράς.Ακόμα,υπάρχουν στην αγορά και τα ολοκληρωμένα PLD και SFPGA.Για να προγραμματιστούν τα συγκεκριμένα έτοιμα ολοκληρωμένα πέρα απο την VHDL και έναν simulator (π.χ. Quartus II της ALTERA) είναι αναγκαίο να έχουμε και ένα εργαλείο σύνθεσης (Synthesis Tool) στην διάθεση μας.Τέτοια εργαλεία υπάρχουν στην αγορά κυρίως για FPGAs και PLDs των οποίων η λειτουργικότητα τους είναι ελαφρώς περιορισμένη σε σχέση με τις εκδόσεις που παρέχονται για Workstations.Τέλος,αξίζει να αναφέρουμε πως έχουν αναπτυχθεί και άλλες γλώσσες περιγραφής υλικού όπως είναι η VERILOG η οποία κάνει χρήση του επιπέδου RTL,η

SLIDE (Structured Language for Interface Description and Evaluation) το 1981,η CONLAN (CONsensus LANguage) το 1983,η ISPS (Instruction Set Processor Specification) το 1979,η ADLIB (A Design Language for Indicating Behaviour) το 1979,η OODE (Object Oriented Description Environment) το 1981,η BORIS (Block Oriented Interacting Simulation System) το 1984,η ZEUS η TEGAS η TI-HDL και η CDL,αλλά η VHDL έχει ξεχωρίσει γιατί είναι standard και είναι κορυφαία στον ακαδημαϊκό χώρο και στον χώρο της αγοράς και επειδή κορυφαίες εταιρίες χρησιμοποιούν την VHDL για να σχεδιάσουν ψηφιακά κυκλώματα.

## 1.2 FPGAS - FIELD PROGRAMMABLE GATE ARRAYS




**Εικόνα 1: Κάτοψη γενικής αρχιτεκτονικής FPGA**

Τα FPGAs δημιουργήθηκαν από την εταιρία Xilinx και άρχισαν να πωλούνται το 1984. Τα FPGAs είναι ψηφιακά ολοκληρωμένα κυκλώματα τα οποία περιέχουν προγραμματιζόμενα μπλοκ ψηφιακής λογικής. Ακόμα, υπήρχαν τα PLDs(Programmable Logic Devices) που ήταν και αυτά ψηφιακά ολοκληρωμένα κυκλώματα τα οποία ήταν προγραμματιζόμενα και χρειάζονταν λίγο χρόνο για την σχεδίαση τους. Το μειονέκτημα των PLDs όμως ήταν ότι δεν μπορούσαν να υλοποιήσουν πολύ μεγάλες ή πολύπλοκες λογικές συναρτήσεις. Επιπλέον, υπάρχει άλλο ένα είδος ψηφιακών ολοκληρωμένων κυκλωμάτων τα ASICs(Application-Specific Integrated Circuits). Τα ASICs είχαν το πλεονέκτημα ότι μπορούσαν να υλοποιήσουν πολύ μεγάλες και πολύπλοκες λογικές συναρτήσεις, αλλά το μειονέκτημα τους ήταν ότι ο χρόνος για την σχεδίαση του και το κόστος ήταν υπερβολικά μεγάλα.

Τα FPGAs αποτελούνται από προγραμματιζόμενα μπλοκ ψηφιακής λογικής τα οποία ενώνονται μεταξύ τους με την βοήθεια προγραμματιζόμενων διασυνδέσεων. Τα μπλοκ έχουν την μορφή ενός δισδιάστατου πίνακα. Ένα κανάλι διασύνδεσης αποτελείται από έναν αριθμό αγωγών διασύνδεσης, οι οποίοι αγωγοί είναι διατεταγμένοι πάνω στο ολοκληρωμένο σε οριζόντια και κάθετη μορφή. Έτσι, με αυτή την διάταξη γίνεται εφικτή η υλοποίηση πολυεπίπεδων λογικών συναρτήσεων.

Η εταιρία Xilinx που δημιούργησε τα FPGAs έδωσε την δυνατότητα στον χρήστη να τα προγραμματίσει μόνος του. Ο χρήστης είναι αυτός που αποφασίζει την λειτουργία θα κάνει το ολοκληρωμένο, ο οποίος κάνει την επιλογή για το ποιες λογικές συναρτήσεις θα επιλέξει για να δημιουργήσει τα προγραμματιζόμενα λογικά μπλοκ, τις μεταξύ διασυνδέσεις και τις διασυνδέσεις των μπλοκ εισόδων-εξόδων. Ο τρόπος που γίνεται αυτός ο προγραμματισμός των ολοκληρωμένων είναι μέσω λογισμικού και υλικού που διαθέτουν οι εταιρείες κατασκευής των FPGAs.

### 1.2.1 ΤΙ ΕΙΝΑΙ FPGA

Το FPGA έχει την μορφή ενός προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης. Το FPGA κατέχει πολλές τυποποιημένες πύλες και άλλες ψηφιακές λειτουργίες όπως απαριθμητές και καταχωρητές μνήμης. Όταν προγραμματίζεται το FPGA πρέπει να είναι συνδεδεμένο στο τυπωμένο κύκλωμα και έτσι οι επιθυμητές λειτουργίες ενεργοποιούνται και διασυνδέονται μεταξύ τους. Έτσι, στο τέλος το FPGA γίνεται ένα ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία. Το FPGA προγραμματίζεται με κώδικα σε γλώσσα περιγραφής υλικού οι οποίες είναι VHDL,AHDL και Verilog.

Εκτός από το FPGA υπάρχουν και άλλα προγραμματιζόμενα ολοκληρωμένα ψηφιακά κυκλώματα που έχουν παρόμοιο πεδίο εφαρμογών αυτά είναι τα PLD και τα ASIC. Το FPGA όμως έχει κάποια ιδιαίτερα χαρακτηριστικά τα οποία είναι τα ακόλουθα:

- Στο FPGA όταν διακόπτεται η τροφοδοσία του τότε χάνεται και ο προγραμματισμός του. Για αυτό το λόγω χρειάζεται εξωτερικός επεξεργαστής ή μνήμη για να μπορεί να επανέρχεται αυτόματα κάθε φορά που θα επανέρχεται η τροφοδοσία.

- Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του επεξεργαστή ή τα δεδομένα της μνήμης που το ελέγχει.

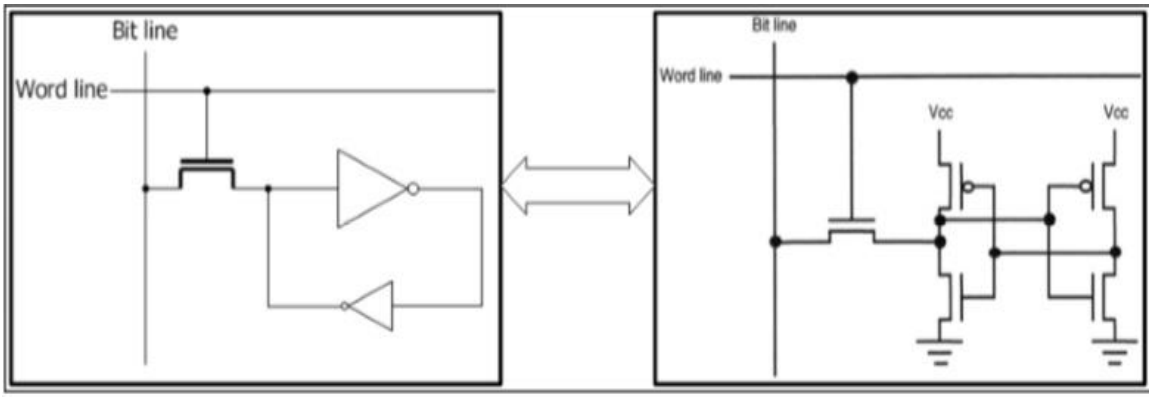
- Μπορεί να επαναπρογραμματιστεί άπειρες φορές.

Η κατανάλωση ισχύος είναι σημαντικά αυξημένη , σε σχέση με τα ASIC

### 1.2.2 ΔΟΜΗ - ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ FPGA

Για την υλοποίηση των FPGAs υπάρχουν πολλές τεχνολογίες. Μερικές από αυτές τις τεχνολογίες είναι οι ακόλουθες: Antifuse, E²PROM, FLASH,SRAM. Η πιο διαδεδομένη από τις τεχνολογίες είναι η SRAM προγραμματιζόμενων στοιχείων. Είναι η πιο γνωστή τεχνολογία γιατί χρησιμοποιείται πιο πολύ στις τυπικές εφαρμογές και ακόμα η αρχιτεκτονική των FPGAs είναι ως επί το πλείστον ανεξάρτητη από την τεχνολογία υλοποίησης. Ακόμα, λιγότερο διαδεδομένη από την τεχνολογία SRAM είναι η Antifuse. Η διαφορά της Antifuse σε σχέση με την SRAM είναι ότι η Antifuse είναι OTP(One Time Programmable) δηλαδή ότι προγραμματίζεται μόνο μία φορά. Επιπλέον, η Antifuse δεν επηρεάζεται καθόλου από φαινόμενα ακτινοβολίας. Έτσι, η τεχνολογία Antifuse είναι κατάλληλη για στρατιωτικές και διαστημικές εφαρμογές.

### 1.2.3 ΤΕΧΝΟΛΟΓΙΑ ΒΑΣΙΣΜΕΝΗ ΣΕ SRAM - ΣΤΟΙΧΕΙΑ

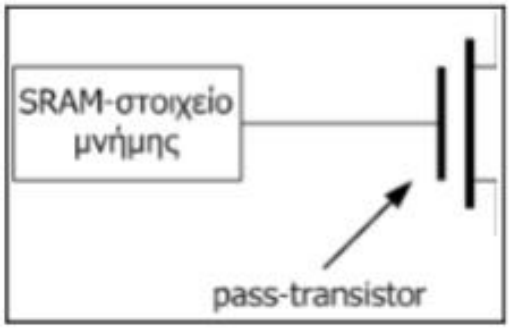



**Εικόνα 2: Ένα SRAM - στοιχείο μνήμης και η υλοποίηση με τρανζίστορ**

Οι μνήμες SRAM χρησιμοποιούν ως δομικό στοιχείο το SRAM-στοιχείο μνήμης. Η πληροφορία που μπορεί να αποθηκεύσει το SRAM-στοιχείο είναι μεγέθους 1 bit. Ενεργοποιώντας την γραμμή Word line για ένα ορισμένο χρονικό διάστημα τότε μπορεί να εισέλθει πληροφορία στο SRAM-στοιχείο μνήμης. Το χρονικό διάστημα που απαιτείται, είναι το διάστημα που θα κάνει η τιμή που βρίσκεται στην γραμμή Bit line να προλάβει να διαδοθεί μέσω των δύο αντιστροφέων που ορίζουν τον βρόχο. Η τιμή που θα αποθηκευτεί θα μείνει εκεί για πάντα εκτός αν αντικατασταθεί με

καινούργια τιμή και επιπλέον όταν διακοπεί η τροφοδοσία η τιμή του SRAM-στοιχείου μνήμης θα χαθεί.

Το SRAM-στοιχείο διασύνδεσης(pass-transistor ή SRAM switch) αποτελείται από δύο στοιχεία, το SRAM-στοιχείο μνήμης και το τρανζίστορ(pass transistor). Η λειτουργία του pass-transistor ως ανοικτός ή κλειστός διακόπτης γίνεται ανάλογα την τιμή που έχει αποθηκευτεί στο SRAM-στοιχείο μνήμης.

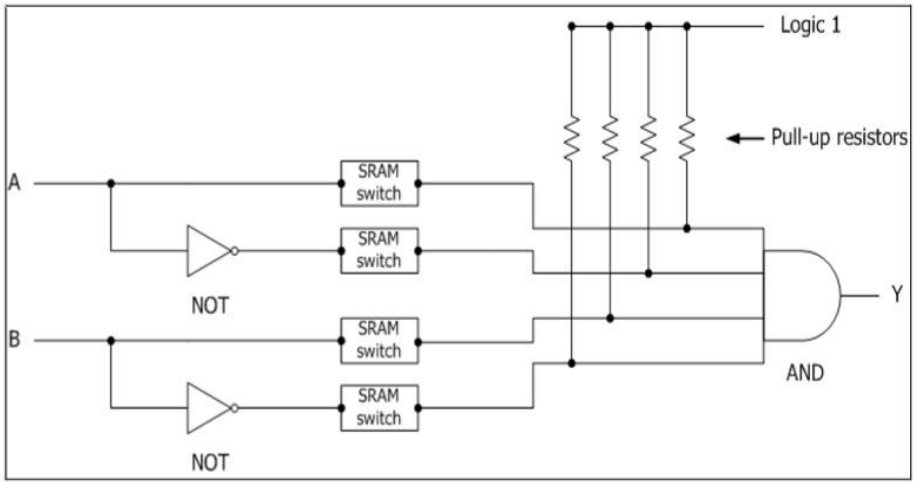



**Εικόνα 3: Ένα SRAM – στοιχείο διασύνδεσης (SRAM switch)**

Η τεχνολογία SRAM έχει και μειονεκτήματα αλλά και πλεονεκτήματα. Ένα από τα μειονεκτήματα είναι το μέγεθός του το οποίο καλύπτει μεγάλο χώρο πάνω στο ολοκληρωμένο. Ο λόγος είναι ότι αποτελείται από πέντε SRAM –στοιχεία μνήμης και ένα pass-transistor. Ακόμα τα SRAM-στοιχεία μνήμης μπορούν να χρησιμοποιηθούν ως latches, flip-flop ή και ως διαδεδομένη μνήμη μέσα στο ολοκληρωμένο. Για τον λόγω ότι τα στοιχεία που περιέχονται στην SRAM-στοιχείων μνήμης χάνονται αν η τροφοδοσία διακοπεί, τότε θα πρέπει να επαναπρογραμματιστουν. Για αυτό τον λόγω υπάρχουν οι εξωτερικές μνήμες E²PROM και FLASH οι οποίες όταν γίνεται η διακοπή και έπειτα επανέρχεται η τροφοδοσία τότε χρειάζονται 1 με 2 δευτερόλεπτα για να επαναπρογραμματιστεί μόνο του και έτσι δίνει την αίσθηση ότι είναι μόνιμα προγραμματισμένο. Με λίγα λόγια ένα SRAM-based FPGA έχει την δυνατότητα να προγραμματίζεται γρήγορα και επαναλαμβανόμενα. Άρα αυτός είναι ο λόγος που το κάνει περιζήτητο για την σχεδίαση νέων εφαρμογών, όπου απαιτούνται επαναλαμβανόμενες δοκιμές και διορθώσεις.

### 1.2.4 ΜΙΑ ΑΠΛΗ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΛΟΓΙΚΗ ΣΥΝΑΡΤΗΣΗ

Ας δούμε μία απλή προγραμματιζόμενη λογική συνάρτηση:

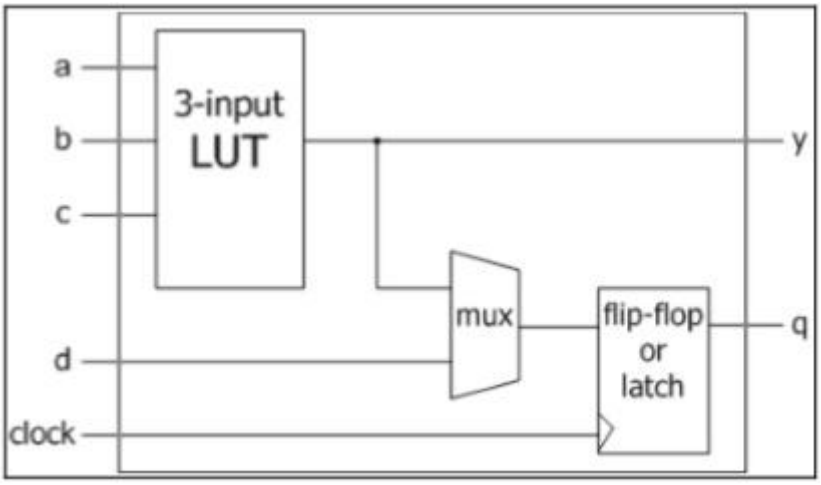



**Εικόνα 4: Μια απλή προγραμματιζόμενη λογική συνάρτηση**

Αρχικά, θεωρούμε ότι θέτοντας την λογική τιμή ‘1’ στο SRAM-στοιχείο μνήμης του SRAM-στοιχείου διασύνδεσης(SRAM switch) τότε αυτό γίνεται αγώγιμο, ενώ θέτοντας την λογική τιμή ‘0’ γίνεται μη αγώγιμο. Αν για παράδειγμα θέσουμε την λογική τιμή ‘1’ στο πρώτο και το τελευταίο SRAM- στοιχείο τότε το αποτέλεσμα που θα πάρουμε θα είναι Y=A AND (NOT B). Εκτός από την τεχνολογία SRAM-στοιχείο θα μπορούσε να χρησιμοποιηθούν και άλλες τεχνολογίες όπως Antifuse, $E^2$PROM και FLASH χωρίς να υπάρξει κάποια αλλαγή.

