Τ.Ε.Ι. ΗΠΕΙΡΟΥ

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΗΠΕΙΡΟΥ

Τμήμα Μηχανικών Πληροφορικής Τ.Ε

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση ενός μικροεπεξεργαστή με VHDL κώδικα

Ροκαδάκη Φρειδερίκη

ΑΜ: 12054

Επιβλέπων καθηγητής: Βαρτζιώτης Φώτιος

Άρτα 2017

Περίληψη

Η VHDL είναι μια γλώσσα περιγραφής υλικού για την ανάπτυξη ολοκληρωμένων κυκλωμάτων και συστημάτων. Στην παρούσα πτυχιακή εργασία σχεδιάστηκε και υλοποιήθηκε ένας μικροεπεξεργαστής με τη βοήθεια της γλώσσας VHDL.

Η εργασία χωρίζεται σε δύο μέρη στο θεωρητικό και στο πρακτικό. Στο πρώτο μέρος γίνονται αναφορές στη γλώσσα VHDL, στις εντολές της καθώς και στους τρόπους με τους οποίους δομείται και συντάσσεται. Ακόμη, γίνονται αναφορές στο πρόγραμμα Quartus II και παρουσιάζονται βήμα-βήμα οι λειτουργίες του προγράμματος για την εκτέλεση ψηφιακών κυκλωμάτων. Τέλος, εξετάζονται ψηφιακά ολοκληρωμένα κυκλώματα όπως είναι τα PLD, CPLD και FPGA.

Στο πρακτικό μέρος, παρουσιάζονται εργαστηριακές ασκήσεις που υλοποιούνται με τη βοήθεια της αναπτυξιακής πλακέτας DE2 της εταιρείας Altera, η οποία βασίζεται στη διάταξη Cyclone II EP2C35F672C6. Τα ψηφιακά κυκλώματα που εκτελούνται είναι πολυπλέκτες, αποκωδικοποιητές, απαριθμητές, αθροιστές, κωδικοποιητές καθώς και καταχωρητές. Τα συγκεκριμένα κυκλώματα χρησιμοποιούνται για να συνθέσουν τον τελικό μικροεπεξεργαστή. Ο επεξεργαστής που υλοποιείται, αποτελείται από ένα κύκλωμα ελέγχου, μια αριθμητική και λογική μονάδα για την πρόσθεση και την αφαίρεση, έναν διάδρομο δεδομένων 16 bits και οκτώ καταχωρητές.

Abstract

VHDL is a hardware description language for the development of integrated circuits and systems. In this thesis a microprocessor was designed and implemented with the help of the VHDL language.

The thesis is divided into two parts in theoretical and practical. In the first part there are references to the VHDL language, in its instructions as well as the ways in which it is constructed and compiled. Still, there are references to the Quartus II project and presented step by step the program functions to perform digital circuits. Finally, digital integrated circuits such as PLD, CPLD and FPGA are examined.

In the practical part, laboratory exercises are carried out with the help of the Altera development board DE2 based on Cyclone II EP2C35F672C6. The digital circuits being executed are multiplexers, decoders, counters, adder, encoders and registers. These circuits are used to synthesize the final microprocessor. The processor being implemented consists of a control circuit, a numeric and logical unit for addition and subtraction, a 16-bit bus and eight registers.

«Δηλώνω υπεύθυνα ότι το παρόν κείμενο αποτελεί προϊόν προσωπικής μελέτης και εργασίας και πως όλες οι πηγές που χρησιμοποιήθηκαν για τη συγγραφή της δηλώνονται σαφώς είτε στις παραπομπές είτε στη βιβλιογραφία.
Γνωρίζω πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαι ενήμερη για την επέλευση των νόμιμων συνεπειών.»

Περιεχόμενα

Λίστα Πινάκων

Λίστα Προγραμμάτων

Λίστα Σχημάτων

Λίστα Εικόνων

ΚΕΦΑΛΑΙΟ 1

Η γλώσσα VHDL

1.1 Γλώσσες περιγραφής Hardware (HDL)

Μια γλώσσα HDL ή αλλιώς γλώσσα περιγραφής υλικού (hardware description language) είναι μια γλώσσα που ανήκει σε μια κλάση γλωσσών προγραμματισμού, γλωσσών προδιαγραφών ή γλωσσών μοντελοποίησης για την τυπική περιγραφή και σχεδίαση ηλεκτρονικών κυκλωμάτων, και συγκεκριμένα, κυκλώματα ψηφιακής λογικής. Αρχικά, οι γλώσσες περιγραφής υλικού (HDL) σχεδιάστηκαν για τη μοντελοποίηση και τη προσομοίωση των συστημάτων. Η ιδέα ήταν να εισάγουν δομές στην γλώσσα που να επιτρέπουν τη μοντελοποίηση και τη προσομοίωση του υλικού στα υψηλότερα επίπεδα αφαίρεσης. Μερικές από τις πιο γνωστές HDL γλώσσες είναι οι Abel, Cupl, Palasm, Verilog και η VHDL που θα χρησιμοποιηθεί παρακάτω για την διεκπεραίωση της πτυχιακής. Η PALASM σχεδιάστηκε στις αρχές της δεκαετίας του 1980 από τον John Birkner. Χρησιμοποιούνταν για να εκφράσει λογικές εξισώσεις για τις εξόδους σε ένα αρχείο κειμένου, το οποίο στην συνέχεια μετασχηματιζόταν σε ένα αρχείο που περιλάμβανε τον χάρτη των θρυαλλίδων. Αργότερα, η επιλογή της μετάφρασης από το schematic έγινε κοινή και αργότερα αυτή γινόταν από γλώσσες περιγραφής υλικού (Hardware description languages - HDL) όπως η Verilog. Το Σεπτέμβριο του 1983 η Assisted Technology κυκλοφόρησε την CUPL (Compiler for Universal Programmable Logic). Ήταν το πρώτο εμπορικό εργαλείο σχεδιασμού που υποστήριζε πολλαπλές οικογένειες PLD. Η Data-I/O κυκλοφόρησε τη γλώσσα περιγραφής υλικού ABEL τον Απρίλιο του 1984. Η ABEL είχε σαν σκοπό την περιγραφή προγραμματιζόμενων λογικών συσκευών και χρησιμοποιούνταν κυρίως για τη σχεδίαση μηχανών πεπερασμένων καταστάσεων. Το 1985 κυκλοφόρησε η πρώτη σύγχρονη γλώσσα περιγραφής υλικού, από την Gateway Design Automation, η Verilog. Η Verilog αποτελεί μια υψηλού επιπέδου γλώσσα που μπορεί να αναπαραστεί και να προσομοιώνει ψηφιακά κυκλώματα. Επιτρέπει στους σχεδιαστές να σχεδιάσουν σε διάφορα επίπεδα αφαίρεσης και είναι η πιο ευρέως χρησιμοποιούμενη HDL με μια κοινότητα χρηστών που ξεπερνά τους 50.000 ενεργά σχεδιαστές. Τέλος, το 1987 οι απαιτήσεις από το Υπουργείο Άμυνας των Ηνωμένων Πολιτειών οδήγησαν στην ανάπτυξη της VHDL.

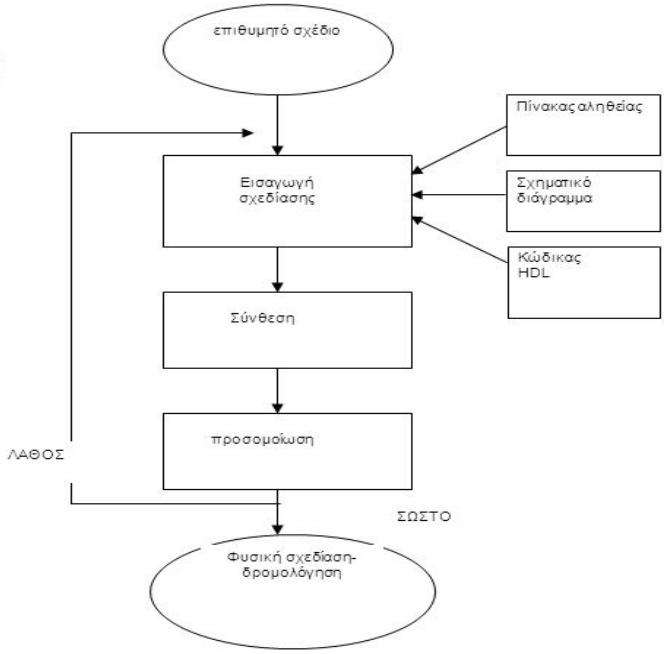
1.2 Σχετικά με τη VHDL

Η γλώσσα VHDL είναι η πρότυπη βιομηχανική γλώσσα περιγραφής ψηφιακών κυκλωμάτων. Το αρχικό πρότυπο της γλώσσας VHDL υιοθετήθηκε το 1987 και ονομάστηκε IEEE1076. Μία αναθεωρημένη εκδοχή υιοθετήθηκε το 1993 και ονομάστηκε IEEE1164. Αρχικά, είχε δύο σκοπούς: καταρχήν να αποτελεί μία γλώσσα κειμένου, η οποία θα περιγράφει τη δομή περίπλοκων ψηφιακών κυκλωμάτων και κατά δεύτερον, παρείχε δυνατότητες μοντελοποίησης της συμπεριφοράς των ψηφιακών κυκλωμάτων, και έτσι μπορούσε να χρησιμοποιηθεί ως είσοδος σε προγράμματα λογισμικού που προσομοίωναν τη συμπεριφορά των ψηφιακών κυκλωμάτων. Η VHDL είναι μια γλώσσα περιγραφής υλικού για την ανάπτυξη ολοκληρωμένων κυκλωμάτων και συστημάτων. Ως λέξη αποτελεί συντόμευση των λέξεων VHSIC Hardware Description Language όπου τα αρχικά VHSIC είναι για την συντόμευση Very High-Speed Integrated Circuit (Ολοκληρωμένα Κυκλώματα Υψηλής Ταχύτητας). Χρησιμοποιείται για καταγραφή, προσομοίωση αλλά και για την εισαγωγή σχεδίων σε συστήματα σχεδίασης CAD. Τα εργαλεία των προγραμμάτων CAD χρησιμοποιούνται για να συνθέσουν με τη βοήθεια της γλώσσας VHDL την υλοποίηση των κυκλωμάτων που περιγράφονται από τη γλώσσα. Ακόμη, να τονιστεί ότι η VHDL σε αντίθεση με τις συνηθισμένες γλώσσες προγραμματισμού οι οποίες είναι ακολουθιακές, οι εντολές της είναι από τη φύση τους συντρέχουσες, δηλαδή εκτελούνται παράλληλα. Για το λόγο αυτόν η περιγραφή της γλώσσας VHDL αναφέρεται συνήθως ως κώδικας, και όχι ως πρόγραμμα. Στη VHDL μόνο οι εντολές που τοποθετούνται μέσα σε μια διεργασία (PROCESS), συνάρτηση (FUNCTION), ή διαδικασία (PROCEDURE) εκτελούνται ακολουθιακά.

1.3 Ροή Σχεδίασης

Όπως αναφέρθηκε προηγουμένως, μία από τις πιο σημαντικές χρησιμότητες της VHDL είναι ότι επιτρέπει τη σύνθεση ενός κυκλώματος ή συστήματος σε ένα προγραμματιζόμενο στοιχείο (PLD ή FPGA) ή σε μια συσκευή ASIC. Τα βήματα που ακολουθούνται κατά τη διάρκεια ενός τέτοιου έργου συνοψίζονται στην παρακάτω εικόνα. Ο σχεδιασμός ξεκινάει με τη σύνταξη του κώδικα VHDL, ο οποίος αποθηκεύεται σε ένα αρχείο με προέκταση .vhd και με όνομα ίδιο με εκείνο της οντότητας (ENTITY) του κώδικα. Το πρώτο βήμα στη διαδικασία σύνθεσης είναι η μεταγλώττιση του κώδικα. Η μεταγλώττιση είναι η μετατροπή της υψηλού επιπέδου γλώσσας VHDL, η οποία περιγράφει το κύκλωμα σε επίπεδο

καταχωρητή σε ένα δικτύωμα σε επίπεδο πύλης. Το δεύτερο βήμα είναι η βελτιστοποίηση, η οποία εκτελείται στο δικτύωμα πυλών ως προς την ταχύτητα ή ως προς την επιφάνεια. Σε αυτό το στάδιο το κύκλωμα μπορεί να προσομοιωθεί. Τελειώνοντας, ένα λογισμικό τοποθέτησης-καλωδίωσης θα παράγει το φυσικό σχεδιασμό για ένα τσιπ PLD/FPGA ή θα παράγει τις μάσκες για μια συσκευή ASIC.



Εικόνα 1 Ροή σχεδίασης

1.4 Δομή του κώδικα

Η VHDL είναι μια γλώσσα που δημιουργήθηκε έχοντας αρχές του δομημένου προγραμματισμού. Η κύρια ιδέα της γλώσσας είναι να ορίζει τις εισόδους και τις εξόδους ενός κυκλώματος, κρύβοντας την εσωτερική του δομή. Στη συνέχεια περιγράφονται τα βασικά στοιχεία της γλώσσας VHDL καθώς και μια αναφορά στους βασικούς κανόνες σύνταξης και στη βασική δομή ενός προγράμματος VHDL. Ένα αυτόνομο τμήμα κώδικα αποτελείται από τρεις βασικές ενότητες οι οποίες είναι :

- Η ενότητα δήλωσης βιβλιοθήκης : όπου περιέχεται μία λίστα με όλες τις βιβλιοθήκες που θα χρησιμοποιηθούν για την υλοποίηση του κυκλώματος.
- Η ενότητα entity : η οποία καθορίζει τους ακροδέκτες εισόδου και εξόδου του κυκλώματος.
- Η ενότητα architecture : η συγκεκριμένη ενότητα περιέχει τον κατάλληλο κώδικα VHDL, ο οποίος περιγράφει με ποιον τρόπο πρέπει να συμπεριφέρεται το κύκλωμα.