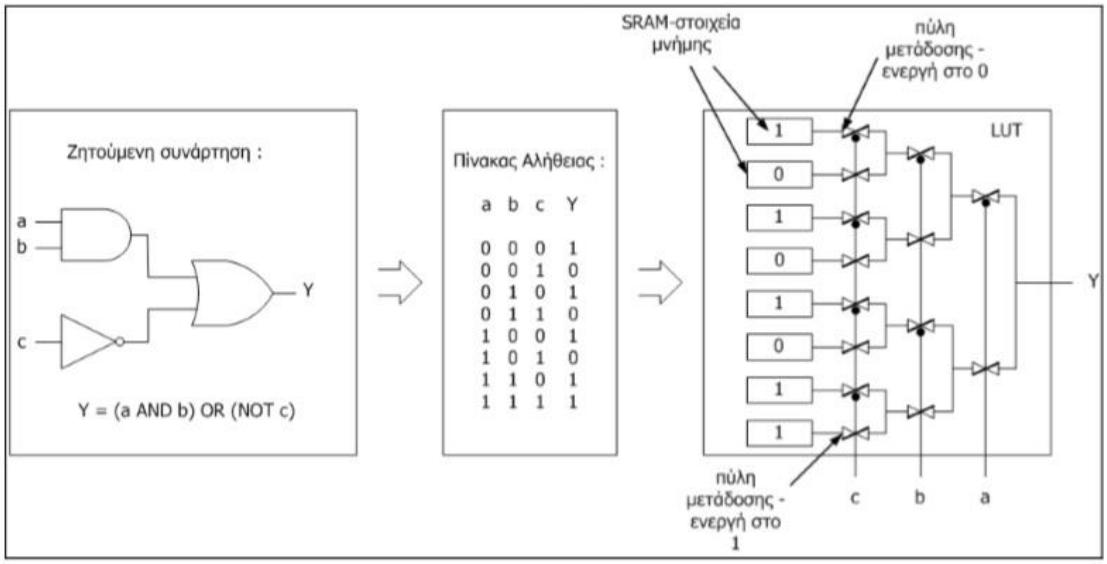
### 1.2.5 ΤΑ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΑ ΛΟΓΙΚΑ ΜΠΛΟΚ ΤΩΝ ΠΡΩΤΩΝ FPGAS

Στη τεχνολογία CMOS ήταν ήταν βασισμένα τα πρώτα FPGAs. Έτσι, για να προγραμματιστούν χρησιμοποιούσαν SRAM-στοιχεία. Στα πρώτα FPGAs τα προγραμματιζόμενα λογικά μπλοκ περιελάμβαναν ένα LUT(LookUp Table) τριών εισόδων, έναν καταχωρητή(register) του ενός bit ο οποίος μπορεί να συμπεριφέρεται ως D flip-flop ή ως latch και ένα πολυπλέκτη(multiplexer-mux).




**Εικόνα 5: Η γενική μορφή του προγραμματιζόμενου λογικού μπλόκ των πρώτων FPGAS**

Σε αυτό το σημείο θα αναλύσουμε τη λειτουργία και μια πιθανή υλοποίηση της δομής του LUT. Οι πύλες μετάδοσης οι οποίες είναι ενεργές στο λογικό ‘1’, δίνουν την δυνατότητα διέλευσης της εισόδου στην έξοδο όταν το σημείο ελέγχου είναι στο λογικό ‘1’. Ανάλογο έργο επιτελείται όταν οι πύλες μετάδοσης είναι ενεργές στο λογικό ‘0’ και επιτρέπει την διέλευση της εισόδου στην έξοδο όταν το σημείο ελέγχου είναι στο λογικό ‘0’. Αν για παράδειγμα θέσουμε την λογική τιμή ‘0’ και στις τρεις εισόδους του σχήματος της εικόνας _,τότε η έξοδος Υ θα έχει τιμή του πρώτου SRAM-στοιχείου μνήμης που είναι η λογική τιμή ‘1’. Οι τιμές που έχουν τα SRAM-στοιχεία μνήμης δεν είναι συγκεκριμένα αλλά είναι ανάλογα τον προγραμματισμό τους. Ανάλογα με τις εισόδους έχουμε και τις τιμές που παίρνουν τα SRAM-στοιχεία μνήμης, δικαιολογώντας έτσι την ονομασία LUTs. Στην πραγματικότητα όμως τα SRAM-στοιχεία μνήμης είναι συνδεδεμένα μεταξύ τους σε μορφή αλυσίδας.



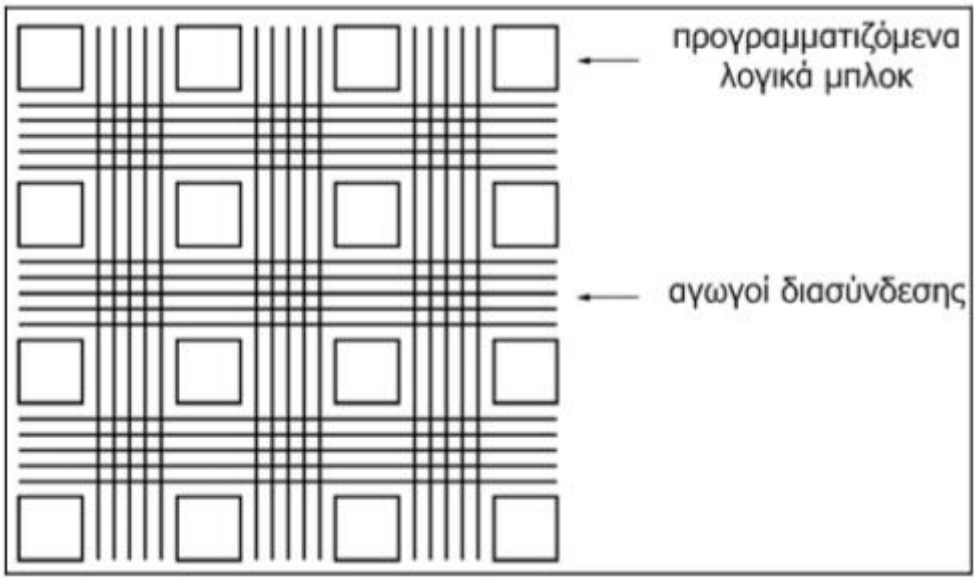

**Εικόνα 6: Ο τρόπος λειτουργίας και η δομή ενός LUT τριών εισόδων**

Ακόμα με την πάροδο του χρόνου και την αλματώδη ανάπτυξη της τεχνολογίας τα τελευταία χρόνια τα FPGAs συνεχίζουν να έχουν πολλά από τα χαρακτηριστικά των πρώτων FPGAs αλλά έχουν μεγαλύτερη πολυπλοκότητα.

### 1.2.5.1 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΔΙΑΣΥΝΔΕΣΗΣ

Η αρχιτεκτονική διασύνδεσης είναι ένα πολύ βασικό κομμάτι του FPGA. Σύμφωνα με την αρχιτεκτονική διασύνδεσης γίνεται η τοποθέτηση των προγραμματιζόμενων διασυνδέσεων και των αγωγών πάνω στο FPGA κατά την κατασκευή του. Οι υπηρεσίες που προσφέρει η αρχιτεκτονική διασύνδεσης είναι ο μεγάλος βαθμός διασύνδεσης, μεγάλη ταχύτητα μετάδοσης σημάτων και ακόμα περιλαμβάνει ικανοποιητικό αριθμό αγωγών.

Το ψευδώνυμο που έχει δοθεί στην γενική αρχιτεκτονική των σύγχρονων FPGAs είναι το Island-Style. Τα ``νησιά΄΄ είναι τα προγραμματιζόμενα λογικά μπλοκ τα οποία ``επιπλέουν΄΄ στη ``θάλασσα΄΄ διασυνδέσεων. Στο σχήμα της επόμενης εικόνας βλέπουμε τους αγωγού διασύνδεσης οι οποίοι ομαδοποιούνται σε κανάλια και διασχίζουν το ολοκληρωμένο σε οριζόντια και κάθετη μορφή.




**Εικόνα 7: Η απλοποιημένη μορφή της αρχιτεκτονικής island – style**

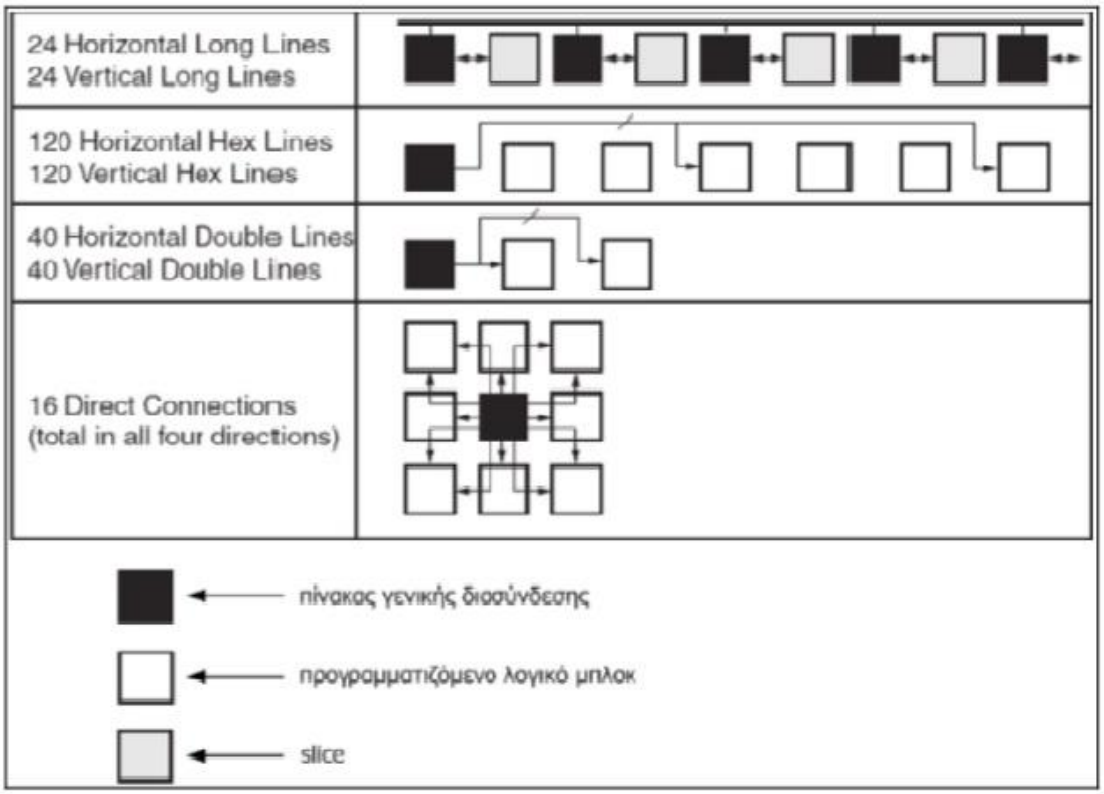
Η αρχιτεκτονική διασύνδεσης διαφέρει λίγο σε σχέση με τα παλαιότερα FPGAs και τα σύγχρονα FPGAs. Τα σύγχρονα FPGAs έχουν σαν χαρακτηριστικό την ιεραρχία και την ποικιλία στο μήκος των αγωγών διασύνδεσης. Ο αγωγός διασύνδεσης αποτελείται από ένα συνεχές αγώγιμο τμήμα που στη αρχή και στο τέλος του

υπάρχουν προγραμματιζόμενες διασυνδέσεις. Οι προγραμματιζόμενες διασυνδέσεις υλοποιούνται κυρίως με SRAM-στοιχεία διασύνδεσης και σε κάποιες περιπτώσεις με βαθμίδες απομόνωσης τριών καταστάσεων.

Ο λόγος που το μήκος των αγωγών διασύνδεσης δεν είναι σταθερό είναι διοτι η διάδοση των σημάτων κατά την διασύνδεση απομακρυσμένων προγραμματιζόμενων μπλοκ θα γινόταν με πολύ μεγάλη καθυστέρηση. Οι τύποι αγωγών που υπάρχουν σε κάθε κανάλι είναι οι ακόλουθοι:

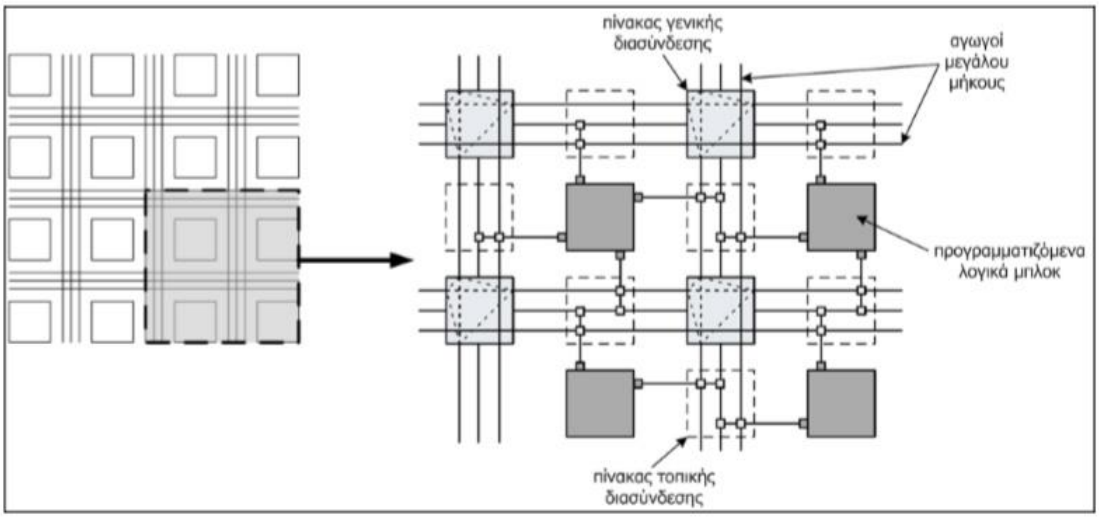
- αγωγοί με μήκος ικανό για την διασύνδεση γειτονικών προγραμματιζόμενων λογικών μπλοκ.
- αγωγοί για την διασύνδεση ενός προγραμματιζόμενου λογικού μπλοκ με το μπλοκ που ακολουθεί μετά το γειτονικό και βρίσκεται στην ίδια στήλη ή στην ίδια σειρά των προγραμματιζόμενων λογικών μπλοκ.
- αγωγοί για την διασύνδεση ενός προγραμματιζόμενου λογικού μπλοκ με το τρίτο ή το έκτο μπλοκ της ίδιας σειράς ή στήλης.
- αγωγοί που διατρέχουν όλο το ολοκληρωμένο για διασύνδεση κάθε προγραμματιζόμενου λογικού μπλοκ με κάποιο ή κάποια άλλα προγραμματιζόμενα λογικά μπλοκ, τα οποία ανήκουν στην ίδια σειρά ή στήλη.

Οι προαναφερόμενες διασυνδέσεις υλοποιούνται μέσω των πινάκων τοπικής και γενικής διασύνδεσης.




**Εικόνα 8: Είδη αγωγών διασύνδεσης στο FPGA Virtex – II Pro**

Η αρχιτεκτονική διασύνδεσης περιλαμβάνει επιπλέων δύο δομές, τους πίνακες τοπικής και γενικής διασύνδεσης. Οι πίνακες τοπικής διασύνδεσης είναι συνήθως τοποθετημένοι στις τέσσερις πλευρές κάθε προγραμματιζόμενου λογικού μπλοκ. Ακόμα, περιέχουν προγραμματιζόμενες διασυνδέσεις μέσω των οποίων συνδέουν τους ακροδέκτες ενός γειτονικού προγραμματιζόμενου λογικού μπλοκ με αγωγούς που οδηγούν σε πίνακες γενικής διασύνδεσης. Ο πίνακας γενικής διασύνδεσης είναι τοποθετημένος στις διασταυρώσεις των οριζόντιων και κάθετων καναλιών συνδέοντας με προγραμματιζόμενες διασυνδέσεις αγωγούς των οποίων τα άκρα καταλήγουν εκεί. Λόγω του μεγάλου μεγέθους και της μεγάλης χωρητικότητας C των προγραμματιζόμενων διασυνδέσεων έχουμε ότι κάθε αγωγός που ανήκει σε ένα κανάλι μπορεί να συνδεθεί μόνο σε ένα υποσύνολο των αγωγών του κάθε καναλιού.




**Εικόνα 9: Διασύνδεση των προγραμματιζόμενων λογικών μπλόκ**

## 1.3 ΤΙ ΕΙΝΑΙ ΜΙΑ ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ

Γενικά,μια μηχανή πεπερασμένων καταστάσεων (Finite State Machine – FSM) είναι μια ειδική τεχνική μοντελοποίησης για ακολουθιακά λογικά κυκλώματα,όπου αυτο το μοντέλο που θα προκύψει απο τον σχεδιασμό μπορεί να φανεί χρήσιμο για τον σχεδιασμό κάποιων συγκεκριμένων τύπων συστημάτων τα οποία εκτελούν μια καθορισμένη ακολουθία λειτουργιών.Τέτοια συστήματα μπορεί να είναι για παράδειγμα ψηφιακοί ελεγκτές,μικροεπεξεργαστές και διάφορα εξαρτήματα υπολογιστών.Οι μηχανές πεπερασμένων καταστάσεων (FSM) χρησιμοποιούνται κυρίως σε εφαρμογές της επιστήμης των υπολογιστών και σε δίκτυα δεδομένων,για παράδειγμα οι μηχανές πεπερασμένων καταστάσεων μπορούμε να πούμε πως είναι η βάση για προγράμματα ελέγχου ορθγραφίας,ελέγχου γραμματικής,κατασκευής ευρετηρίου ή αναζήτησης μεγάλων κειμένων,αναγνώρισης ομιλίας,μετασχηματισμού κειμένου με χρήση γλωσσών όπως η XML και η HTML και πρωτοκόλλων δικτύων τα οποία καθορίζουν τον τρόπο επικοινωνίας των δικτύων.