1.4.1 Δηλώσεις βιβλιοθηκών

Για τη δήλωση μιας βιβλιοθήκης αυτό που έχει να κάνει κάποιος είναι να γράψει δύο γραμμές κώδικα, όπου η πρώτη θα περιέχει το όνομα της βιβλιοθήκης και η δεύτερη γραμμή την πρόταση use. Αυτό γίνεται ως εξής :

Library όνομα_βιβλιοθήκης ;

Use όνομα_βιβλιοθήκης.όνομα_πακέτου.μέρη_πακέτου ;

Ακόμη, στους σχεδιασμούς είναι απαραίτητα τουλάχιστον ένα από τα τρία πακέτα, από τρεις διαφορετικές βιβλιοθήκες :

- IEEE.std_logic_1164 (από τη βιβλιοθήκη IEEE)
- Standard (από τη βιβλιοθήκη std)
- Work (από τη βιβλιοθήκη work)

Οι δύο τελευταίες βιβλιοθήκες είναι πάντα ορατές εξ' ορισμού γι' αυτό το λόγο δεν είναι απαραίτητο να τις δηλώνουμε. Αντιθέτως, η βιβλιοθήκη ieee πρέπει οπωσδήποτε να δηλώνεται όταν χρησιμοποιείται ο τύπος δεδομένων STD_LOGIC. Η std αποτελεί μια βιβλιοθήκη πόρων για το σχεδιαστικό περιβάλλον της VHDL όπως για παράδειγμα τύποι δεδομένων, είσοδος/έξοδος κειμένου. Η βιβλιοθήκη work είναι το σημείο αποθήκευσης του σχεδιασμού. Η βιβλιοθήκη ieee περιέχει αρκετά πακέτα, μερικά από αυτά αναλύονται παρακάτω :

- Std_logic_1164 : καθορίζει τα λογικά συστήματα std_logic.
- Std_logic_arith : καθορίζει τύπους δεδομένων όπως Signed και Unsigned καθώς και τις σχετιζόμενες αριθμητικές αλλά και λογικές πράξεις.
- Std_logic_signed : ο τύπος signed χρησιμοποιείται σε προγράμματα κυκλωμάτων που χειρίζονται προσημασμένους αριθμούς.
- Std_logic_unsigned : ο τύπος unsigned χρησιμοποιείται σε προγράμματα που χειρίζονται μη-προσημασμένους αριθμούς.

1.4.2 Δήλωση οντότητας

Η ενότητα entity (οντότητα) περιέχει τα σήματα εισόδου και εξόδου. Το όνομα της οντότητας μπορεί να είναι οποιοδήποτε έγκυρο όνομα της γλώσσας VHDL. Η σύνταξη της φαίνεται παρακάτω :

Entity όνομα_οντότητας is

Port (

Όνομα_θύρας : κατάσταση_σήματος τύπος_σήματος ;

Όνομα_θύρας : κατάσταση_σήματος τύπος_σήματος ;

End όνομα_οντότητας ;

1.4.3 Δήλωση αρχιτεκτονικής

Η ενότητα architecture περιέχει την περιγραφή του τρόπου με τον οποίο πρέπει να συμπεριφέρεται το κύκλωμα. Η σύνταξη της είναι ως εξής :

Architecture όνομα_αρχιτεκτονικής of όνομα_οντότητας is

[δηλώσεις]

Begin

(κώδικας)

End όνομα_αρχιτεκτονικής ;

Η αρχιτεκτονική αποτελείται από δύο μέρη : το δηλωτικό μέρος, όπου δηλώνονται τα εσωτερικά σήματα και οι σταθερές, και το τμήμα του κώδικα. Το όνομα της architecture μπορεί να είναι οτιδήποτε ακόμα και να είναι το ίδιο με το όνομα της οντότητας.

1.4.4 Συστατικά (components)

Τα συστατικά (components) αποτελούν μια κεντρική ιδέα στην VHDL. Ένα συστατικό μπορεί να χρησιμοποιηθεί για την κατασκευή βιβλιοθηκών από συστατικά, όπως για παράδειγμα, οι μικροεπεξεργαστές, ειδικά κυκλώματα αλλά και άλλα κυκλώματα. Ο σωστός σχεδιασμός ενός συστατικού έχει ως αποτέλεσμα την αντιγραφή του όσες φορές χρειαστεί μέσα στο κύκλωμα. Αυτό σημαίνει ότι τα συστατικά είναι επαναχρησιμοποιούμενα. Παρακάτω παρουσιάζεται η σύνταξη δήλωσης ενός συστατικού.

Component όνομα_συστατικού_στοιχείου IS

Port (

Όνομα_θύρας : κατάσταση_σήματος τύπος σήματος ;

Όνομα_θύρας : κατάσταση_σήματος τύπος σήματος ;

...) ;

End_component ;

1.4.5 Πακέτα (packages)

Ένα πακέτο της γλώσσας VHDL λειτουργεί σαν αποθηκευτικός χώρος. Χρησιμοποιείται για να συγκρατεί προγράμματα της γλώσσας VHDL που είναι γενικής χρήσης, όπως τα προγράμματα που ορίζουν ένα τύπο. Ένα πακέτο μπορεί να έχει δύο μέρη : τη δήλωση πακέτου και το σώμα του πακέτου**.** Παρακάτω παρουσιάζεται η γενική μορφή δήλωσης ενός πακέτου.

PACKAGE όνομα_πακέτου IS

(δηλώσεις)

END όνομα_πακέτου ;

[PACKAGE BODY όνομα_πακέτου IS

(περιγραφές Συναρτήσεων και Διαδικασιών)

END όνομα_πακέτου ;]

1.5 Τύποι δεδομένων και αντικείμενα

Η γλώσσα VHDL παρέχει δύο αντικείμενα για να χειρίζεται μη στατικές τιμές δεδομένων. Παρέχει λοιπόν τα σήματα (signals) και τις μεταβλητές (variables). Για τον καθορισμό των στατικών τιμών χρησιμοποιεί τα αντικείμενα τύπου Constant (σταθερές) και την εντολή Generic.

Τα σήματα καθώς και οι σταθερές μπορούν να χρησιμοποιηθούν είτε σε συντρέχοντα είτε σε ακολουθιακό τύπο κώδικα. Επίσης, και τα δύο αντικείμενα μπορούν να είναι καθολικά δηλαδή να είναι ορατά σε όλο τον κώδικα. Από την άλλη, οι μεταβλητές μπορούν να χρησιμοποιηθούν μόνο μέσα σε ένα τμήμα ακολουθιακού κώδικα, δηλαδή, μέσα σε μια

συνάρτηση, διεργασία ή διαδικασία. Η τιμή μιας μεταβλητής δεν μπορεί ποτέ να μεταδοθεί έξω από το τμήμα ενός ακολουθιακού κώδικα.