Πιο συγκεκριμένα,μια μηχανή πεπερασμένων καταστάσεων (FSM) αποτελείται απο:

$$FSM=(S , I , O , f , g , So)$$

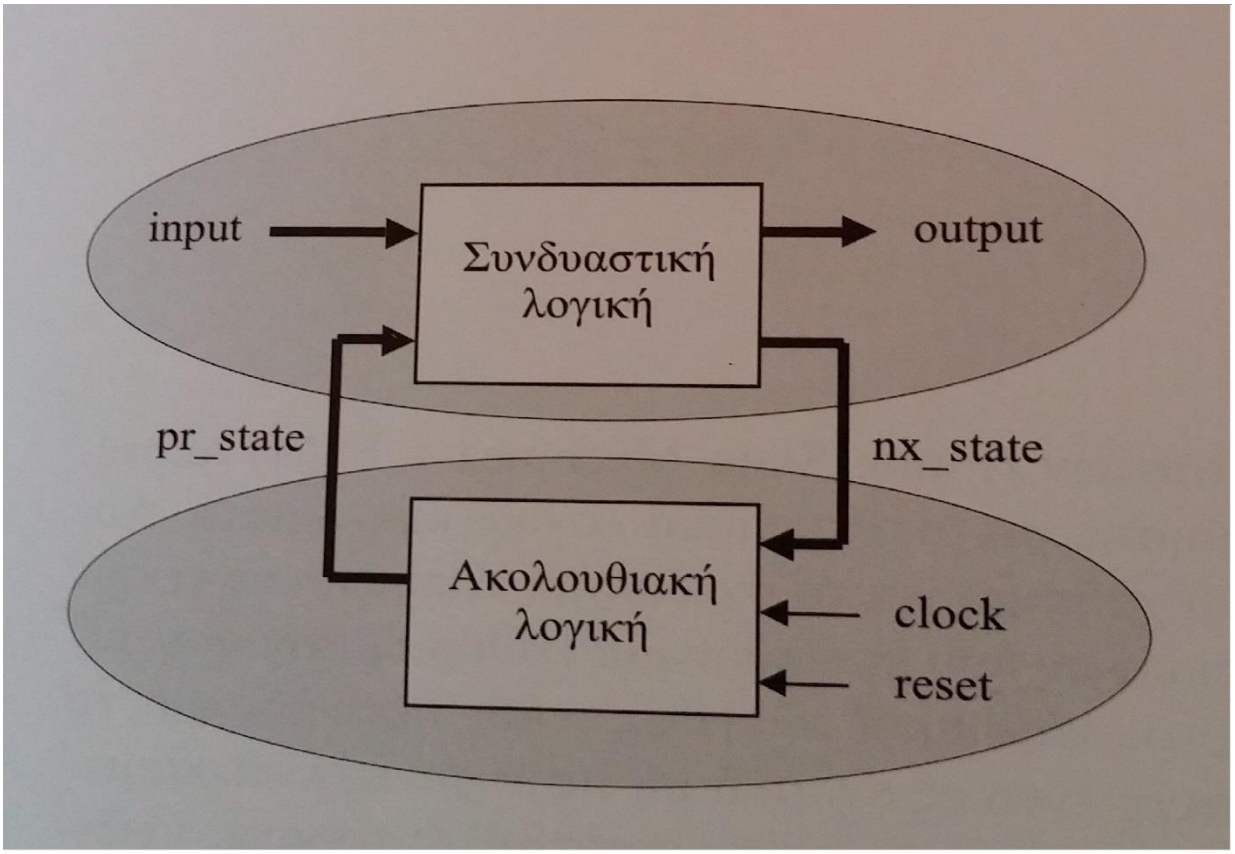
- Ένα πεπερασμένο σύνολο καταστάσεων S
- Ένα πεπερασμένο σύνολο απο γράμματα εισόδου I

- Ένα πεπερασμένο σύνολο απο γράμματα εξόδου O
- Μια συνάρτηση μετάβασης f ,η οποία αναθέτει νέα κατάσταση σε κάθε ζεύγος κατάστασης και εισόδου.
- Μια συνάρτηση εξόδου g ,η οποία αναθέτει μια έξοδο σε κάθε ζεύγος κατάστασης και εισόδου.
- Ένα ειδικό στοιχείο,του συνόλου S,που ονομάζεται αρχική κατάσταση So

Παρακάτω απεικονίζεται το δομικό διάγραμμα μιας μηχανής πεπερασμένων καταστάσεων μίας φάσης.Όπως παρατηρούμε διακρίνουμε δύο τμήματα,το τμήμα της συνδιαστικής λογικής (πάνω τμήμα) και το τμήμα της ακολουθιακής λογικής (κάτω τμήμα).

Το τμήμα της συνδιαστικής λογικής (πάνω τμήμα) περιλαμβάνει δύο εισόδους: την pr_state η οποία είναι η τρέχουσα κατάσταση και την input η οποία είναι η κύρια εξωτερική είσοδος.Τέλος,διαθέτει και δύο έξόδους: την nx_state η οποία μας δείχνει την επόμενη κατάσταση και την output η οποία είναι η κύρια έξοδος.

Το τμήμα της ακολουθιακής λογικής (κάτω τμήμα) περιλαμβάνει τρεις εισόδους: την clock ή clk η οποία είναι το ρολόι,την reset ή rst η οποία είναι απαραιτητη για το σήμα επαναφοράς και την nx_state η οποία μας δείχνει την επόμενη κατάσταση,όπως αναφέραμε και παραπάνω.Τέλος,διαθέτει και μια έξοδο: την pr_state η οποία είναι για την τρέχουσα κατάσταση.

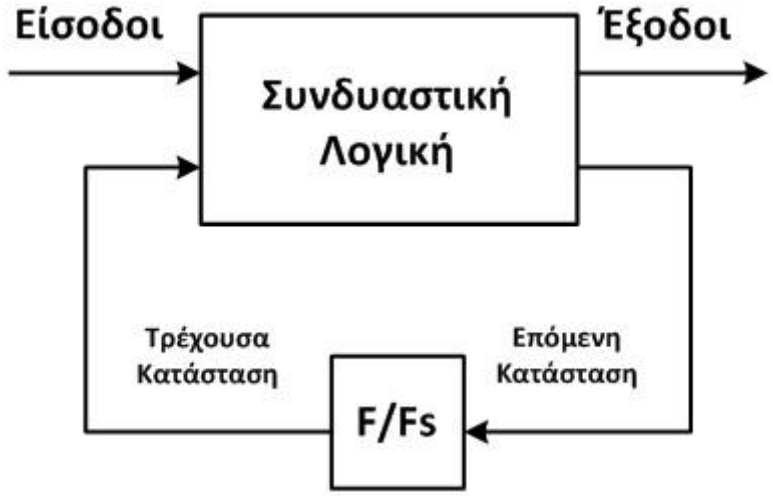



**Εικόνα 10: Διάγραμμα μηχανής καταστάσεων**

### 1.3.1 ΚΑΤΗΓΟΡΙΕΣ FINITE STATE MACHINE (FSM)

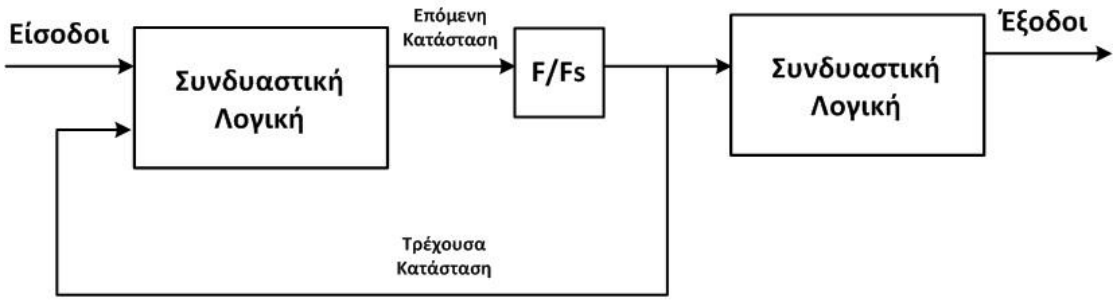
Γνωρίζοντας πλέον τι είναι μια μηχανή πεπερασμένων καταστάεων που χρησιμοποιείται και ποιά είναι τα δομικά της στοιχεία,πάμε να γνωρίσουμε τώρα και τις κατηγορίες στις οποίες διακρίνονται οι μηχανές πεπερασμένων καταστάσεων (FSMs).Για την κατασκευή μοντέλων υπολογιστικών μηχανών έχουν αναπτυχθεί πολλά διαφορετικά είδη μηχανών πεπερασμένης κατάστασης,αλλα έχουν επικρατήσει μόνο δύο,τα οποία διακρίνονται σε δύο κατηγορίες όπως φαίνονται παρακάτω και είναι οι εξής:

- Mealy FSMs: Οι μηχανές αυτού του είδους είναι γνωστές σαν μηχανές Mealy, επειδή μελετήθηκαν για πρώτη φορά απο τον G.H.Mealy το 1955.Χαρακτηριστικό των συγκεκριμένων μηχανών είναι ότι οι έξοδοι αντιστοιχούν σε μεταβάσεις μεταξύ καταστάσεων,δηλαδή με άλλα λόγια οι έξοδοι είναι συνάρτηση των εισόδων και της τρέχουσας κατάστασης.




**Εικόνα 11: Διάγραμμα Mealy FSM**

- Moore FSMs: Οι συγκεκριμένες μηχανές πεπερασμένης κατάστασης έγιναν γνωστές σαν μηχανές Moore,διότι τις είχε παρουσιάσει ο E.F.Moore το 1956 έναν χρόνο αργότερα μετά την εμφάνιση των Mealy μηχανών.Χαρακτηριστικό αυτών των μηχανών είναι ότι η έξοδος προσδιορίζεται μόνο απο την κατάσταση,δηλαδή πιο απλά οι έξοδοι είναι συνάρτηση **μόνο** της τρέχουσας κατάστασης.Ιδιαίτερο χαρακτηριστικό των μηχανών Moore (Moore FSMs) είναι ότι απαιτούν έναν κύκλο ρολογιού παραπάνω σε σχεση με τις μηχανές Mealy (Mealy FSMs) δηλαδή συνολικά οι μηχανές Moore απαιτούν δύο κύκλους ρολογιού,έναντι ενός των μηχανών Mealy.Οι κύκλοι ρολογιού είναι δύο,ένας κύκλος υπολογισμού της κατάστασης και ένας κύκλος για τον υπολογισμό της εξόδου.




**Εικόνα 12: Διάγραμμα Moore FSM**

### 1.3.2 ΑΝΙΧΝΕΥΤΗΣ ΑΛΦΑΡΙΘΜΗΤΙΚΩΝ ΠΑΡΑΣΤΑΣΕΩΝ

Επιθυμούμε να σχεδιάσουμε ένα κύκλωμα το οποίο θα δέχεται σαν είσοδο μια σειριακή δέσμη απο bit και θα δίνει στην έξοδο τιμή ‘1’ όταν εμφανιστεί η ακολουθία ‘’111’’.Θα πρέπει επίσης να λαμβάνονται υπόψη οι επικαλύψεις,δηλαδη αν εμφανιστεί η ακολουθία ...01111110... , τότε η έξοδος θα πρέπει να παραμείνει σε υψηλή στάθμη για τρεις διαδοχικούς κύκλους ρολογιού.Ακόμα θα πρέπει να επισημάνουμε ότι η έξοδος δεν εξαρτάται απο την τρέχουσα είσοδο,άρα κατά συνέπεια η έξοδος είναι σύγχρονη μηχανή Moore,επομένως δεν είναι απαραίτητη η χρήση της μηχανής Mealy.Τέλος,θα πρέπει να πούμε πως το κύκλωμα απαιτεί δύο flip-flop,που κωδικοποιούν τις τέσσερις καταστάσεις της μηχανής πεπερασμένων καταστάσεων όπως θα δούμε στην παράγραφο 2.1 .

## 1.4 ΤΙ ΕΙΝΑΙ ΜΙΑ ΑΡΙΘΜΗΤΙΚΗ ΛΟΓΙΚΗ ΜΟΝΑΔΑ (ALU)

Με τον όρο αριθμητική λογική μονάδα (ALU – Arithmetic Logic Unit) ουσιαστικά αναφερόμαστε σε μια μονάδα όπου αυτη η μονάδα είναι μια μονάδα του επεξεργαστή πάνω στην οποία εκτελούνται όλες οι αριθμητικές και λογικές πράξεις.Οι αριθμητικές πράξεις που μπορεί να εκτελέση είναι η πρόσθεση και η αφαίρεση (είναι και ο πολλαπλασιασμός και η διαίρεση όπου αυτές οι πράξεις προκύπτουν συναρτήση των άλλων δύο πράξεων),ενώ οι λογικές πράξεις είναι οι: not,and,or,nand,nor,xor καθώς και οι πράξεις των ολισθήσεων και των συγκρίσεων.Γενικά, μια αριθμητική λογική μονάδα (ALU) μπορεί να εκτελέσει πράξεις με ακέραιους και πραγματικούς αριθμούς,αλλά στην παρούσα εργασία θα ασχοληθούμε μόνο με ακέραιους αριθμούς,όλα αυτά θα τα δούμε παρακάτω πιο αναλυτικά.

Για να πραγματοποιηθούν οι πράξεις της πρόσθεσης και της αφαίρεσης χρησιμοποιείται ένα ψηφιακό κύκλωμα το οποίο εκτελεί την ανάλογη πράξη που εμείς θέλουμε να εκτελέσουμε(πρόσθεση ή αφαίρεση) και ονομάζεται αθροιστής (άν πρόκειται για την πράξη της πρόσθεσης) ή αφαιρετής (άν πρόκειται για την πράξη της αφαίρεσης) αντίστοιχα.Ο αθροιστής ή αφαιρετής (όπως αναλύσαμε παραπάνω) αποτελεί στην ουσία ένα βοηθητικό υπο-σύστημα της αριθμητικής λογικής μονάδας.Αποτελείται απο τόσους πλήρεις αθροιστές,όσο είναι το μέγεθος σε bit των εισόδων Α και Β της αριθμητικής λογικής μονάδας μας,οι οποίοι συνδέονται μεταξύ τους ώς εξής: Απο τις εισόδους Α και Β των αθροιστών δημιουργούνται οι είσοδοι Α και Β του αθροιστή – αφαιρετή.Απο τις εξόδους των αθροιστών δμιουργείται η έξοδος του αθροιστή – αφαιρετή με τον ίδιο τρόπο όπως και με τις εισόδους.Η έξοδος κρατουμένου του κάθε αθροιστή συνδέεται με την είσοδο κρατουμένου του επόμενου αθροιστή.Εξαιρούνται η είσοδος κρατουμένου του πρώτου αθροιστή, που αποτελεί

την είσοδο κρατουμένου του αθροιστή – αφαιρετή και η έξοδος του τελευταίου αθροιστή που αποτελεί την έξοδο κρατουμένου του αθροιστή – αφαιρετή.Η είσοδος Β του αθροιστή - αφαιρετή συνδέεται με ένα δικτύωμα xor.Το δικτύωμα αυτό στην μία είσοδό του έχει τον αριθμό εισόδου και στην άλλη παίρνει το σήμα επιλογής του αθροιστή – αφαιρετή.Όταν το σήμα επιλογής είναι ‘0’ , τότε το δικτύωμα xor περνάει τον αριθμό ώς έχει (A XOR ‘0’ = A).Στην αντίθετη περίπτωση, όταν δηλαδή το σήμα επιλογής είναι ‘1’ , το δικτύωμα xor δημιουργεί σε συνδυασμό με το σήμα επιλογής το «συμπλήρωμα ώς προς 2» του αριθμού (A XOR ΄1’ = A΄ συν την μονάδα στο αρχικό κρατούμενο) και έτσι ο αθροιστής γίνεται αφαιρετής, καθώς εκτελεί αφαίρεση [(A + (-B)) = (A – B)].

Όσον αφορά τις λογικές πράξεις και την πράξη της σύγκρισης τα πράγματα είναι πιο ξεκάθαρα διότι για να υλοποιηθούν ακολουθούν Behavioral σχεδίαση σε αντίθεση με τον αθροιστή – αφαιρετή ο οποίος ακολουθεί Structural σχεδίαση η οποία και είναι σαφώς πιο πολύπλοκη – δύσκολη.Όταν η αριθμητική λογική μονάδα (ALU) εκτελεί μια λογική πράξη, έχει στο εσωτερικό της ένα δικτύωμα πυλών όπως and, or και nor, το οποίο παίρνει στις εισόδους του τους δύο αριθμούς και βγαζει στην έξοδο το αποτέλεσμα της αντίστοιχης πράξης.Στην περίπτωση της σύγκρισης, υπάρχει ένας συγκριτής ο οποίος όταν η είσοδος Α είναι μικρότερη απο την είσοδο Β βγάζει ‘1’, ενώ στην αντίθετη περίπτωση, όταν δηλαδή η είσοδος Α είναι μεγαλύτερη ή ίση απο την είσοδο Β, βγάζει ‘0’. Η έξοδος του συγκριτή συνδέεται με το λιγότερο σημαντικό bit της εξόδου της ALU, ενώ τα υπόλοιπά της bit παίρνουν την τιμή ‘0’ όταν εκτελείται σύγκριση.

Τέλος, για την πράξη της ολίσθησης (Shift) μπορούμε να πούμε πως υπάρχουν πολλοί τρόποι για να την υλοποιήσει κανείς σε μια αριθμητική λογική μονάδα (ALU). Καταρχήν, μπορούμε να χρησιμοποιήσουμε έναν καταχωρητή ολίσθησης σε συνδυασμό με έναν μετρητή. Ο καταχωρητής θα φορτώνει παράλληλα τον αριθμό που πρέπει να ολισθήσει και θα εκτελει την ολίσθηση μέχρι ο μετρητής, στον οποίο έχουμε φορτώσει παράλληλα τον αριθμό που μας λέει πόσες ολισθήσεις θα γίνουν (Shift Amount ή shamt), να φτάσει στην τιμή ‘0’. Ένας άλλος τρόπος είναι να χρησιμοποιηθούν πολυπλέκτες στις εισόδους των οποίων θα συνδεθούν κατάλληλα τα ψηφία του αριθμού που θα ολισθαίνει, έτσι ώστε να βγάζουν στην έξοδό τους τον αριθμό ολισθημένο όσες φορές λέει το κοινό σήμα επιλογής τους.

## 1.5 ΕΡΓΑΛΕΙΟ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗΣ QUARTUS II ΤΗΣ ALTERA

Το εργαλείο - λογισμικό σχεδίασης QUARTUS II είναι ένα προγραμματιζόμενο λογισμικό σχεδιασμού που παράγεται απο την ALTERA και αποτελεί ουσιαστικά μια λογική συσκευή. Το QUARTUS II επιτρέπει την ανάλυση και την σύνθεση σχεδίων HDL γλωσσών, επιτρέπει την συγκέντρωση σχεδιών τα οποία είναι σε θέση να αναλύση και να μας δώσει αποτελέσματα. Ακόμα, εξετάζει RTL διαγράμματα, κάνει

προσομοίωση του σχεδίου μας και περιλαμβάνει την εφαρμογή της VHDL και της VERILOG για την περιγραφή του υλικού δίνοντα μας με αυτόν τον τρόπο την οπτική επεξεργασία των λογικών κυκλωμάτων και την προσομοίωση τους. Επίσης, είναι εύχρηστο και με μεγάλο πλουραλισμό επιλογών. Υπάρχουν διάφορες εκδόσεις του λογισμικού ανάλογα με τις δυνατότητες του κάθε ηλεκτρονικού υπολογιστή, οι οποίες βρίσκονται όπως και το λογισμικό στην ηλεκτρονική διεύθυνση της εταιρείας www.altera.com. Τέλος για να κατεβάσουμε και να εγκαταστήσουμε το λογισμικό θα πρέπει να πάμε στο Download Center (www.altera.com/download) της ιστοσελίδας, αφού πρώτα δημιουργήσουμε έναν προσωπικό λογαριασμό.

Χαρακτηριστικά το λογισμικό QUARTUS II περιλαμβάνει: Ένα SOPC Builder, ένα εργαλείο λογισμικού που εξαλείφει εργασίες ολοκλήρωσης, ένα χειροκίνητο σύστημα με αυτόματη δημιουργία και λογική διασύνδεση για την δημιουργία ενός testbench που θα ελέγχει την λειτουργικότητα Qsys, ένα εργαλείο του συστήματος ολοκλήρωσης που είναι η επόμενη γενιά των SOPC Builder. Χρησιμοποιεί μια νέα αρχιτεκτονική που λέγεται FPGA on chip η οποία διπλασιάζει την απόδοση Fmax εναντίον του SOPC Builder, SoCEDS, ένα σύνολο εργαλείων ανάπτυξης βοηθητικά προγράμματα, run – time λογισμικό και παραδείγματα εφαρμογών για να μας βοηθήσει να αναπτύξουμε λογισμικό για SoC FPGA και ενσωματωμένα συστήματα. Επίσης, περιλαμβάνει, DSP Builder, ένα εργαλείο που είναι ουσιαστικά μια «γέφυρα» μεταξύ του εργαλείου Matlab / Simulink και του λογισμικού QUARTUS II, έτσι δίνει την δυνατότητα στους σχεδιαστές των FPGA να έχουν στην διάθεση τους όλους τους αναπτυξιακούς αλγορίθμους μαζί με τις δυνατότητες ελέγχου του εργαλείου Matlab / Simulink ταυτόχρονα σε επίπεδο συστήματος. Καταληκτικά, μας παρέχει και μαι εξωτερική εργαλειοθήκη για διεπαφή μνήμης η οποία προσδιορίζει τα θέματα βαθμονόμησης και μετρά τα περιθώρια βελτίωσης για κάθε σήμα DQS.

### 1.5.1 ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΕΤΑΙΡΕΙΑ ALTERA

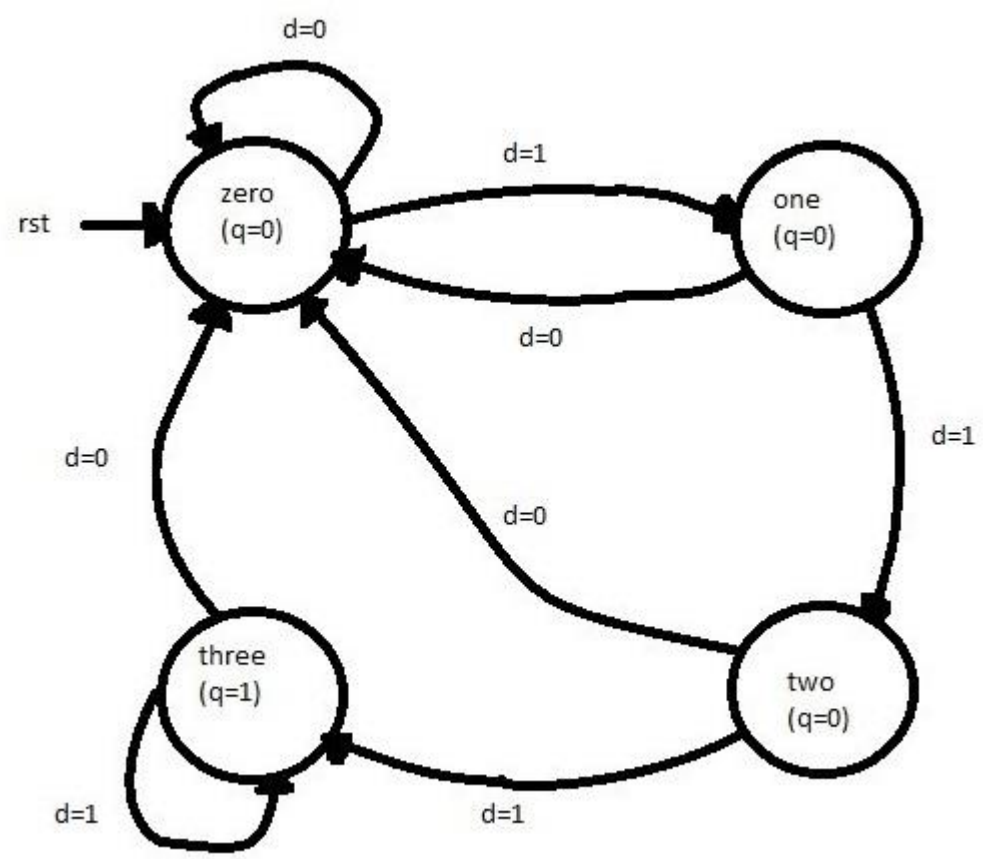
Η ALTERA ιδρύθηκε το 1983 απο τους Robert Hartmann, Michael Magranet, Paul Newhagen και Jim Sansbury οι οποίοι ήταν οι οραματιστές εκείνης της εποχής και εκμεταλεύτηκαν μια έρευνα της ημέρας για να ακολουθήσει η ίδρυση της εταιρείας. Πίστευαν ότι οι πελάτες ημιαγωγών θα επωφεληθούν απο ένα προγραμματιζόμενο απο τον χρήστη τυποποιημένο προϊον με συστοιχίες πυλών. Για την αντιμετώπιση αυτών των αναγκών της αγοράς, οι ιδρυτές της ALTERA πρωτοστάτησαν και έφτιαξαν την πρώτη επαναπρογραμματιζόμενη λογική διάταξη, το EP300, δίνοντας ζωή σε ένα εντελώς νέο τμήμα της αγοράς σε ημιαγωγούς. Αυτή η νέα, ευέλικτη λύση κατάφερε να νικήσει παραδοσιακά τυποποιημένα προϊόντα στην αγορά και να ξεκινήσει την φήμη της (η Altera) ώς ηγέτη ημιαγωγών καινοτομίας. Η ALTERA βρίσκεται στην πρώτη γραμμή της τεχνολογικής καινοτομίας, παρέχοντας στους πελάτες της προγραμματιζόμενες λύσεις για ηλεκτρονικά συστήματα που διαμορφώνουν τον σύγχρονο κόσμο μας. Με έδρα στη Silicon Valley, στο San Jose της καλιφόρνιας των Ηνωμένων Πολιτειών της Αμερικης (Η.Π.Α) έχει αναλάβει τον εφοδιασμό της βιομηχανίας που έχει σχέση με τεχνολογίες διεργασιών, IP πυρήνες και εργαλεία ανάπτυξης για περισσότερα απο 30 χρόνια. Η καινοτόμα νοοτροπία της

σε συνδυασμό με την τεχνολογική υπεροχή και την επιχειρηματική της αριστεία την καθιστούν μία κερδοφόρα επιχείρηση η οποία κερδίζει συνεχώς έδαφος στις αγορές.

Η ALTERA CORPORATION έχει την παροχή κορυφαίων λύσεων για τους πελάτες της μετα την επινόηση της πρώτης πρωγραμματιζόμενης λογικής συσκευής του κόσμου όπως αναφέραμε και παραπάνω. Σήμερα, οι περισσότεροι απο 3.000 εργαζόμενοι σε 19 χώρες παρέχουν ακομα πιο ευφυείς λύσεις προσαρμοσμένες στην λογική που περιλμβάνουν τα FPGA, EO και CPLDs. Έχει ένα ευρύ χαρτοφυλάκιο με λύσεις που καλύπτουν ένα ευρύ φάσμα προκλήσεων σε επίπεδο συστήματος, συμπεριλαμβανομένων των επιδόσεων κατανάλωσης ισχύος, το συνολικό κόστος, του χρόνου διάθεσης στην αγορά και την παραγωγικότητα της ομάδας σχεδιασμού. Τα προϊόντα της ALTERA χρησιμοποιούνται απο κορυφαίες εταιρείες σε πολλές διαφορετικές βιομηχανίες, συμπεριλαμβανομένης της αυτοκινητοβιομηχανίας, των ηλεκτρονικών υπολογιστών και αποθήκευσης, των καταναλωτών, των επικοινωνιών, της δικτύωσης, το cloud computing, της άμυνας, της ιατρικής, καθώς και σε ασύρματα και ενσύρματα δίκτυα. Τα προϊόντα της ALTERA είναι: το Stratix το οποίο ανήκει στην οικογένεια των FPGA και EO και χρησιμοποιείται για την επεξεργασία δεδομένων και αλγορίθμων επιτάχυνσης, το Arria το οποίο επίσης ανήκει στην οικογένεια των FPGA και EO και χρησιμοποιείται σε συστήματα υψηλών επιδόσεων, το Max της οικογένειας FPGA και CPLDs το οποίο είναι ένα μη-πτητικό σύστημα, το single-chip που είναι ένα σύστημα μεγάλου όγκου, προϊόντα ενέργειας που υποστηρίζουν FPGAs, EO, CPLDs καθώς και εργαλεία ανάπτυξης software, IP και σχέδια ανφοράς για εφαρμογές.

# ΚΕΦΑΛΑΙΟ 2ο

## 2.1 ΔΙΑΓΡΑΜΜΑ ΜΗΧΑΝΗΣ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ (FSM)



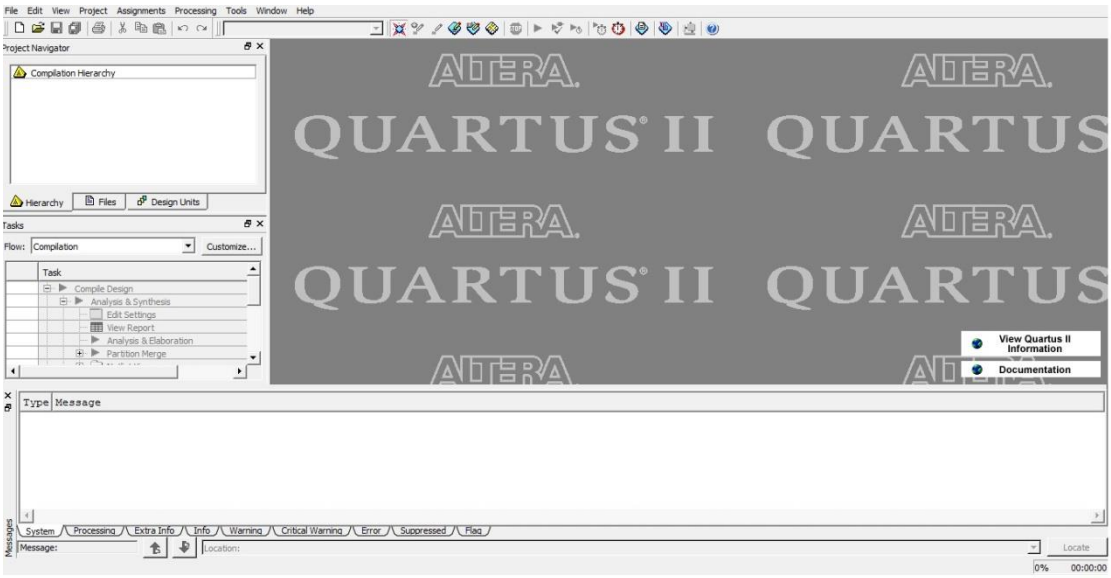

**Εικόνα 13: Διάγραμμα μηχανής πεπερασμένων καταστάσεων**

Στο διάγραμμα αυτό βλέπουμε μια μηχανή πεπερασμένων καταστάσεων που η ιδιότητα της είναι να ανιχνεύει αλφαριθμητικές παραστάσεις. Το κύκλωμα αυτό δέχεται μια σειριακή δέσμη από bit. Οι καταστάσεις του κυκλώματος έχουν τεθεί έτσι ώστε ανάλογα με το bit που θα εισέλθει να ενεργοποιείται και μία κατάσταση. Για παράδειγμα η πρώτη κατάσταση στην οποία βρισκόμαστε είναι η zero. Αν το πρώτο bit που θα εισέλθει είναι ‘0’ τότε η κατάσταση θα συνεχίσει να είναι ενεργή. Αν όμως το bit που θα εισέλθει είναι ‘1’ τότε η κατάσταση που θα ενεργοποιηθεί είναι η one. Έπειτα αν το επόμενο bit είναι το ‘1’ τότε αυτόματα θα ενεργοποιηθεί η κατάσταση two, αλλιώς αν δοθεί ‘0’ θα ενεργοποιηθεί ξανά η κατάσταση zero. Αν όμως μέσα στα bit που εισέρχονται ανιχνευθούν τρία διαδοχικά ‘1’ τότε θα δοθεί η τιμή ‘1’ στην έξοδο (q=1) και θα τερματίσει να λειτουργεί το κύκλωμα.

## 2.2 ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟΣ ΤΡΟΠΟΣ ΥΛΟΠΟΙΗΣΗΣ

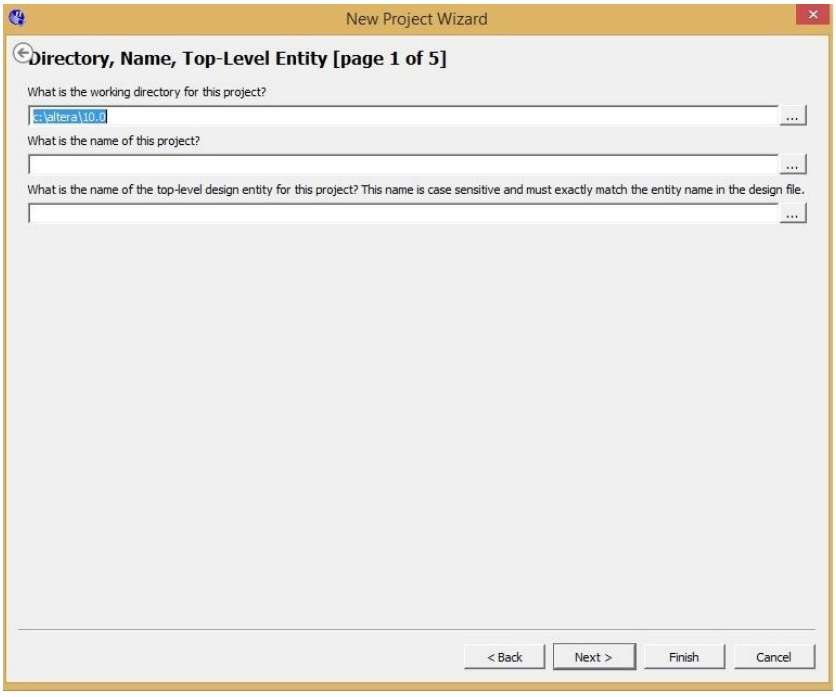
Για να δημιουργήσουμε ένα καινούργιο project στο πρόγραμμα QUARTUS II πρέπει να εκτελέσουμε κάποια βήματα. Αυτά τα βήματα που πρέπει να εκτελεστούν είναι τα ακόλουθα :

**Βήμα 1:** Ανοίγουμε το QUARTUS II το οποίο έχουμε εγκαταστήσει στο υπολογιστή μας. Στην επόμενη εικόνα βλέπουμε τι μας εμφανίζεται ανοίγοντας το πρόγραμμα.



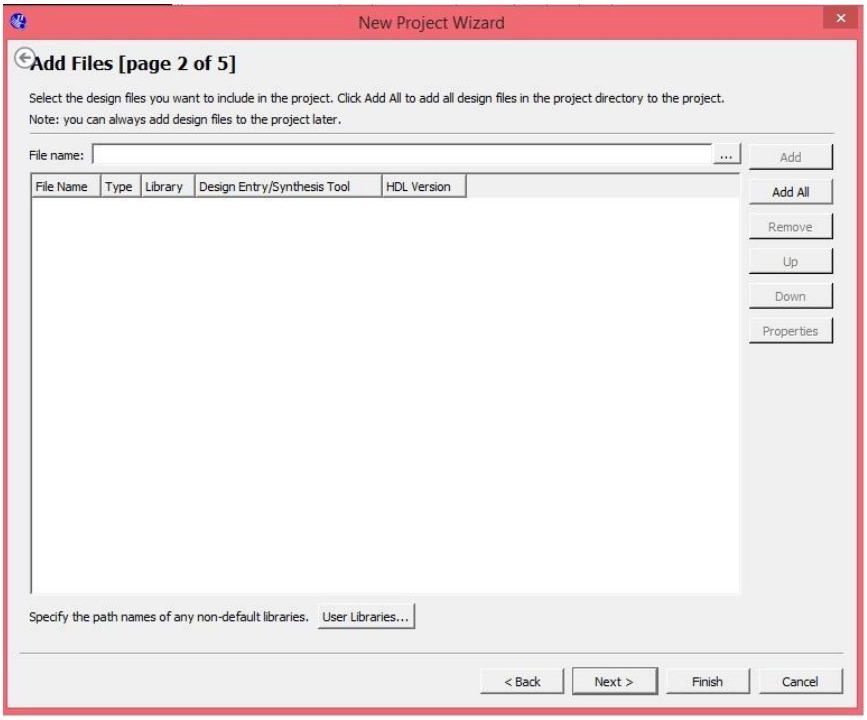

**Εικόνα 14: Αρχική εικόνα του προγράμματος QUARTUS II**

**Βήμα 2:** Από το μενού επιλέγουμε **File-New Project Wizard** αυτό γίνεται για να δημιουργήσουμε το έργο μας πάνω στο λογισμικό. Και έπειτα μας εμφανίζετε το ακόλουθο παράθυρο.New Project Wizard:Directory,Name,Top-Level όπου δίνουμε το όνομα του καταλόγου όπου θα αποθηκευτεί το έργο μας, έπειτα ονομάζουμε το έργο μας με όποιο όνομα εμείς επιθυμούμε.