1.5.1 Σήματα (Signals)

Τα σήματα χρησιμοποιούνται για τη μετάδοση τιμών εσωτερικά αλλά και εξωτερικά του κυκλώματος, καθώς και μεταξύ των εσωτερικών μονάδων του. Το αντικείμενο τύπου signal αναπαριστά τις κυκλωματικές διασυνδέσεις (καλώδια) όπως, για παράδειγμα οι θύρες μιας οντότητας είναι αντικείμενα τύπου signal. Στη συνέχεια παρουσιάζεται η δήλωση ενός αντικειμένου signal :

Signal όνομα : τύπος [εύρος] [:= αρχική_τιμή] ;

Παραδείγματα χρήσης των σημάτων :

Signal count : Integer range 0 to 100;

Signal control : Bit := '0' ;

1.5.2 Σταθερές (Constants)

Μια σταθερά χρησιμοποιείται για τον καθορισμό προεπιλεγμένων τιμών. Τα αντικείμενα τύπου Constant μπορούν να δηλωθούν σε ένα πακέτο, σε μια οντότητα ή ακόμα και σε μια αρχιτεκτονική. Η σύνταξη μιας σταθεράς είναι η ακόλουθη :

Constant όνομα : τύπος := τιμή ;

Παράδειγμα:

Constant set_bit: Bit := '1' ;

1.5.3 Μεταβλητές (Variables)

Τα αντικείμενα τύπου variable αναπαριστούν μόνο τοπικές πληροφορίες. Ένα αντικείμενο τέτοιου τύπου μπορεί να χρησιμοποιηθεί μόνο μέσα σε μια διεργασία, συνάρτηση ή διαδικασία και η τιμή του δεν μπορεί να μεταβιβαστεί άμεσα έξω από τον ακολουθιακό κώδικα. Η ενημέρωση ενός αντικειμένου τύπου variable είναι άμεση και η νέα του τιμή

μπορεί να χρησιμοποιηθεί απευθείας στην επόμενη γραμμή κώδικα. Η σύνταξη μιας μεταβλητής είναι η ακόλουθη :

VARIABLE όνομα: τύπος [εύρος] [:=αρχική τιμή];

Παραδείγματα χρήσης μιας μεταβλητής:

VARIABLE count: INTEGER RANGE 0 TO 100;

VARIABLE y: STD_LOGIC_VECTOR (7 DOWN TO 0):= "100100";

1.5.4 Τελεστές και ιδιότητες (operators and operations)

Η VHDL παρέχει διάφορους τύπους προκαθορισμένων τελεστών που είναι οι εξής:

- Τελεστές ανάθεσης τιμής
- Λογικοί τελεστές
- Αριθμητικοί τελεστές
- Τελεστές σύγκρισης
- Τελεστές ολίσθησης
- Τελεστές συνένωσης

Τελεστές ανάθεσης τιμής

Οι τελεστές αυτοί χρησιμοποιούνται για την απόδοση τιμών σε σήματα, μεταβλητές και σταθερές. Οι τελεστές ανάθεσης τιμής είναι:

<= Χρησιμοποιείται για την απόδοση τιμής σε ένα αντικείμενο τύπου signal.

:= Χρησιμοποιείται για την ανάθεση τιμής σε αντικείμενο τύπου VARIABLE, CONSTANT ή GENERIC. Επίσης, χρησιμοποιείται και για την απόδοση αρχικών τιμών.

= > Χρησιμοποιείται για την ανάθεση τιμής σε μεμονωμένα στοιχεία διανύσματος ή με την εντολή OTHERS.

Λογικοί τελεστές

Χρησιμοποιούνται για την εκτέλεση λογικών πράξεων. Τα δεδομένα πρέπει να είναι τύπου BIT, STD_LOGIC ή STD_ULOGIC. Οι λογικοί τελεστές είναι οι εξής:

- NOT
- AND
- OR
- NAND
- NOR
- XOR
- XNOR

Αριθμητικοί τελεστές

Χρησιμοποιούνται για την εκτέλεση αριθμητικών πράξεων. Τα δεδομένα μπορεί να είναι τύπου INTEGER, SIGNED ή UNSIGNED. Τέτοιοι τελεστές είναι:

+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
**	Ύψωση σε δύναμη
MOD	Ακέραιο υπόλοιπο διαίρεσης
REM	Υπόλοιπο
ABS	Απόλυτη τιμή

Πίνακας 1 Αριθμητικοί τελεστές

Τελεστές σύγκρισης

Χρησιμοποιούνται για την πραγματοποίηση συγκρίσεων. Τα δεδομένα μπορεί να ανήκουν σε οποιοδήποτε τύπο από αυτούς που αναφέρθηκαν πιο πάνω: Οι τελεστές σύγκρισης είναι οι:

=	Ίσο
/=	Όχι ίσιο
<	Μικρότερο
>	Μεγαλύτερο
<=	Μικρότερο ή ίσο
>=	Μεγαλύτερο ή ίσο

Πίνακας 2 Τελεστές σύγκρισης

Τελεστές ολίσθησης

Οι τελεστές ολίσθησης χρησιμοποιούνται για την ολίσθηση δεδομένων. Η σύνταξή τους είναι η ακόλουθη: <αριστερός τελεστής> <πράξη ολίσθησης> <δεξιός τελεστής>. Ο αριστερός τελεστής πρέπει να είναι τύπου BIT_VECTOR, ενώ ο δεξιός τελεστής πρέπει να είναι τύπου INTEGER. Οι τελεστές ολίσθησης είναι:

Sll	Λογική ολίσθηση αριστερά
Srl	Λογική ολίσθηση δεξιά
Sla	Αριθμητική ολίσθηση αριστερά
Sra	Αριθμητική ολίσθηση δεξιά
Rol	Λογική αριστερή περιστροφή
Ror	Λογική δεξιά περιστροφή

Πίνακας 3 Τελεστές ολίσθησης

Τελεστές συνένωσης

Οι τελεστές αυτοί χρησιμοποιούνται για την ομαδοποίηση τιμών. Τα δεδομένα μπορεί να είναι οποιουδήποτε τύπου από τους τελεστές που χρησιμοποιούνται για την εκτέλεση λογικών πράξεων. Τέτοιοι τελεστές είναι οι:

- &
- (,,,)

Ιδιότητες

Ο σκοπός των ιδιοτήτων στην γλώσσα VHDL έχει ως σκοπό την μεγαλύτερη ευελιξία της γλώσσας αλλά και την δημιουργία γενικών τμημάτων κώδικα. Οι συγκεκριμένες ιδιότητες χωρίζονται σε δύο κατηγορίες:

- Ιδιότητες δεδομένων: Χρησιμοποιούνται για να επιστραφεί μια τιμή σχετικά με ένα διάνυσμα.
- Ιδιότητες σημάτων: Χρησιμοποιούνται για την παρακολούθηση ενός σήματος και επιστρέφουν τιμή TRUE ή FALSE.

Ιδιότητες δεδομένων

Οι προκαθορισμένες συνθέσιμες ιδιότητες δεδομένων είναι οι ακόλουθες:

d'Low	Επιστρέφει το χαμηλότερο δείκτη του πίνακα
d'High	Επιστρέφει τον υψηλότερο δείκτη του πίνακα
d'Left	Επιστρέφει τον αριστερότερο δείκτη του πίνακα
d'Right	Επιστρέφει τον δεξιότερο δείκτη του πίνακα
d'Length	Επιστρέφει το μέγεθος ενός διανύσματος
d'Range	Επιστρέφει το εύρος του διανύσματος
d'Reverse_range	Επιστρέφει το εύρος του διανύσματος σε αντίστροφη σειρά

Πίνακας 4 Ιδιότητες δεδομένων

Η εντολή Generic

Μια εντολή generic μπορεί να χρησιμοποιηθεί στην παραμετροποίηση μιας σχεδίασης. Με τον τρόπο αυτό γίνεται εισαγωγή μιας πληροφορίας μέσα στο μοντέλο, π.χ. χρονική πληροφορία, η οποία μπορεί να αλλάζει κάθε φορά που ένα στιγμιότυπο του component που την περιέχει, χρησιμοποιείται. Αυτό δηλώνεται μέσω της εντολής generic map. Ένα generic δηλώνεται στο σώμα μιας οντότητας πριν τη δήλωση port και μεταχειρίζεται ως constant μέσα σε μία αρχιτεκτονική. Η σύνταξη της εντολής generic είναι η εξής:

GENERIC (όνομα_ παραμέτρου: τύπος_ παραμέτρου:= τιμή_ παραμέτρου);

ΚΕΦΑΛΑΙΟ 2

Εργαλεία σχεδίασης ψηφιακών κυκλωμάτων

Στους σημερινούς υπολογιστές υπάρχουν λογικά κυκλώματα που περιέχουν περίπλοκα συστήματα και είναι αδύνατο ένας χρήστης να τα σχεδιάσει με το χέρι. Για την σχεδίαση τέτοιων κυκλωμάτων υπάρχουν εταιρείες που προσφέρουν διάφορα εργαλεία σχεδίασης.