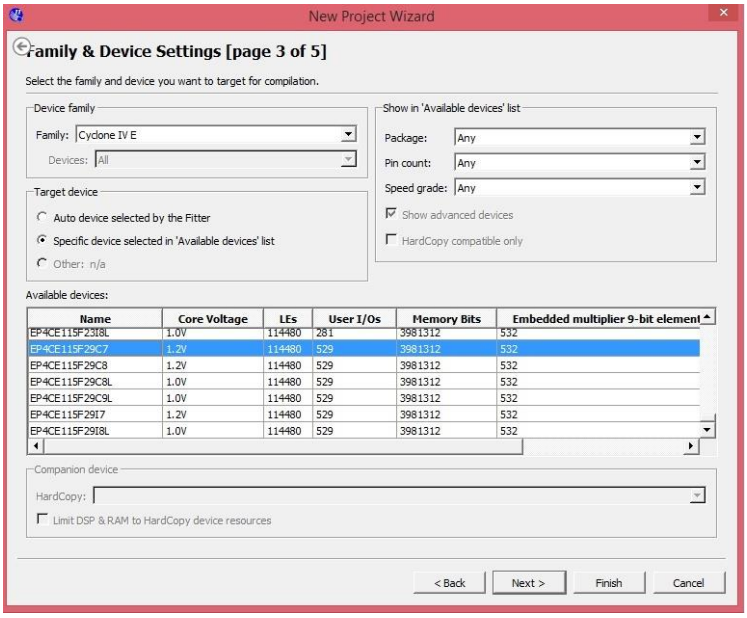
**Εικόνα 15: Πρώτο βήμα για δημιουργία Project**

**Βήμα 3:** Στο επόμενο παράθυρο με όνομα New Project Wizard:Add Files ΔΕΝ συμπληρώνουμε τίποτα(διότι είναι το πρώτο μας έργο) και πάμε στην επόμενη σελίδα.



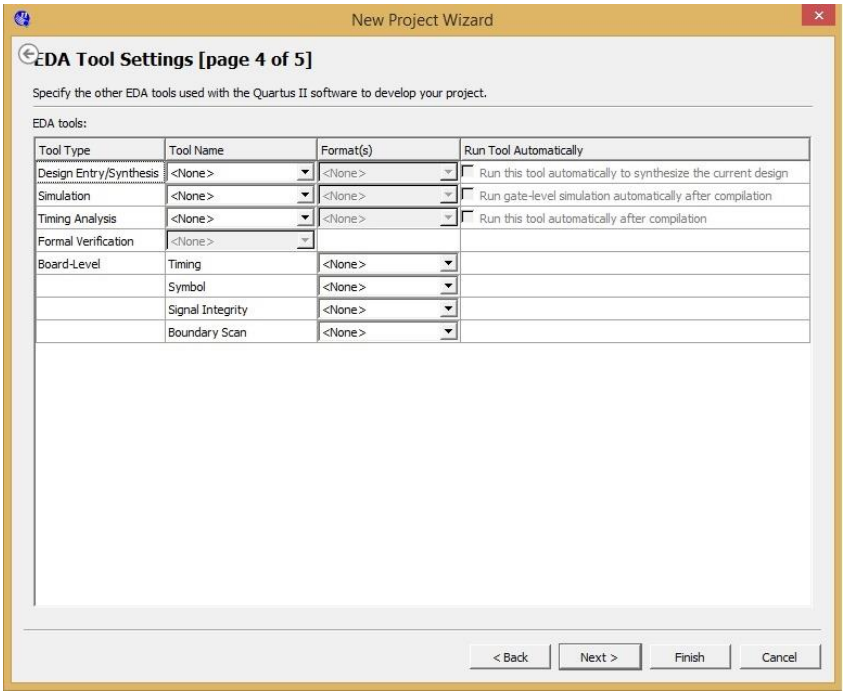

**Εικόνα 16: Δεύτερο βήμα για δημιουργία Project**

**Βήμα 4:** Στο επόμενο παράθυρο επιλέγουμε στο **Device family** και πιο συγκεκριμένα στο **family** την κατηγορία Cyclone IV E και ως **Device** τον τύπο EP4CE115F29C7



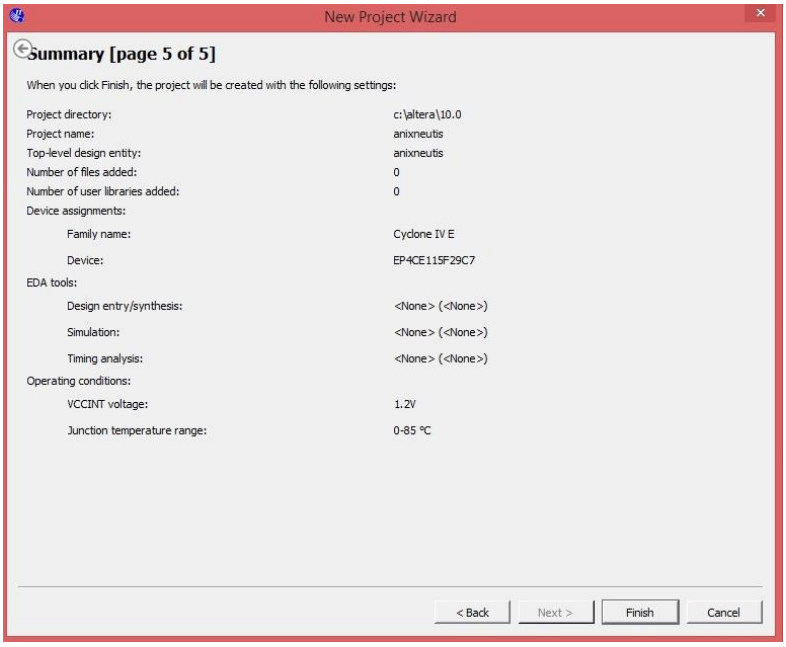

**Εικόνα 17: Τρίτο βήμα για δημιουργία Project**

**Βήμα 5:** Στην επόμενη σελίδα New Project Wizard:EDA Tool Settings ΔΕΝ συμπληρώνουμε τίποτα (δεν είναι απαραίτητο για το κύκλωμα μας) και πάμε στην επόμενη.




**Εικόνα 18: Τέταρτο βήμα για δημιουργία Project**

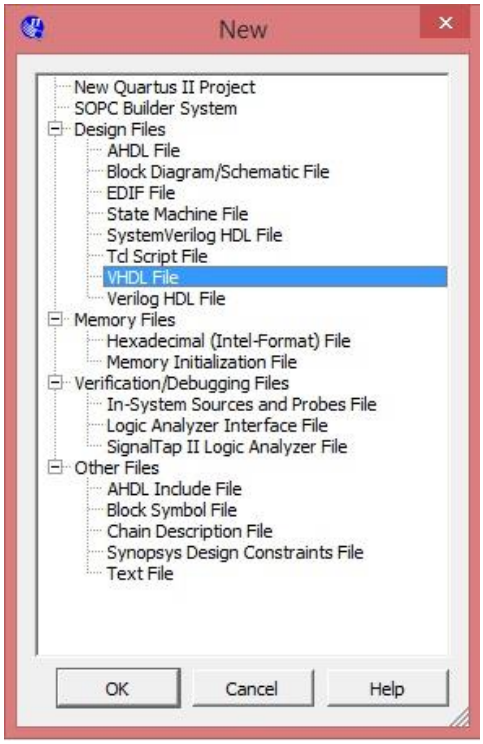
**Βήμα 6:** Στην τελευταία σελίδα New Project Wizard:Summary γίνεται μια ανασκόπηση των παραπάνω επιλογών.




**Εικόνα 19: Πέμπτο βήμα για δημιουργία Project**

Αφού ολοκληρώσαμε το στάδιο δημιουργίας του project μας ,στη συνέχεια δημιουργούμε και το αρχείο με το κώδικα υλοποίησης στην γλώσσα VHDL. Για να γίνει αυτό κάνουμε τα εξής:

Επιλέγουμε το μενού **Files** και στη συνέχεια **New** και έπειτα την καρτέλα **Design Files** και από εκεί επιλέγουμε το **VHDL File**.



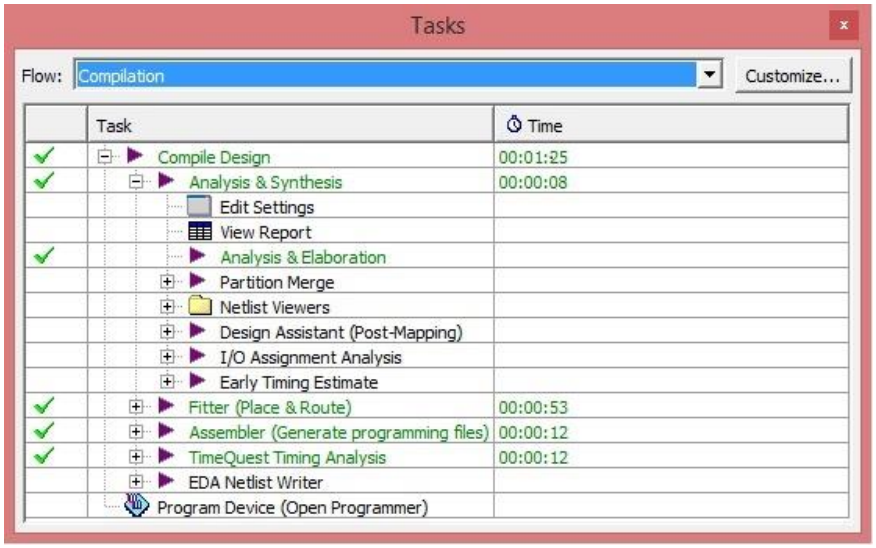

**Εικόνα 20: Δημιουργία κειμενογράφου με τύπο VHDL File**

Έτσι μας εμφανίζεται ο κειμενογράφος για να πληκτρολογήσουμε τον κώδικα μας. Στον χώρο αυτό πληκτρολογούμε τον κώδικα και έπειτα επιλέγουμε **start compilation** για να γίνει προσομοίωση του κώδικα μας για να δούμε αν υπάρχουν τυχόν σφάλματα λογικά ή συντακτικά.

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3 ENTITY anixneutis IS port(d,clk,rst:IN BIT;
 4 |                  q:OUT BIT);
 5 END anixneutis;
 6 ARCHITECTURE my_arch OF anixneutis IS TYPE state IS (zero,one,two,three);
 7 SIGNAL pr_state,nx_state:state;
 8 BEGIN
 9 PROCESS(rst,clk)
10 BEGIN
11 IF (rst='1') THEN
12 pr_state <= zero;
13 ELSIF (clk'EVENT AND clk='1') THEN
14 pr_state <= nx_state;
15 END IF;
16 END PROCESS;
17 PROCESS (d,pr_state)
18 BEGIN
19 CASE pr_state IS
20 WHEN zero=>
21 q <= '0';
22 IF (d='1') THEN nx_state <=one;
23 ELSE nx_state <= zero;
24 END IF;
25 WHEN  one =>
26 q <= '0';
27 IF (d='1') THEN nx_state <= two;
28 ELSE nx_state <= zero;
29 END IF;
30 WHEN two =>
31 q <= '0';
32 IF (d='1') THEN nx_state <= three;
33 ELSE nx_state <=zero;
34 END IF;
35 WHEN three =>
36 q <= '1';
37 IF (d='0') THEN nx_state <=zero;
38 ELSE nx_state <= three;
39 END IF;
40 END CASE;
41 END PROCESS;
42 END my_arch;
```

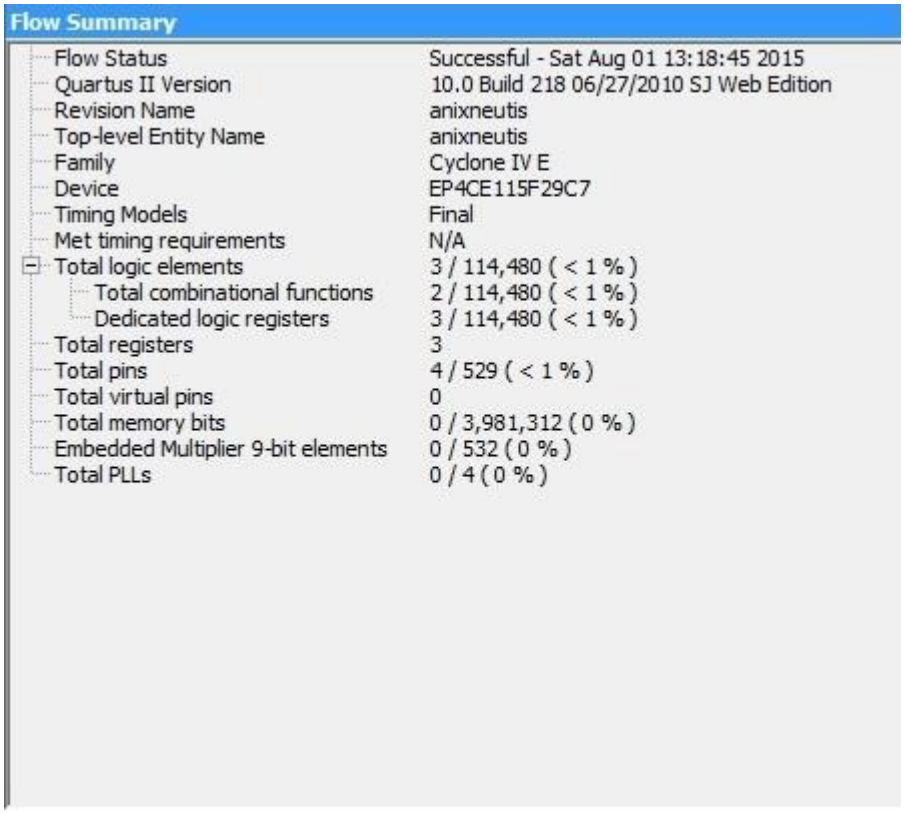
**Εικόνα 21: Ολοκληρωμένος ο κώδικας μας σε γλώσσα VHDL**

Κατά την προσομοίωση αν δεν υπάρχει κάποιο σφάλμα τότε το πρόγραμμα θα μας εμφανίσει την εικόνα που ακολουθεί.




**Εικόνα 22: Παράθυρο εμφάνισης σφαλμάτων κατά την προσομοίωση του κώδικα**

Στην επόμενη εικόνα βλέπουμε διάφορα στοιχεία τα οποία απαρτίζουν το project μας. Βλέπουμε την ημερομηνία που δημιουργήσαμε το project το όνομα που δώσαμε σε αυτό και γενικά ότι τροποποιήσεις κάνα με στα βήματα της δημιουργίας του project.




**Εικόνα 23: Περιληπτικά τα δεδομένα που χρησιμοποιήσαμε στο project**

Αυτά ήταν τα βήματα για την δημιουργία ενός νέου Project στο πρόγραμμα QUARTUS II.

### 2.2.1 ΒΙΒΛΙΟΘΗΚΕΣ

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

**Εικόνα 24: Βιβλιοθήκες για την γλώσσα VHDL**

Όταν δημιουργούμε ένα μπλοκ κώδικα πρέπει να χρησιμοποιήσουμε κάποιες βιβλιοθήκες μέσα από τις οποίες λειτουργούν οι δεσμευμένες λέξεις του κώδικα μας. Οι βιβλιοθήκες πάντα γράφονται στην αρχή του κώδικα. Όπως για παράδειγμα αν κάνουμε χρήση μαθηματικών εξισώσεων θα πρέπει να χρησιμοποιήσουμε και άλλη μία βιβλιοθήκη ειδική για μαθηματικές εξισώσεις.

Στην δική μας περίπτωση χρησιμοποιήσαμε την ποιο γνωστή βιβλιοθήκη που χρησιμοποιείτε παντού στους κώδικες VHDL. Η βιβλιοθήκη που χρησιμοποιήσαμε είναι η ieee η οποία προέρχεται από το ινστιτούτο IEEE που το 1987 έκανε πρότυπη γλώσσα υλοποίησης την γλώσσα VHDL. Με αυτή την βιβλιοθήκη ουσιαστικά λειτουργούν οι πιο απλές και βασικές εντολές του κώδικα.

Για να θέσουμε την βιβλιοθήκη γράφουμε **LIBRARY ieee;**. Έπειτα θέτουμε το πακέτο δεδομένων που θα χρησιμοποιήσουμε. Η εντολή για το πακέτο δεδομένων είναι **USE ieee.std_logic_1164.all;** και έτσι οι εντολές που γράφουμε κάνουν μία συγκεκριμένη δουλεία μέσα στον κώδικα μας, η οποία επιτρέπει να εκτελεστούν οι λογικές πράξεις αλλά όχι η αριθμητικές πράξεις.

### 2.2.2 ΟΝΤΟΤΗΤΑ

```
3   ENTITY anixneutis IS port(d,clk,rst:IN BIT;
4                             q:OUT BIT);
5   END anixneutis;
```

**Εικόνα 25: Οντότητα, το δεύτερο κομμάτι του κώδικα σε γλώσσα VHDL**

Η οντότητα αποτελεί το δεύτερο μέρος του κώδικα μας. Ουσιαστικά είναι ο συνδετικός κρίκος ανάμεσα στις βιβλιοθήκες και στην αρχιτεκτονική. Σε αυτή

δηλώνουμε τις εισόδους και τις εξόδους που θα χρησιμοποιήσουμε στο τρίτο μέρος του κώδικά μας που είναι η αρχιτεκτονική. Η οντότητα ορίζεται γράφοντας **Entity *όνομα_οντότητας* is** και έπειτα τοποθετούμε τις εισόδους και τις εξόδους που θα χρησιμοποιήσουμε. Στην συγκεκριμένη περίπτωση το όνομα της οντότητας που χρησιμοποιήσαμε είναι anixneutis γιατί έτσι ονομάσαμε και το project μας αλλά έτσι ονομάσαμε και το όνομα του αρχείου που δημιουργήσαμε και αποθηκεύσαμε τον κώδικα μας.

Έπειτα δηλώνουμε τις εισόδους και τις εξόδους. Αυτό το κάνουμε γράφοντας **port (*όνομα εισόδου* ή *όνομα εξόδου*: IN (άν πρόκειται για είσοδο) ή OUT (άν πρόκειται για έξοδο).** Στην περίπτωση μας έχουμε τρεις εισόδους οι οποίες η κάθε μία έχει μήκος 1 bit και έχουμε και μία έξοδο που και αυτή έχει μήκος 1 bit. Έτσι δηλώσαμε τις εισόδους και την έξοδό μας και είμαστε έτοιμοι να τις χρησιμοποιήσουμε στο κύριο μέρος του κώδικά μας που είναι η αρχιτεκτονική.

### 2.2.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Η αρχιτεκτονική είναι το τρίτο και σπουδαιότερο κομμάτι ενός κώδικα σε γλώσσα VHDL. Με αυτό το κομμάτι ολοκληρώνετε ο κώδικας μας. Σε αυτό το σημείο προγραμματίζουμε τι θα κάνει το Project μας. Ο τρόπος υλοποίησης της αρχιτεκτονικής είναι Architecture *όνομα_αρχιτεκτονικης* of *όνομα_οντότητας* is και έπειτα δηλώνουμε τον τύπο και τις καταστάσεις που χρησιμοποιούμε. Όνομα αρχιτεκτονικής στην δική μας περίπτωση είναι το **my_arch** και όνομα οντότητας **anixneutis**. Έπειτα, γράφουμε **BEGIN** και έτσι γράφουμε αναλυτικά τον κώδικα που θέλουμε. Για να κλείσουμε την αρχιτεκτονική γράφουμε **END *όνομα_αρχιτεκτονικης*** που στην προκειμένη περίπτωση είναι **END my_arch** και έτσι κλείνουμε το κομμάτι τις αρχιτεκτονικής.

Η λειτουργία του κώδικα που γράψαμε είναι απλή. Θέτουμε σαν αρχική κατάσταση την **zero.** Αν το πρώτο bit από την σειριακή ακολουθία που εισέρχεται στην μηχανή μας είναι το ‘0’ τότε παραμένει στην ίδια κατάσταση, αλλιώς ενεργοποιείται η κατάσταση **one.** Έπειτα, αν η τρέχων κατάσταση είναι η **one** και το επόμενο bit είναι το ‘0’ τότε ξαναγυρίζει στην κατάσταση **zero**, αλλιώς πάει στην κατάσταση **two.** Στην συνέχεια, αν η τρέχων κατάσταση είναι η **two** και το bit που εισέρχεται είναι το ‘0’ τότε πάει στην αρχική κατάσταση που είναι η **zero**, αλλιώς αν εισέλθει το ‘1’ τότε πάει στην κατάσταση **three**. Αν η τρέχων κατάσταση είναι η **three** και εισέλθει το ‘1’ τότε παραμένει στην ίδια κατάσταση που είναι και η τελική, αλλιώς πάει στην κατάσταση την αρχική που είναι η **zero.**