Η Mentor Graphics παρέχει το ModelSim, το οποίο είναι ένα περιβάλλον για την προσομοίωση γλωσσών προγραμματισμού όπως είναι η VHDL και η Verilog. Το ModelSim μπορεί να χρησιμοποιηθεί ανεξάρτητα ή σε συνδυασμό με το εργαλείο Quartus. Η Altera παρέχει ένα από τα πιο γνωστά εργαλεία σχεδίασης το Quartus, που αποτελεί ένα ολοκληρωμένο εργαλείο σύνθεσης, γραφικής προσομοίωσης και προγραμματισμού. Η εταιρεία Synopsys, Inc., το 1989 εισήγαγε το πρώτο εμπορικό λογισμικό για σύνθεση λογικής. Με τον τρόπο αυτό το κύκλωμα περιγράφεται σε γλώσσα περιγραφής εξοπλισμού (hardware description language – HDL) και στη συνέχεια συνθέτει αυτόματα το μοντέλο netlists. Το 1988 ιδρύθηκε στην Αμερική η εταιρεία Cadence Design Systems, Inc. Η Cadence διαθέτει αρκετά εργαλεία σχεδίασης που παρέχουν τη δυνατότητα σχεδίασης κυκλωμάτων σε όλες τις υπάρχουσες περιγραφές, και καθιστούν δυνατή την επίτευξη μιας ροής σχεδίασης πλήρως καθορισμένης από το χρήστη, που χρησιμοποιεί ιεραρχικές δομές.

2.1 Το λογισμικό Quartus II

Το εργαλείο Quartus II παρέχει ένα περιβάλλον σχεδιασμού συστημάτων ανεξάρτητο αρχιτεκτονικής, με πολλαπλές πλατφόρμες. Δίνει την δυνατότητα ολοκληρωμένου σχεδιασμού συστημάτων, γρήγορης επεξεργασίας και άμεσου προγραμματισμού των συσκευών της Altera. Καλύπτει όλο το φάσμα λογικού σχεδιασμού, με δυνατότητες δημιουργίας πολύπλοκων και ιεραρχικών σχεδιασμών, δυναμική σύνθεση, διαμέλιση, λειτουργική και χρονική εξομοίωση, χρονική ανάλυση, αυτόματο εντοπισμό λαθών, προγραμματισμό συσκευών και επιβεβαίωση της λειτουργίας τους. Γίνονται αποδεκτοί σχεδιασμοί σε VHDL, Verilog, AHDL (Altera HDL) καθώς και σχηματικά διαγράμματα που δημιουργούνται από τον ειδικό γραφικό Editor του εργαλείου.

Ο compiler αποτελεί μία από τις ισχυρές δυνατότητες του Quartus και δίνει την καλύτερη δυνατή υλοποίηση του συστήματος. Με δυνατότητες αυτόματου εντοπισμού των λαθών

στον αρχικό σχεδιασμό ή στην υλοποιημένη μορφή του στο FPGA καθώς και με την εκτεταμένη τεκμηρίωση λαθών διευκολύνει κατά πολύ την διαδικασία σχεδιασμού.

Το σχεδιαστικό λογισμικό της Altera Quartus II επιτρέπει τον εύκολο σχεδιασμό, προσομοίωση και υλοποίηση σχεδιασμών λογικής με διαφορετικό βαθμό πολυπλοκότητας. Αποτελείται από διάφορες εφαρμογές, στις οποίες παρέχεται άμεση πρόσβαση από τον βασικό διαχειριστή του QUARTUS II.

2.2 Εισαγωγή σχεδίασης

Για την σχεδίαση ενός κυκλώματος με το Quartus υπάρχουν δυο μέθοδοι εισαγωγής σχεδίασης: *η μέθοδος σχηματικών διαγραμμάτων* και *η μέθοδος γραφής πηγαίου κώδικα* σε μία κατάλληλη γλώσσα.

Σχηματικό διάγραμμα

Ένα λογικό κύκλωμα μπορεί να σχεδιαστεί με λογικές πύλες που συνδέονται με καλώδια. Σε ένα εργαλείο σχηματικού διαγράμματος τα στοιχεία του κυκλώματος όπως οι λογικές πύλες απεικονίζονται σαν γραφικά σύμβολα και οι συνδέσεις μεταξύ των κυκλωματικών στοιχείων σχεδιάζονται σαν γραμμές. Το Quartus παρέχει βιβλιοθήκες μέσα στις οποίες υπάρχουν διάφορα σύμβολα που αναπαριστούν λογικές πύλες διαφόρων τύπων με διάφορους αριθμούς πυλών. Επίσης, υπάρχει η δυνατότητα χρησιμοποίησης των προγραμμάτων που έχουν ήδη δημιουργηθεί από το χρήστη καθώς όλα τα προγράμματα αποθηκεύονται στη μνήμη του προγράμματος.

Πηγαίος κώδικας

Η εισαγωγή σχεδίασης ενός λογικού κυκλώματος μπορεί να πραγματοποιηθεί με τη χρήση της γλώσσας VHDL και για την είσοδο στο Quartus χρησιμοποιείται ο Text Editor. Ο πηγαίος κώδικας της γλώσσας VHDL έχει τη μορφή απλού κειμένου και με αυτό τον τρόπο είναι πιο εύκολη η χρήση της από τον σχεδιαστή να την συμπεριλάβει στα συνοδευτικά κείμενα του κυκλώματος. Η VHDL έχει χρησιμοποιηθεί για την υλοποίηση μεγάλων κυκλωμάτων όπως οι μικροεπεξεργαστές που περιέχουν εκατομμύρια τρανζίστορ.