```
 6  ARCHITECTURE my_arch OF anixneutis IS TYPE state IS (zero,one,two,three);
 7   SIGNAL pr_state,nx_state:state;
 8  BEGIN
 9  PROCESS(rst,clk)
10   BEGIN
11  IF (rst='1') THEN
12   pr_state <= zero;
13  ELSIF (clk'EVENT AND clk='1') THEN
14   pr_state <= nx_state;
15   END IF;
16   END PROCESS;
17  PROCESS (d,pr_state)
18   BEGIN
19  CASE pr_state IS
20   WHEN zero=>
21   q <= '0';
22  IF (d='1') THEN nx_state <=one;
23  ELSE nx_state <= zero;
24   END IF;
25   WHEN  one =>
26   q <= '0';
27  IF (d='1') THEN nx_state <= two;
28  ELSE nx_state <= zero;
29   END IF;
30   WHEN two =>
31   q <= '0';
32  IF (d='1') THEN nx_state <= three;
33  ELSE nx_state <=zero;
34   END IF;
35   WHEN three =>
36   q <= '1';
37  IF (d='0') THEN nx_state <=zero;
38  ELSE nx_state <= three;
39   END IF;
40   END CASE;
41   END PROCESS;
42    END my_arch;
```

**Εικόνα 26: Αρχιτεκτονική, το τρίτο κομμάτι του κώδικα σε γλώσσα VHDL**

# ΚΕΦΑΛΑΙΟ 3ο

## 3.1 ΚΥΚΛΩΜΑ ΑΡΙΘΜΗΤΙΚΗΣ ΛΟΓΙΚΗΣ ΜΟΝΑΔΑΣ – ALU

Το κύκλωμα ή chip της αριθμητικής λογικής μονάδας (ALU),όπως βλέπουμε στο παρακάτω διάγραμμα αποτελείται απο:

- Δύο εισόδους, την είσοδο **a** και την είσοδο **b**, όπου η κάθε μια αντίστοιχα αντιπροσοπεύει έναν ακέραιο 8 - bit αριθμό. Το έυρος του αριθμού καθορίζεται απο τον χρήστη (στην περίπτωση μας 8 – bit) και ποικίλει

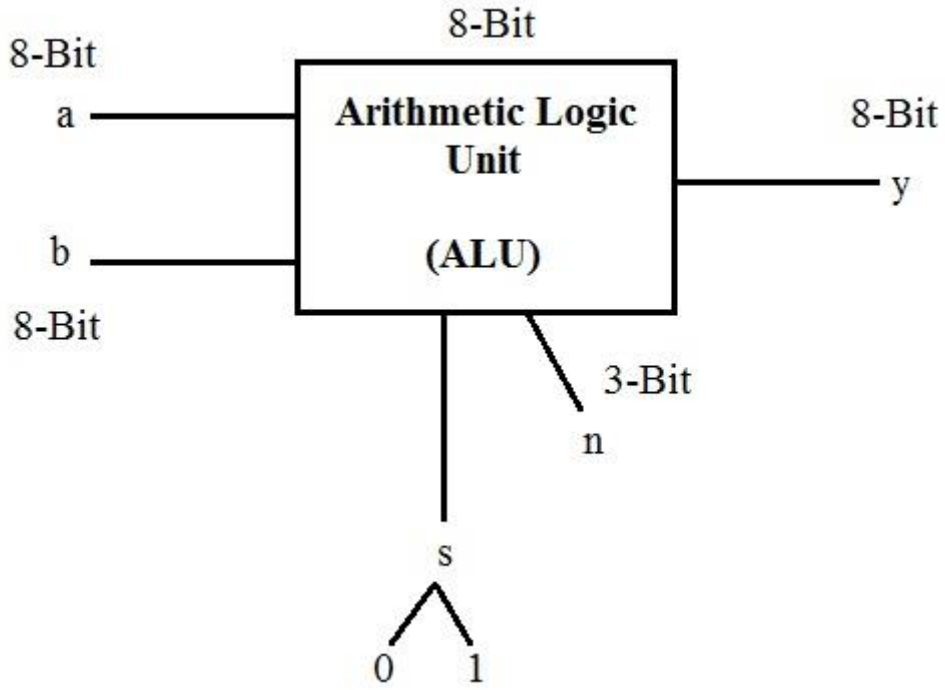
ανάλογα με τις ανάγκες και τις προτιμήσεις του καθενώς, στην προκειμένει περίπτωση προτιμήθηκε να είναι ένας 8 – bit αριθμός.

- Μια είσοδο **S** η οποία διακρίνει αν θα κάνουμε αριθμητικές πράξεις ή λογικές πράξεις ανάμεσα στους δύο 8 – bit αριθμούς a και b. Άν η είσοδος **S** λαμβάνει την τιμή 0 τότε η αριθμητική λογική μονάδα θα πραγματοποιήσει αριθμητική πράξει ανάμεσα στους δύο αριθμούς, αλλιώς αν η είσοδος **S** λαμβάνει την τιμή 1 τότε η αριθμητική λογική μονάδα θα πραγματοποιήσει λογική πράξει ανάμεσα στους δύο αριθμούς.

- Μια είσοδο **n** όπου με την βοήθεια της θα διακρίνουμε ποιά πράξη θα πραγματοποιηθεί απο τις δύο κατηγορίες πράξεων σύμφωνα με τους πίνακες των πράξεων όπως θα δούμε στην συνέχεια. Η συγκεκριμένη είσοδος διαφέρει σε έυρος απο τις προηγούμενες δύο εισόδους (την είσοδο **a** και την είσοδο **b**) και σε αντίθεση με τις άλλες δύο είναι 3 – bit αντί για 8 – bit. Αυτό οφείλεται καθαρά για λόγους ευελιξίας και ευχρηστίας της συγκεκριμένης είσοδου και γενικότερα της αριθμητικής λογικής μονάδας και δεν επειρεάζει το αποτέλεσμά μας όπως θα δούμε και θα αναφερθούμε παρακάτω.

- Μια έξοδο **y** στην οποία θα εμφανίζεται το αποτέλεσμα της αριθμητικής λογικής μονάδας μετά την εκτέλεση της πράξεις ανάμεσα στους δύο 8 – bit αριθμούς **a** και **b**, το οποίο αποτέλεσμα θα είναι και αυτό όπως είναι λογικό 8 – bit.

Συμπερασματικά,η ιδέα της συγκεκριμένης αριθμητικής λογικής μονάδας βασίζεται στην είσοδο δύο 8 – bit αριθμών (**a** , **b**) των διαχωρισμό της πράξης (αν θα είναι αριθμητική (**S = 0**) ή λογική πράξη (**S = 1**)) την εκτέλεση της πράξης (**n**) και την εμφάνιση του αποτελέσματος μετά την εκτέλεση ( **y** ).




**Εικόνα 27: Αριθμητική Λογική Μονάδα – ALU**

### 3.2 ΠΙΝΑΚΕΣ ΑΡΙΘΜΗΤΙΚΩΝ ΚΑΙ ΛΟΓΙΚΩΝ ΠΡΑΞΕΩΝ ΤΗΣ ALU

Στους παρακάτω πίνακες παρουσιάζεται το πλήθος και το είδος των πράξεων που έχουν την δυνατότητα να εκτελεστούν στην αριθμητική λογική μονάδα – ALU.

| **ΑΡΙΘΜΗΤΙΚΕΣ ΠΡΑΞΕΙΣ ( S=0 )** | **ΛΟΓΙΚΕΣ ΠΡΑΞΕΙΣ ( S=1)** |
|---|---|
| a + 1 Όταν n=00000001 | not Όταν n=00000001 |
| a – 1 Όταν n=00000010 | and Όταν n=00000010 |
| b + 1 Όταν n=00000011 | or Όταν n=00000011 |
| b – 1 Όταν n=00000100 | nand Όταν n=00000100 |
| a + b Όταν n=00000101 | nor Όταν n=00000101 |
| a – b Όταν n=00000110 | xor Όταν n=00000110 |

**Εικόνα 28: Πίνακες των πράξεων για την είσοδο n με 8 – bit έυρος**

Όπως διαπιστώνουμε απο τους παραπάνω δύο πίνακες το πλήθος των πράξεων που θα εκτελέσουμε στην αριθμητική λογική μονάδα – ALU, απο κάθε κατηγορία ξεχωριστά, είναι έξι ( 6 ), ενώ το είδος των πράξεων μεταξύ των δύο κατηγοριών ποικίλει. Στην πρώτη κατηγορία (Αριθμητικές Πράξεις (**S=0**)) έχουμε την πρόσθεση και την αφαίρεση της μονάδας με τον 8 – bit ακέραιο αριθμό **a** αντίστοιχα. Στην συνέχεια γίνεται η ίδια διαδικασία και για τον 8 – bit ακέραιο αριθμό **b**, δηλαδή η πρόσθεση και η αφαίρεση της μονάδας. Τέλος, πραγματοποιείται η πράξη της πρόσθεσης και της αφαίρεσης μεταξύ των δύο 8 – bit ακέραιων αριθμών **a** και **b**. Στην δεύτερη κατηγορία (Λογικές Πράξεις (**S=1**)) οι πράξεις οι οποίες εκτελούνται μεταξύ των δύο 8 – bit ακέραιων αριθμών είναι οι: not, and, or, nand, nor, xor.

Για να εκτελέσουμε ένα απο τα έξι είδη πράξεων που έχουν οι δύο κατηγορίες (Αριθμητικές Πράξεις, Λογικές Πράξεις) αντίστοιχα, θα πρέπει να δώσουμε την κατάλληλη κωδικοποίηση στην είσοδο **n** για να πραγματοποιηθεί η πράξη της αντίστοιχης κωδικοποίησης.Όπως παρατηρούμε απο τους παραπάνω πίνακες των πράξεων το έυρος της εισόδου **n** για κάθε πράξη είναι ένας 8 – bit αριθμός ( δηλαδή ένας αριθμός με οκτώ ψηφία απο ‘0’ και ‘1’ ), αλλά αν παρατηρήσουμε ακόμα καλύτερα θα διαπιστώσουμε πως σε όλες τις πράξεις τα πρώτα πέντε ψηφία (πέντε bit) του 8 – bit αριθμού είναι ΙΔΙΑ και είναι ο αριθμός ΜΗΔΕΝ ‘0’. Για τον λόγο αυτό μπορούμε να τα παραλήψουμε στην διαδικασία της κωδικοποίησης και να χρησιμοποιήσουμε ΜΟΝΟ τα τελευταία τρία ψηφία (τρία bit) απο όλο τον 8 – bit αριθμό της εισόδου **n.** Έτσι η είσοδος **n** θα μετατραπεί απο 8 – bit σε 3 – bit αριθμός και θα γίνει ποιό ευέλικτη, εύχρηστη και γρήγορη προς τον χρήστη όλη η λειτουργία της αριθμητικής λογικής μονάδας - ALU γενικότερα. Επομένως, οι παραπάνω πίνακες των πράξεων διαμορφώνονται ώς εξής:

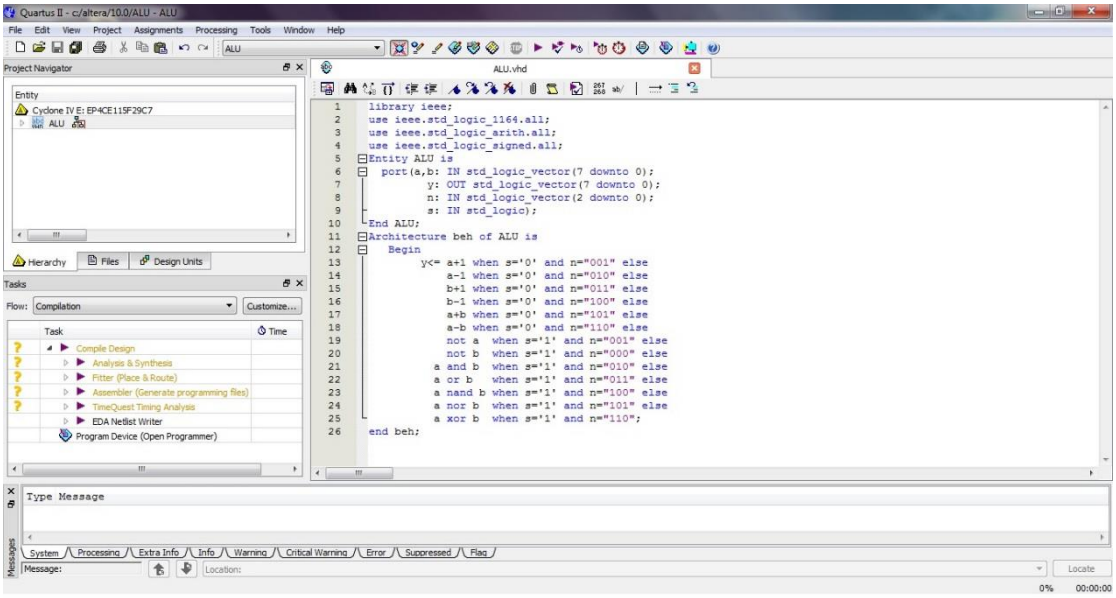
| **ΑΡΙΘΜΗΤΙΚΕΣ ΠΡΑΞΕΙΣ ( S=0 )** | **ΛΟΓΙΚΕΣ ΠΡΑΞΕΙΣ ( S=1)** |
|---|---|
| a + 1 Όταν n=001 | not Όταν n=001 |
| a – 1 Όταν n=010 | and Όταν n=010 |
| b + 1 Όταν n=011 | or Όταν n=011 |
| b – 1 Όταν n=100 | nand Όταν n=100 |
| a + b Όταν n=101 | nor Όταν n=101 |
| a – b Όταν n=110 | xor Όταν n=110 |

**Εικόνα 29: Πίνακες των πράξεων για την είσοδο n με 3 – bit έυρος**

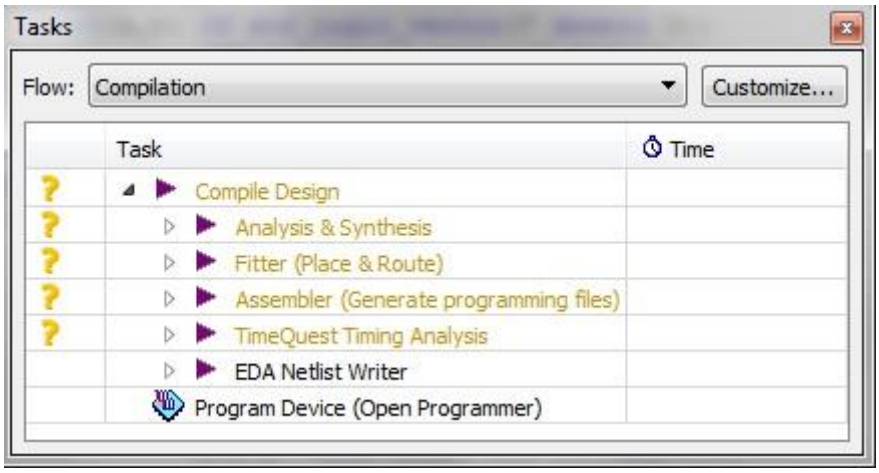
## 3.3 ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟΣ ΤΡΟΠΟΣ ΥΛΟΠΟΙΗΣΗΣ ΤΗΣ ALU

Με την βοήθεια του εργαλείου QUARTUS II θα υλοποιήσουμε στην πράξη την αριθμητική λογική μονάδα μας στην οποία έχουμε αναφερθεί παραπάνω καλύπτωντας όλες τις παραμέτρους και τις πτυχές της που θα πρέπει να γνωρίζουμε πρωτού προχωρήσουμε στο συγκεκριμένο βήμα. Όπως λέει και ο τίτλος της συγκεκριμένης παραγράφου θα προγραμματίσουμε σε κώδικα της γλώσσας VHDL για να υλοποιήσουμε την αριθμητική λογική μονάδα μας. Ουσιαστικά, πρόκειται να συνθέσουμε κώδικα της γλώσσας VHDL πάνω στον κειμενογράφο του εργαλείου QUARTUS II να τον εκτελέσουμε και εφόσον είναι σωστά συνταγμένος και δομημένος (θα μας το εμφανίσει το πρόγραμμα αφου «τρέξουμε» τον κώδικα) πλέον τότε θα είμαστε σε θέση να προσαρμόσουμε των κώδικα ο οποίος αντιπροσωπεύει την αριθμητική λογική μονάδα – ALU μας πάνω στο FPGA ( την πλακέτα ) μας. Αυτην τη δυνατότητα ( της προσαρμογής ) μας την παρέχει το συγκεκριμένο πρόγραμμα, εκτός φυσικά απο τις υπόλοιπες δυνατότητες που μας παρέχει για να έχουμε το επιθυμητό αποτέλεσμα κάθε φορά που εμείς θέλουμε, μέσω του pin planner (θα αναφερθούμε παρακάτω γι'αυτο) το οποίο είναι το FPGA μας σε εικονική μορφή πάνω στο εργαλείο QUARTUS II. Έκει δίνουμε την κατάλληλη συνδεσμολογία, όπως θα δούμε και παρακάτω, για να προσαρμόσουμε τον κώδικα μας πάνω στο FPGA και να λάβουμε το επιθυμητό αποτέλεσμα. Για να γίνει αυτο θα πρέπει πρώτα να έχουμε συνδέση το FPGA (δηλαδή την πλακέτα) στον υπολογιστή μας και να ακολουθήσουμε ξανά τα βήματα που αναφέρθηκαν και στο κεφάλαιο 2$^{ο}$ για να φτάσουμε στον κειμενογράφο του εργαλέιου μας. Αφού κάνουμε όλα τα παραπάνω είμαστε έτοιμοι να ξεκινήσουμε την σύνθεση του κώδικα μας.

Παρακάτω βλέπουμε τον ολοκληρωμένο κώδικα της αριθμητικής λογικής μονάδας – ALU, αφού έχουμε κάνει αποθήκευση (Save) το αρχείο του κώδικα αλλά, χωρίς να έχουμε πραγματοποιήσει μεταγλώττιση (compilation) των κώδικα μας. Αυτο το καταλαβαίνουμε απο το παράθυρο Tasks (εικόνα χ) το οποίο μας δείχνει ότι ο κώδικας μας είναι σε αναμονή για μεταγλώττιση με το σύμβολο ? σε κίτρινη απόχρωση.

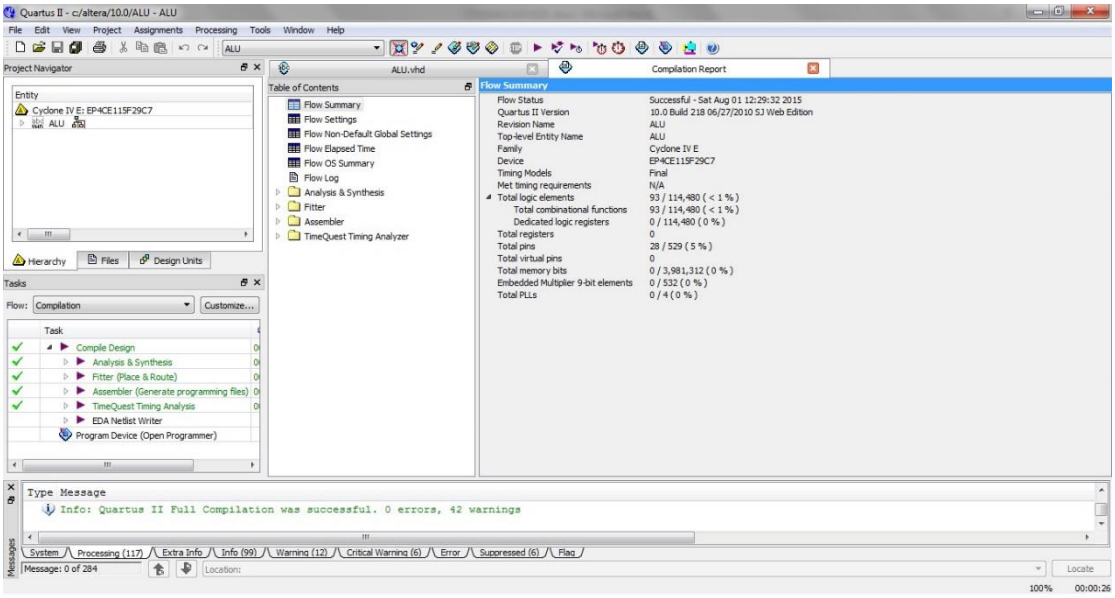



**Εικόνα 30: Ολοκληρωμένος κώδικας της ALU, χωρίς μεταγλώττιση**



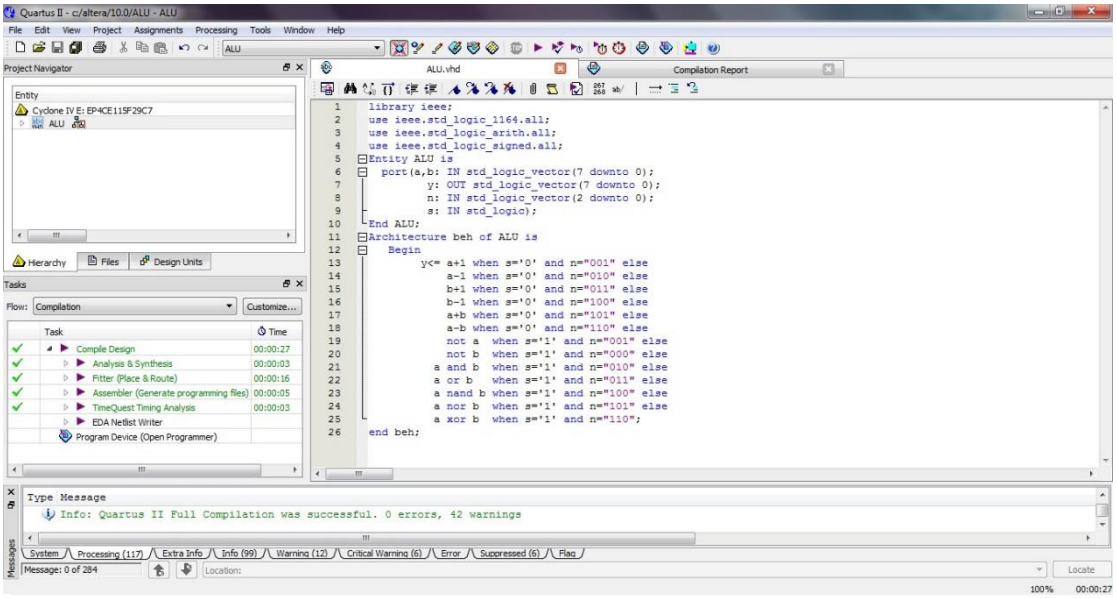

**Εικόνα 31: Παράθυρο Tasks πριν την μεταγλώττιση**

Για να πραγματοποιήσουμε την μεταγλώττιση του κώδικα μας πατάμε το κουμπί με όνομα Start Compilation το οποίο βρίσκεται πάνω στην γραμμή εργαλειών του προγράμματος και έχει την μορφή ενός μικρού τριγώνου. Αφού πατήσουμε το κουμπί Start Compilation ξεκινάει η μεταγλώττιση και απαιτούνατι μερικά δευτερόλεπτα για να ολοκληρωθεί, έπειτα εμφανίζεται το παρακάτω παράθυρο στο οποίο παρουσιάζεται μια γενική ανασκόπηση με τις ρυθμίσεις που έχουν γίνει στο έργο μας γενικότερα.

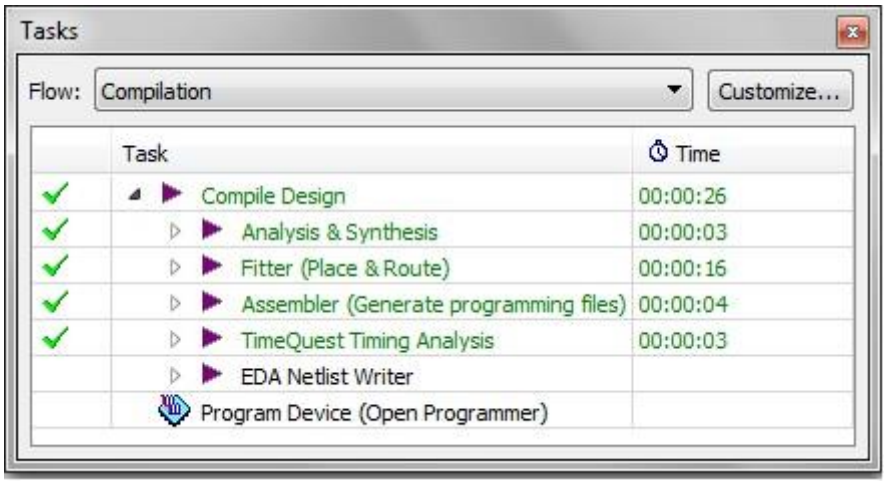



**Εικόνα 32: Παράθυρο ανασκόπησης ρυθμίσεων του έργου μας**

Για να μεταβούμε στο παράθυρο του κώδικα, έχοντας πραγματοποιήσει το στάδιο της μεταγλώττισης, κάνουμε διπλό αριστερό κλίκ στο έργο μας που έχει όνομα ALU και είναι τοποθετημένο στο παράθυρο Entity όπως βλέπουμε στην εικόνα Α. Έχωντας υλοποιήση την μεταγλώττιση του κώδικά μας ανοίγουμε το παράθυρο του κώδικα και παρατηρούμε οτι έχει επικυρωθεί η μεταγλώττιση (Compilation), όπως φένεται και στην εικόνα χ1 αλλά και απο το παράθυρο Tasks (εικόνα χ2) του οποίου ο συμβολισμός ? σε κίτρινη απόχρωση έχει μετατραπέι σε ν πράσινης απόχρωσης και καταγράφονται και οι αντίστοιχοι χρόνοι που απαιτήθηκαν για κάθε ένα τμήμα ξεχωριστά.

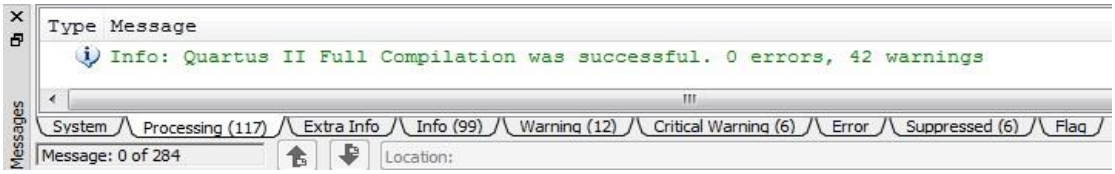


**Εικόνα 33: Ολοκληρωμένος κώδικας της ALU, με μεταγλώττιση**




**Εικόνα 34: Παράθυρο Tasks μετά την μεταγλώττιση**

Επίσης, παρατηρούμε και ένα παράθυρο μηνυμάτων (εικόνα χ3) στο οποίο καταγράφονται τα λάθη και κάποιες προειδοποιήσεις του κώδικα μας.

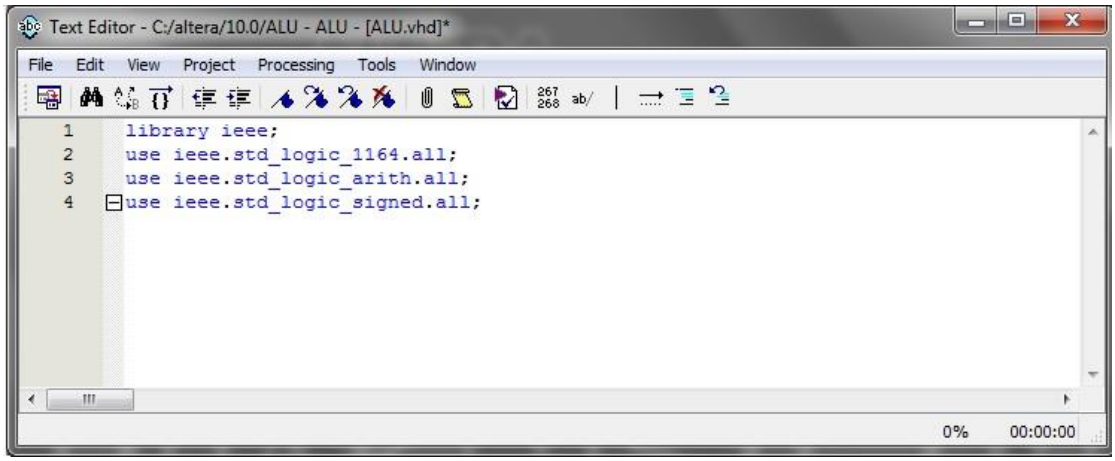