2.3 Λογική σύνθεση και βελτιστοποίηση

Το επόμενο βήμα μετά την ολοκλήρωση της περιγραφής της ψηφιακής λογικής είναι η διαδικασία της σύνθεσης. Στο QUARTUS II η διαδικασία της σύνθεσης αναφέρεται ως «Analysis & Synthesis». Κατά την διάρκεια αυτής της διαδικασίας η είσοδος μετατρέπεται στις κατάλληλες λογικές συναρτήσεις, με τρόπο που να ταιριάζει στην τεχνολογία της συγκεκριμένης διάταξης που τελικά θα διαμορφωθεί. Για παράδειγμα, έστω ότι σαν είσοδο είναι κάποιος πίνακας αληθείας, και στόχος μια διάταξη CPLD. Κατά την διάρκεια της σύνθεσης δημιουργούνται λογικές συναρτήσεις που εκφράζουν τους πίνακες αληθείας, με τρόπο κατάλληλο ώστε τελικά να απεικονιστούν σε μια διάταξη λογικών πινάκων (Logic Arrays). Στους λογικούς πίνακες AND-OR στηρίζονται οι γεννήτριες συναρτήσεων των CPLDs. Αν έχουμε σαν είσοδο σχηματικό διάγραμμα και στόχο ένα FPGA, κατά την σύνθεση παίρνουμε λογικές εξισώσεις ισοδύναμες με το κύκλωμα του σχηματικού διαγράμματος, οι οποίες μπορούν να μεταφερθούν σε πίνακες αναφοράς (look-up tables). Τέλος, όταν η εισαγωγή γίνεται μέσω μιας γλώσσας περιγραφής υλικού, κατά την διαδικασία της σύνθεσης τίθεται σε λειτουργία ένα άλλο τμήμα της σύνθεσης, ο μεταφραστής, και σαν έξοδο έχουμε την περιγραφή του κυκλώματος σε χαμηλό επίπεδο, κατάλληλο για την τεχνολογία της διάταξης-στόχου.

Με άλλα λόγια δηλαδή, η σύνθεση χρησιμοποιείται ως ένα αυτόματο εργαλείο υλοποίησης του αρχικού κυκλώματος που δίνει ο σχεδιαστής, με τρόπο κατάλληλο για την τεχνολογία του στόχου. Ταυτόχρονα, το κύκλωμα που παίρνουμε στην έξοδο της σύνθεσης είναι καλύτερο από το αρχικό. Δηλαδή, κατά την σύνθεση έχουμε και βελτιστοποίηση του κυκλώματος.

2.4 Προσαρμογή

Η διαδικασία της προσαρμογής (fitting) λέγεται αλλιώς και δρομολόγηση (place and route). Στη φάση αυτή χρησιμοποιείται η βάση δεδομένων που έχει δημιουργηθεί κατά την ανάλυση και σύνθεση, και αντιστοιχίζεται η ψηφιακή λογική στους ελεύθερους πόρους της διάταξης-στόχου. Μάλιστα, λαμβάνονται αυστηρά υπόψη και οι χρονικές απαιτήσεις που τέθηκαν από τον σχεδιαστή. Με άλλα λόγια, στη φάση αυτή καθορίζεται το πόσα και ποια συγκεκριμένα λογικά στοιχεία (logic elements-LE’s) του ολοκληρωμένου κυκλώματος (chip) θα χρησιμοποιηθούν. Επίσης επιλέγονται συνδέσεις από τον προγραμματιζόμενο πίνακα διασυνδέσεων (programmable interconnect), ώστε να διασυνδεθούν τα απαραίτητα LE’s μεταξύ τους.

2.5 Χρονική ανάλυση

Η χρονική ανάλυση (timing analysis), παράγεται ως αποτέλεσμα της προσαρμογής. Στην χρονική ανάλυση παράγονται οι καλύτεροι και οι χειρότεροι χρόνοι του κυκλώματος, με βάση τις προβλεπόμενες καθυστερήσεις κατά μήκος των διαδρομών. Δηλαδή, υπολογίζονται οι καθυστερήσεις που υφίστανται τα σήματα από την είσοδο τους στο κύκλωμα ως την έξοδο τους. Οι καθυστερήσεις που προκύπτουν είναι αποτέλεσμα του μήκος των καλωδίων και του αριθμό των ενδιάμεσων βαθμίδων. Η χρονική ανάλυση είναι ένα σημαντικό σημείο κατά τη σχεδίαση ενός κυκλώματος, καθώς θα πρέπει οι χρόνοι που παίρνονται από την εκτέλεση της να είναι συμβατοί με αυτούς που καθορίζονται από το ρολόι του συστήματος. Σε περίπτωση μη συμβατότητας των χρόνων θα πρέπει να διορθώνονται οι αποκρίσεις αυτές, κάνοντας κατάλληλες χρονικές εκχωρήσεις (timing assignments) στο λογισμικό και επαναλαμβάνοντας την δρομολόγηση.

Κατά την τελευταία φάση (assembling) γίνεται η παραγωγή συμβολικού (.hex) και δυαδικού (.sof) κώδικα, καθώς επίσης και η δημιουργία των προγραμματιστικών αρχείων, που θα χρησιμοποιηθούν για την τελική διαμόρφωση. Ένα αρχείο (.sof) εισάγεται στην αναπτυξιακή πλακέτα με σκοπό να προγραμματιστεί το FPGA ώστε να είναι το σύστημα που είχε δημιουργηθεί.

Οι παραπάνω διαδικασίες, δηλαδή η σύνθεση, η προσαρμογή, η χρονική ανάλυση καθώς και η διαδικασία του «Assembler», συνθέτουν την διαδικασία της μετάφρασης (Compilation). Δηλαδή απεικονίζουν το επιθυμητό κύκλωμα σε μία προγραμματιζόμενη διάταξη (FPGA ή CPLD) και δημιουργούν κώδικα για την προσομοίωση λειτουργίας (Simulation), και τον προγραμματισμό των διατάξεων (device programming).

2.6 Προσομοίωση

Όταν πραγματοποιείται η λειτουργία της προσομοίωσης (Simulation) γίνεται ένας έλεγχος κατά πόσο το κύκλωμα λειτουργεί σωστά, οπότε, δεν αρκεί μόνο η λειτουργία της σύνθεσης κατά την οποία γίνεται βελτιστοποίηση του κυκλώματος για να συμπεράνουμε ότι το κύκλωμα που σχεδιάσαμε είναι σωστό.

Ο έλεγχος της σωστής λειτουργίας γίνεται στο τμήμα των διεργασιών που ονομάζεται προσομοίωση και πραγματοποιείται με το συνδυασμό δύο παραμέτρων. Ο ένας είναι το αρχικό σχέδιο και ο άλλος είναι κάποιες τιμές που δίνει ο χρήστης στις εισόδους του κυκλώματος ώστε να ελέγξει αν οι τιμές που θα πάρει στην έξοδο του ανταποκρίνονται στις

αρχικές προδιαγραφές που είχαν τεθεί για το κύκλωμα. Ο προσομοιωτής εξάγει τις τιμές της εξόδου του κυκλώματος μέσω ενός πίνακα αληθείας ή μέσω ενός διαγράμματος χρονισμού. Στην δεύτερη περίπτωση πρέπει να ληφθεί υπόψη ότι ο προσομοιωτής θεωρεί ότι ο χρόνος που χρειάζεται να μεταβούν τα σήματα των εισόδων στις πύλες είναι μηδενικός, κάτι που στην πραγματικότητα δεν ισχύει. Αυτό έχει σαν αποτέλεσμα να υπάρχει ένα ποσοστό αμφιβολίας ακόμα και μετά την προσομοίωση για το αν το κύκλωμα που σχεδιάστηκε μπορεί να υλοποιηθεί.

2.7 Προγραμματισμός και διαμόρφωση της συσκευής

Η διαδικασία του προγραμματισμού είναι η τελευταία ενέργεια που γίνεται για την ολοκλήρωση της δημιουργίας του κυκλώματος. Ως γνωστό τα CPLDs ή τα FPGAs πρέπει να προγραμματιστούν για να υλοποιήσουν το κύκλωμα που σχεδιάστηκε. Το αρχείο που χρησιμοποιείται είναι το .sof αρχείο το οποίο δημιουργήθηκε στην φάση της χρονικής ανάλυσης.