**Εικόνα 35: Παράθυρο μηνυμάτων για τυχών σφάλματα**

### 3.3.1 ΕΙΣΑΓΩΓΗ ΒΙΒΛΙΟΘΗΚΩΝ

Στο παρών κεφάλαιο θα εξετάσουμε το ένα απο τα τρία μέρη του κώδικα της αριθμητικής λογικής μονάδας – ALU το οποίο είναι η τοποθέτηση των βιβλιοθηκών, οι οποίες είναι το βασικό συστατικό για το ξεκίνημα της σύνθεσηςτου κώδικα. Αρχικά, το πρώτο πράγμα που κάνουμε είναι να δηλώσουμε την ύπαρξη βιβλιοθηκών στον κώδικα μας, και αυτο το πραγματοποιούμε γράφωντας library ieee; στην πρώτη γραμμή. Έπειτα τοποθετούμε της βιβλιοθήκες που είναι απαραίτητες οι οποίες είναι τρείς συνολικά και είναι οι εξής: 1) Η use ieee.std_logic_1164.all; η οποία επιτρέπει να εκτελεστούν οι λογικές πράξεις αλλα όχι η αριθμητικές πράξεις. 2) Η use ieee.std_logic_arith.all; με την οποία εκτελούνται οι αριθμητικές πράξεις αλλά όχι οι λογικές και τέλος 3) Η use ieee.std_logic_signed.all; η οποία ορίζει προσημασμένους

αριθμούς, δηλαδή αριθμούς με πρόσημο + ή αριθμούς με πρόσημο - ανάλογα με την περίπτωση. Το std_logic που χρησιμοποιείται είναι τύπος δεδομένων. Αξίζει να αναφέρουμε, για εγκυκλοπαιδικούς κυρίως λόγους, ότι υπάρχει και άλλη μία βιβλιοθήκη ευρέως γνωστή στην VHDL που χρησιμοποιείται και είναι η use ieee.std_logic_unsigned.all; η οποία ορίζει αριθμούς χωρίς πρόσημο + καθώς επίσης και μερικοί ακόμα τύποι δεδομένων πέραν του std_logic και είναι οι: bit, integer, real και boolean. Όλα τα παραπάνω απεικονίζωνται στην παρακάτω εικόνα χ3.

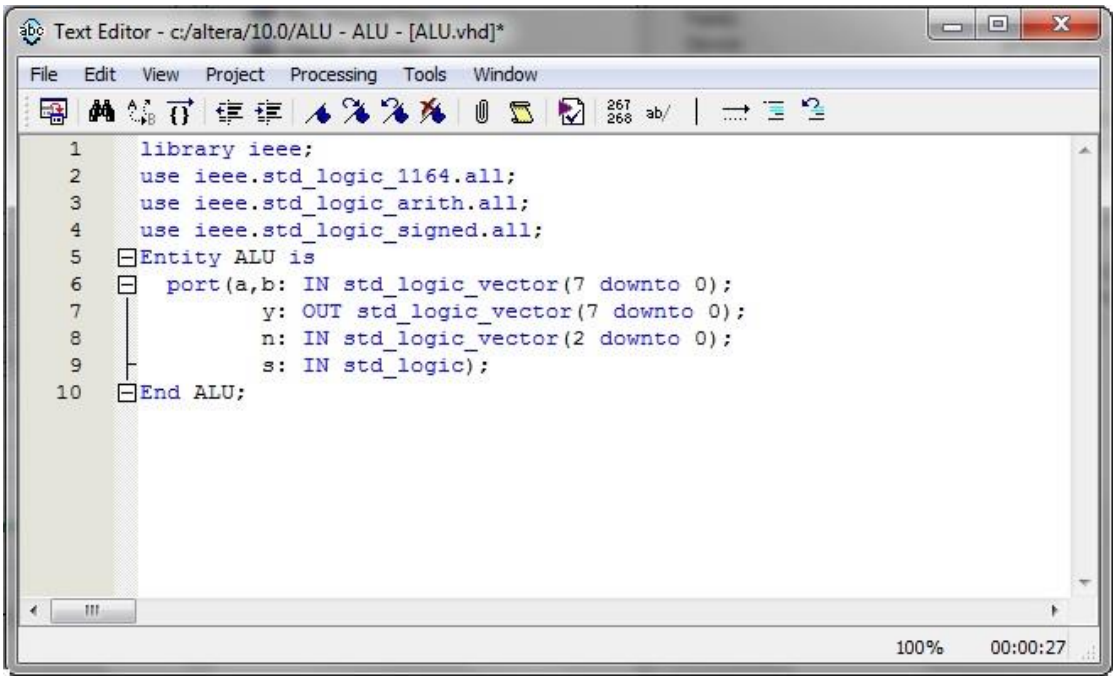



**Εικόνα 36: Εισαγωγή βιβλιοθηκών**

### 3.3.2 ΟΝΤΟΤΗΤΑ

Το δεύτερο μέρος του κώδικα της αριθμητικής λογικής μονάδα – ALU είναι η δημιουργία της οντότητας, η οποία είναι ο συνδετικός κρίκος μεταξύ των βιβλιοθκών και της αρχιτεκτονικής, την οποία θα εξετάσουμε παρακάτω. Η οντότητα τοποθετήται πάντα μετά απο τις βιβλιοθήκες και πριν απο την αρχιτεκτονική σε ένα πρόγραμμα της VHDL και χρησιμοποιείται για να δηλώσουμε τις εισόδους και τις εξόδους του κυκλώματός μας ( στην περίπτωση μας για την αριθμητική λογική μονάδα – ALU ). Γενικά, ο ορισμός της οντότητας σε ένα πρόγραμμα γίνεται γράφωντας Entity *όνομα_οντότητας* is ,όπου το *όνομα_οντότητας* θα πρέπει να είναι ίδιο με το όνομα του πρότζεκτ που έχουμε δημιουργήσει νωρίτερα (New Project Wizard) αλλά και με το όνομα του αρχείου (VHDL File) που έχουμε δημιουργήσει για να αποθηκεύσουμε τον κώδικα μας, διότι διαφορετικά κατά την διαδικασία της μεταγλώττισης (Compilation) του κώδικα το λογισμικό QUARTUS II θα το λάβει ώς συντακτικό λάθος και έπειτα θα πρέπει αναγκαστικά να αντρέξουμε είτε στο αρχείο (VHDL File) είτε στο πρότζεκτ (New Project Wizard) είτε στον κειμενογράφο (Text Editor) για να το διορθώσουμε. Στην συγκεκριμένη περίπτωση ορίζουμε την οντότητα μας γράφωντας Entity ALU is όπως βλέπουμε και στην πέμπτη γραμμή της εικόνας χ4.

Στην συνέχεια τοποθετούμε τις εισόδους και τις εξόδους του κυκλώματος μας γράφωντας port (*όνομα εισόδου* ή *όνομα εξόδου*: IN (άν πρόκειται για είσοδο) ή OUT (άν πρόκειται για έξοδο) *τύπος δεδομένων*_vector(c downto d); όπου c,d είναι τα δύο άκρα του δυανύσματος, άν το μήκος της εισόδου μας είναι 1 – bit μόνο τότε το δυάνυσμα( _vector(c downto d )παραλείπεται. Εδώ θα πρέπει να τονίσουμε την σημασία των δυανυσμάτων (vectors) στην VHDL. Χρησιμοποιούνται για να επεκτείνουμε το έυρος των αριθμών που χρησιμοποιούμε, όταν είναι μεγαλύτεροι του ενός bit και ξεκινούν απο τν θέση μηδέν (0). Για την αριθμητική λογική μονάδα γράφουμε: port (a,b: IN std_logic_vector(7 downto 0); όπου είναι οι δύο είσοδοι των 8 – bit αριθμών αντίστοιχα και ο τύπος δεδομένων τους είναι std_logic. Στην συνέχεια για να δηλώσουμε την έξοδο της αριθμητικής λογικής μονάδας – ALU γράφουμε: y: OUT std_logic_vector(7 downto 0); καθώς και για τις άλλες δύο εισόδους(έχουμε αναφέρει την λειτουργία τους στο κεφάλαιο 3.1) την n και την s οι οποίες συντάσονται ώς εξής: 1) Για την είσοδο n: n: IN std_logic_vector(2 downto 0); 2) Για την είσοδο s: s: IN std_logic); η εντολή port αντιστοιχεί και για την έξοδο y αλλά και για τις άλλες δύο εισόδους n, s αντίστοιχα, γράφεται όμως μία φορά στην οντότητα μετά την εντολή Entity. Τέλος, όταν έχουμε ολοκληρώση τις διάφορες δηλώσεις εισόδων και εξόδων μέσα στην οντότητα θα πρέπει να δηλώσουμε και τον τερμτισμό της, δηλαδή να την κλέισουμε. Αυτό γίνεται γενικά γράφωντας: End *όνομα_οντοτητας*; Το *όνομα_οντοτητας* θα πρέπει να είναι και αυτό ίδιο με εκείνο της εντολής Entity για να μην δημιουργηθεί τυχών σύγχηση, για τους λόγους που αναφέραμε παραπάνω. Στην περίπτωση μας γράφουμε: Entity ALU; και έτσι ολοκληρώνουμε και το κομμάτι της οντότητας.

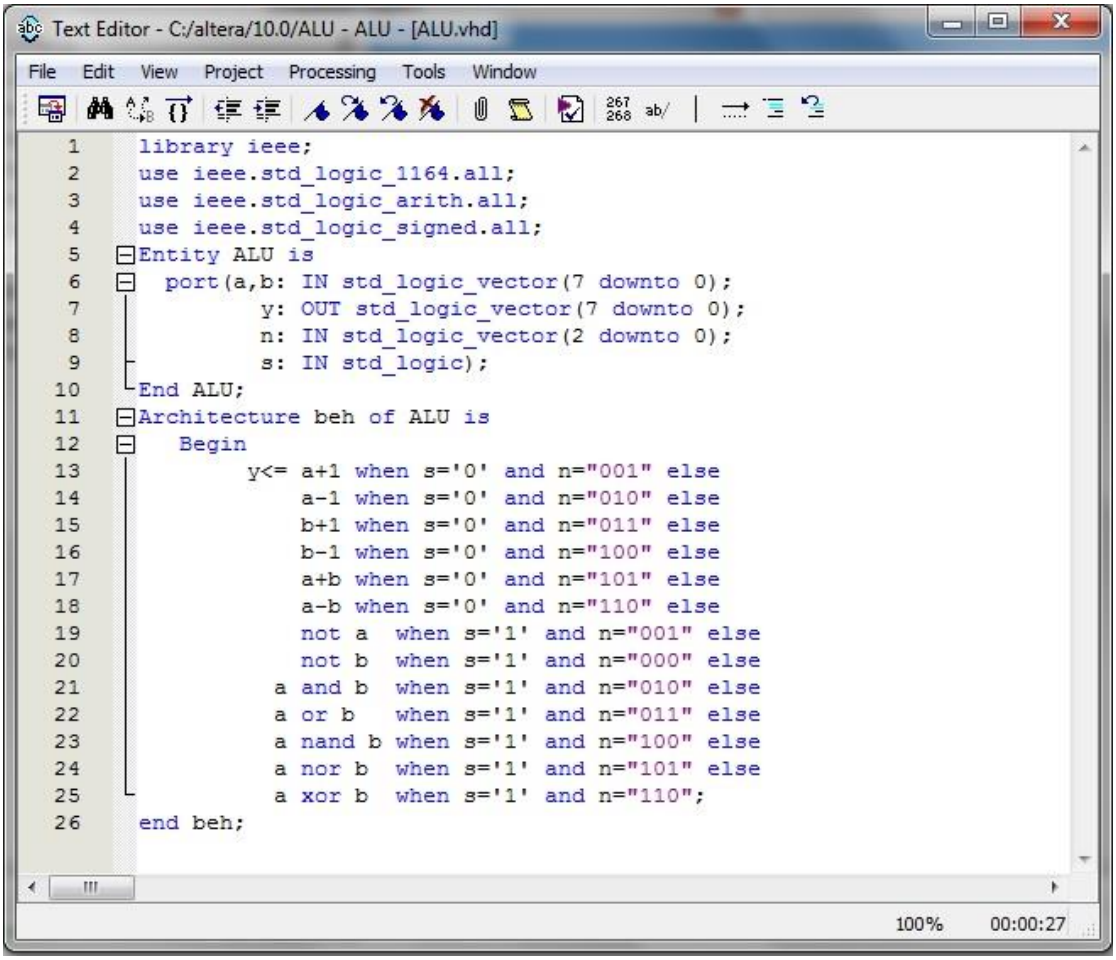



**Εικόνα 37: Οντότητα**

### 3.3.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Περνάμε τώρα στο τρίτο και τελευταίο μέρος του κώδικα της αριθμητικής λογικής μονάδας – ALU το οποίο είναι η αρχιτεκτονική και είναι το πιό ουσιστιακό κομμάτι για την επίτευξη του στόχου μας. Η αρχιτεκτονική ξεκινάει μετά το τέλος της οντότητας για την οποία έχουμε αναφερθεί προηγουμένος και ορίζεται γράφωντας την εξής εντολή: Architecture *όνομα_αρχιτεκτονικης* of *όνομα_οντότητας* is ,όπου το *όνομα_αρχιτεκτονικής* θα πρέπει να είναι διαφορετικό απο το *όνομα_οντότητας* διότι σε διαφορετική περίπτωση θα υπάρχει συνακτικό λάθος. Στο *όνομα_οντότητας* που βρίσκεται στην αρχιτεκτονική βάζουμε το όνομα της οντότητας που έχουμε χρησιμοποιήσει πριν στην οντότητα μας, εδώ αξίζει να τονίσουμε ότι υπάρχει διάκριση μεταξύ πεζών και κεφαλαίων γραμματων γι'αυτό χρειάζεται πολύ προσοχή. Για την περίπτωση της ALU μας γράφουμε: Architecture beh of ALU is ,με *όνομα_αρχιτεκτονικής* = beh και *όνομα_οντότητας* = ALU , όπως παρατηρούμε και στην γραμμή 11 του κώδικα της παρακάτω εικόνας χ5 και είμαστε έτοιμοι να ξεκινήσουμε την σύνθεση της αρχιτκτονικής. Πριν αρχίσουμε θα πρέπει να ανοίξουμε ένα μπλόκ (Block) ,μέσα στο οποίο θα τοποθετηθεί ο κώδικας της αρχιτεκτονικής, και ξεκινάει με την λέξη Begin (γραμμή 12) και κλείνει με την την λέξη end *όνομα_αρχιτεκτονικής;* ,όπου και με αυτήν την (τελευταία) εντολή (end *όνομα_αρχιτεκτονικής;*) κλείνει - τερματίζεται και το τρίτο μέρος της αρχιτεκτονικής. Στην περίπτωση μας γράφουμε end beh; για να κλείσουμε την δομή τηςαριθμητικής λογικής μονάδας – ALU.

Καταλεικτικά, όπως βλέπουμε και στην εικόνα χ5 τοποθετούνται οι αριθμητικές και λογικές πράξεις μέσα στον κώδικα και γίνεται ανάθεση της κάθε πράξης ξεχωριστά στην έξοδο y. Για τις πράξεις που θα πραγματοποιηθούν και για το πως θα πραγματοποιηθούν έχουμε αναφερθεί εκτενέστερα στην παράγραφο 3.2 . Στις γραμμές 13 έως 25 συντάσονται οι πράξεις της αριθμητικής λογικής μονάδας και εκχωρούνται στην έξοδο y, με το σύμβολο εκχώρησης <= το οποίο είναι το αντίστοιχο σύμβολο εκχώρησης στην γλώσσα VHDL. Επίσης, χρησιμοποιούνται και τρείς βοηθητικές εντολές η when η and και η else σε όλες τις εντολές εκτός απο την τελευταία εντολή (γραμμή 25) διότι δεν χρειάζεται καθώς δεν ακολουθεί άλλη εναλλακτική εντολή στην συνέχεια. Συμπερασματικά, απαιτήθηκαν 26 γραμμές κώδικα της γλώσσας VHDL για να υλοποιηθεί η αριθμητική λογική μονάδα – ALU όπως βλέπουμε και στην παρακάτω εικόνα χ5.




**Εικόνα 38: Αρχιτεκτονική**

# ΚΕΦΑΛΑΙΟ 4Ο

## 4.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Αυτό το κεφάλαιο, αναφέρεται στα συμπεράσματα που βγήκαν μετά την ολοκλήρωση της παρούσας εργασίας. Σκοπός της παρούσας εργασίας ήταν η εις βάθος κατανόηση της γλώσσας περιγραφής υλικού VHDL και της σχεδίασης μίας μηχανής πεπερασμένων καταστάσεων ( FSM ) και μιας αριθμητικής λογικής μονάδας ( ALU ) με χρήση της VHDL και την βοήθεια του προγράμματος QUARTUS II. Αρχικά, έγιναν τα ψηφιακά σχέδια του κάθε κυκλώματος ξεχωριστά, τα οποία βρίσκονται στο δεύτερο και στο τρίτο κεφάλαιο της εργασίας. Στην συνέχεια έγινε ο κώδικας για κάθε περίπτωση αντίστοιχα. Μετά την καταγραφή κάθε μέρους του

κώδικα, για να βεβαιωθούμε ότι δουλεύει σωστά , γινόταν η προσομοίωσή του. Οι προσομοιώσεις περιέχονται και αυτές στο δεύτερο και τρίτο κεφάαιλο αντίστοιχα.

Ένα απο τα συμπεράσματα που βγήκαν απο απο την εργασία αυτή, είναι ότι με την χρήση της γλώσσας VHDL μπορούμε ναα σχεδιάσουμε πολύ εύκολα και γρήγορα ένα ψηφιακό κύκλωμα, γράφοντας μερικές γραμμές κώδικα. Εκεί που με την χρήση ψηφιακού σχεδίου θα θέλαμε μέρες ολόκληρες για μια πολύπλοκη σχεδίαση, με την χρήση της γλώσσας αυτής, μέσα σε μερικές ώρες έχουμε πολύ εύκολα το σχέδιο έτοιμο. Επίσης, μπορούμε να προσομοιώσουμε την λειτουργία του κυκλώματος που περιγράψαμε, για να δώσουμε στις εισόδους του κυκλώματος μερικές τιμές, έτσι ώστε να δούμε τις τιμές που δίνουν οι έξοδοί του και να επιβεβαιωθεί η σωστή λειτουργία του.

Ένα δεύτερο συμπέρασμα που βγήκε απο την εκπόνηση της παρούσας εργασίας, είναι ότι με όσο περισσότερη λεπτομέρεια σχεδιάζει κανείς ένα ψηφιακό κύκλωμα, τόσο περισσότερο κατανοεί την λειτουργία του. Επίσης, όσο περισσότερο χρόνο καταναλώνει κανείς για μια ψηφιακή σχεδίαση, τόσες περισσότερες λεπτομέρειες κατανοεί πάνω στην σχεδίαση αυτή. Για παράδειγμα, όταν ασχολείται κάποιος με την υλοποίηση μιάς αριθμητικής λογικής μονάδας, αρχικά σκέφτεται ότι πρέπει να φτιάξει έναν καταχωρητή για την αποθήκευση της κάθε πράξης και την εμφάνιση του αποτελέσματός της στην έξοδο. Στην συνέχεια, πρέπει να αποφασίσει τι πράξεις θα εκτελεί αυτή η μονάδα. Οπότε, όταν αποφασίζει ότι θα εκτελεί εκτός των άλλων τις πράξεις της πρόσθεσης και της αφαίρεσης, σκέφτεται ότι πρέπει να υλοποιηθεί και ένας αθροιστής – αφαιρετής. Για να γίνει αυτό, πρέπει πρώτα να υλοποιηθεί ένας πλήρης αθροιστής. Έτσι κατανοείται σε όλο και μεγαλύτερο βάθος το κύκλωμα της αριθμητικής λογικής μονάδας και η λειτουργία του.

Ένα ακόμα συμπέρασμα που βγήκε απο την παρούσα εργασία είναι ότι κατά την σχεδίαση μιάς αριθμητικής λογικής ομάδας μιάς μηχανής πεπερασμένων καταστάσεων ή γενικότερα οποιουδήποτε κυκλώματος είναι πολύ χρήσιμο πάνω στα σχέδια να αναγράφονται ονομασίες των διάφορων εισόδων, εξόδων και γενικότερα των καλωδίων που συνδέουν τα διάφορα σημεία του εκάστοτε κυκλώματος. Έτσι, όταν υπάρχουν οι ονομασίες τους πάνω στα διάφορα σχέδια των κυκλωμάτων, μπορούν να αποφευχθούν τα λάθη αυτά, ή τουλάχιστον θα είναι πολύ πιο εύκολο να διορθωθούν.

# ΒΙΒΛΙΟΓΡΑΦΙΑ

ΙΣΤΟΤΟΠΟΙ :

- https://el.wikipedia.org/wiki/VHDL
- http://vivliothmmy.ee.auth.gr/1291/1/anafora.pdf
- https://www.ceid.upatras.gr/webpages/faculty/papaioan/dchmnt/2014-2015/cc/lectures/fa-from-chapter11-fromrosen-gr.pdf
- http://elnsite.teilam.gr/ebooks/digital_design/YLIKO_VHDL/THEORY/lecture05.pdf
- http://nefeli.lib.teicrete.gr/browse2/stef/hlk/2012/KaparosEmmanouil/attached-document-1328692817-146343-7986/Kaparos2012.pdf
- https://www.altera.com/
- http://teachers.teicm.gr/kalomiros/Mtptx/e-books/Essential_VHDL_Kalomiros.pdf

ΒΙΒΛΙΑ:

<<ΣΧΕΔΙΑΣΜΟΣ ΚΥΚΛΩΜΑΤΩΝ ΜΕ ΤΗΝ VHDL>>,ΤΟΜΟΣ Α, πρώτη έκδοση, VOLNEI A. PEDRONI