Η εταιρεία Altera επιτρέπει τον προγραμματισμό των συσκευών της με δύο τρόπους. Ο ένας είναι μέσω του κυκλώματος διεπαφής JTAG και ο άλλος ο Active Serial (AS) mode. Η μεταφορά των διαμορφωμένων αρχείων γίνεται από τον υπολογιστή του χρήστη στο board με τη βοήθεια ενός καλωδίου το οποίο συνδέεται σε θύρα του υπολογιστή (παράλληλη ή USB) και στο board όπου βρίσκεται το CPLD ή το FPGA. Η σύνδεση αυτή γίνεται μέσω του κατάλληλου οδηγού (driver) USB-Blaster ή BYTE-BLASTER.

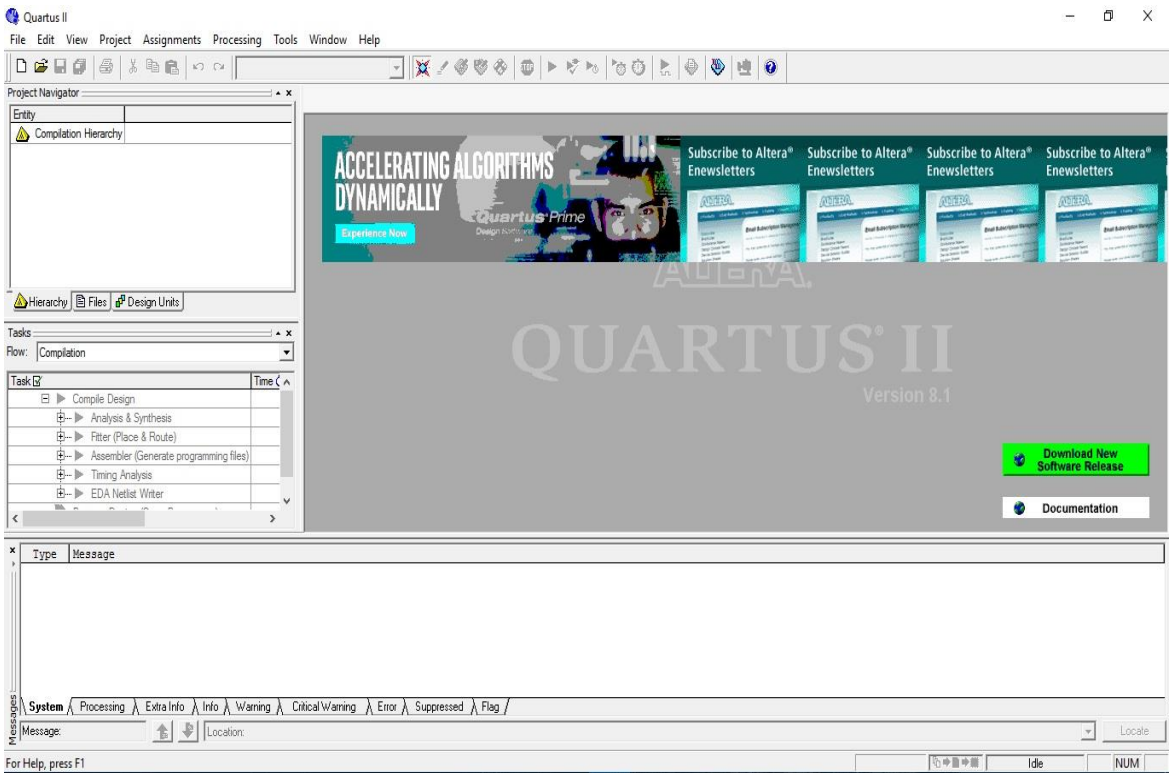
Αν γίνει χρήση της JTAG διεπαφής, τα δεδομένα οδηγούνται κατευθείαν στο ολοκληρωμένο κύκλωμα. Με αυτόν τον τρόπο το ολοκληρωμένο κύκλωμα (αν είναι FPGA) διατηρεί την διαμόρφωση που του έχει δοθεί για όσο διαρκεί η τροφοδοσία του. Αν σταματήσει να τροφοδοτείται χάνεται και η διαμόρφωση του. Αυτό δεν ισχύει για τα κυκλώματα CPLDs, τα οποία διαμορφώνονται μόνιμα, με δυνατότητα επανεγγραφής, όπως οι μνήμες flash EEPROM.

Στην άλλη περίπτωση (AS) μια διάταξη με μνήμες flash, που βρίσκεται πάνω στην ίδια πλακέτα με το FPGA, χρησιμοποιείται για να αποθηκεύει τα αρχεία διαμόρφωσης. Σε αυτήν την περίπτωση το Quartus II στέλνει τα δεδομένα μας στη μνήμη Flash και αυτή με τη σειρά της στο chip FPGA. Στην περίπτωση αυτή η διαμόρφωση συνεχίζεται και στην περίπτωση μη τροφοδοσίας. Για την επιλογή ποιόν από τους δύο τρόπους θα χρησιμοποιηθεί γίνεται

συνήθως μέσω ενός διακόπτη RUN/PROG που βρίσκεται πάνω στο board. Η αυτόματη (default) επιλογή είναι για διαμόρφωση με JTAG, ενώ η δεύτερη για Active Serial (AS).

2.8 Το περιβάλλον του Quartus

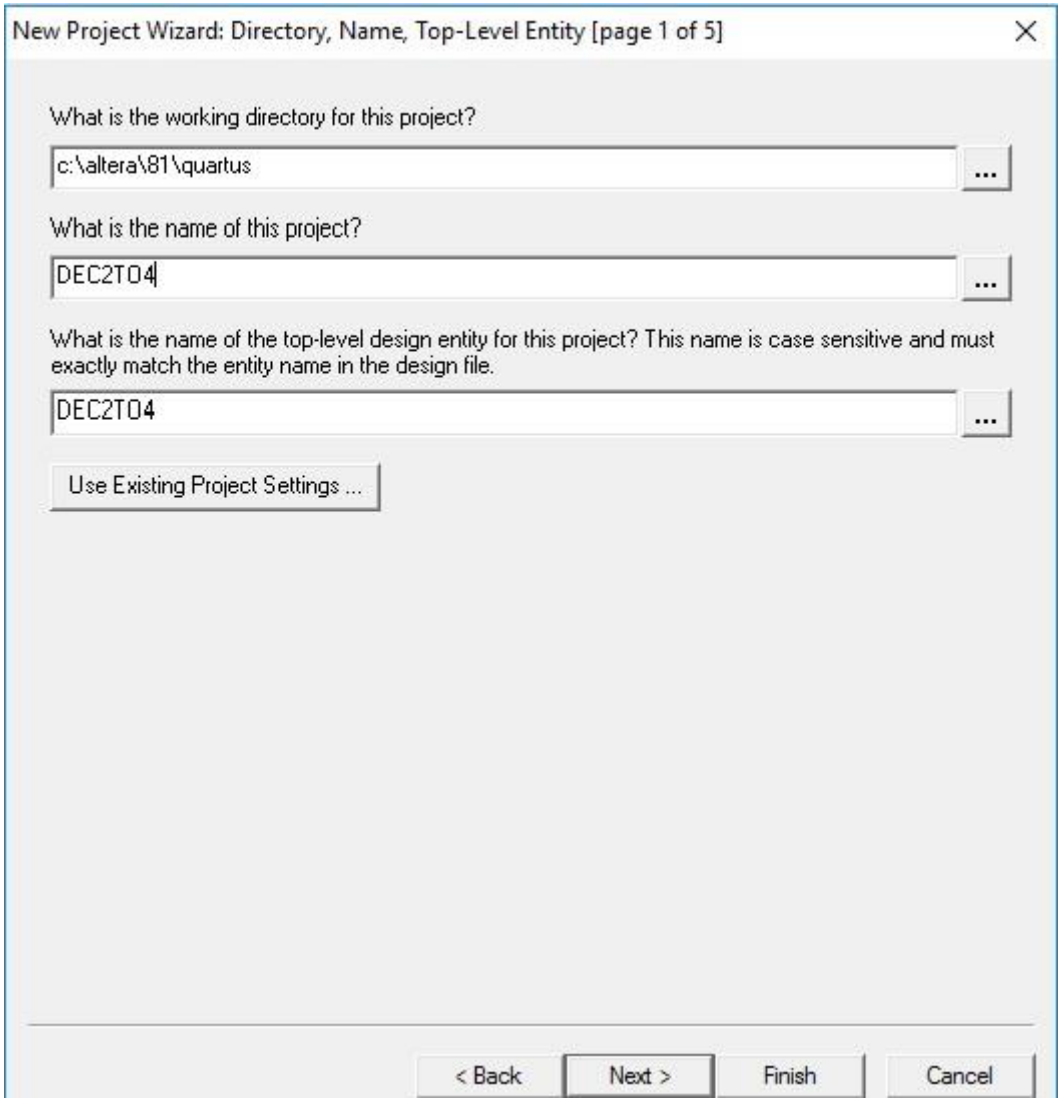
Το βασικό περιβάλλον εργασίας του εργαλείου Quartus φαίνεται στην *Εικόνα 2*. Διακρίνονται ο τίτλος, το βασικό μενού εντολών και τα κουμπιά συντόμευσης.



Εικόνα 2 Το περιβάλλον εργασίας του Quartus II

2.8.1 Ορισμός του σχεδίου (Project)

Κατά την εκκίνηση του προγράμματος Quartus, πρέπει να δοθεί η ονομασία του project. Με τον όρο project εννοείται το σύνολο των αρχείων που δημιουργούνται από το χρήστη (δηλαδή το αρχικό σχέδιο και τις κυματομορφές εισόδου που θα χρησιμοποιηθούν στην προσομοίωση), καθώς και τα αρχεία που δημιουργούνται από το πρόγραμμα με σκοπό να εκτελεστούν οι διάφορες λειτουργίες του.

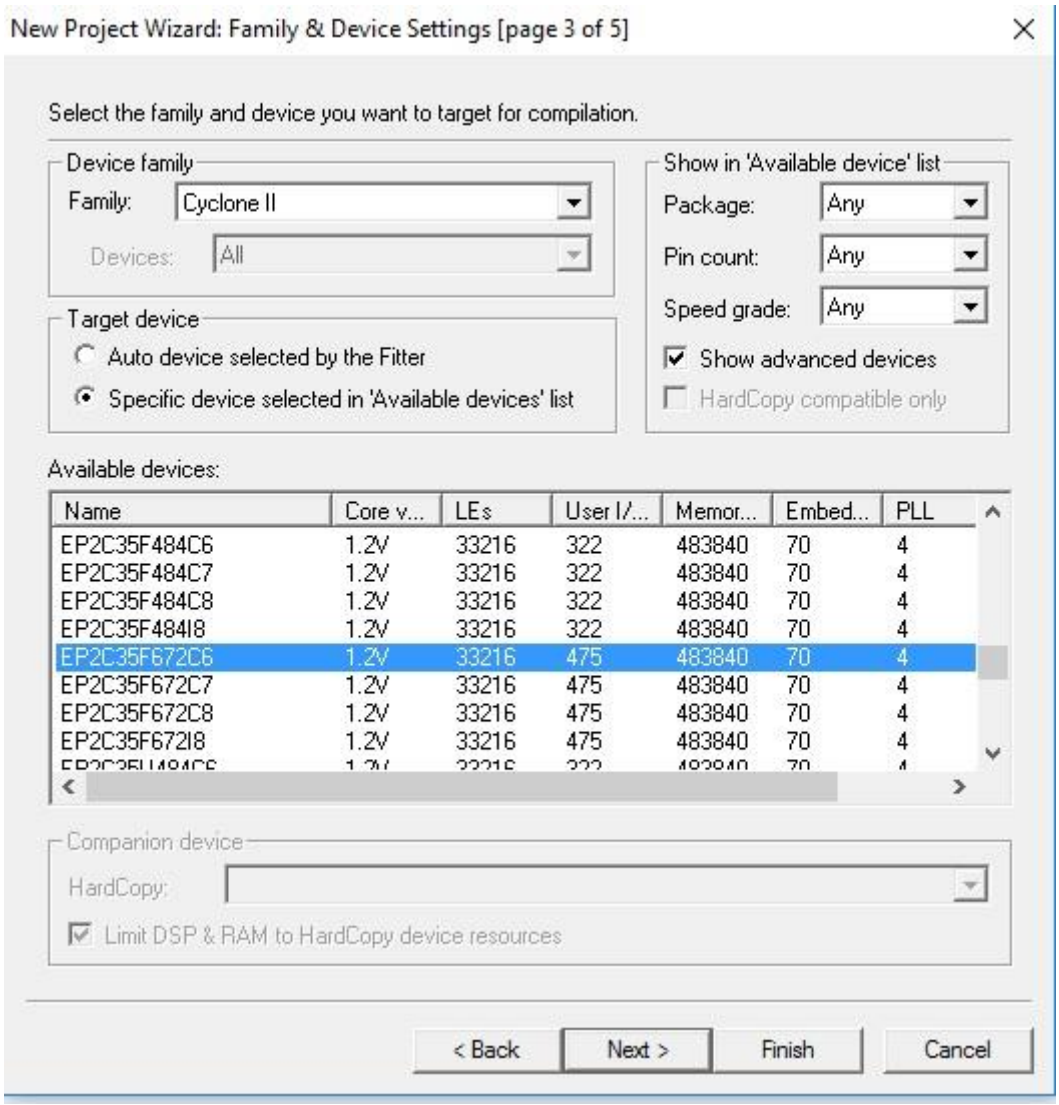


Εικόνα 3 Δήλωση ονόματος του project

Για να δημιουργηθεί ένα project από το μενού File ο χρήστης θα πρέπει να επιλέξει New Project Wizard. Στην *Εικόνα 3* δηλώνεται το όνομα του project καθώς και το όνομα του φακέλου που θα αποθηκευτεί το project. Στο συγκεκριμένο σημείο πρέπει το όνομα του project να είναι το ίδιο με το όνομα της οντότητας διαφορετικά το εργαλείο δε θα εξάγει σωστά αποτελέσματα.

2.8.2 Ορισμός προγραμματιζόμενης συσκευής

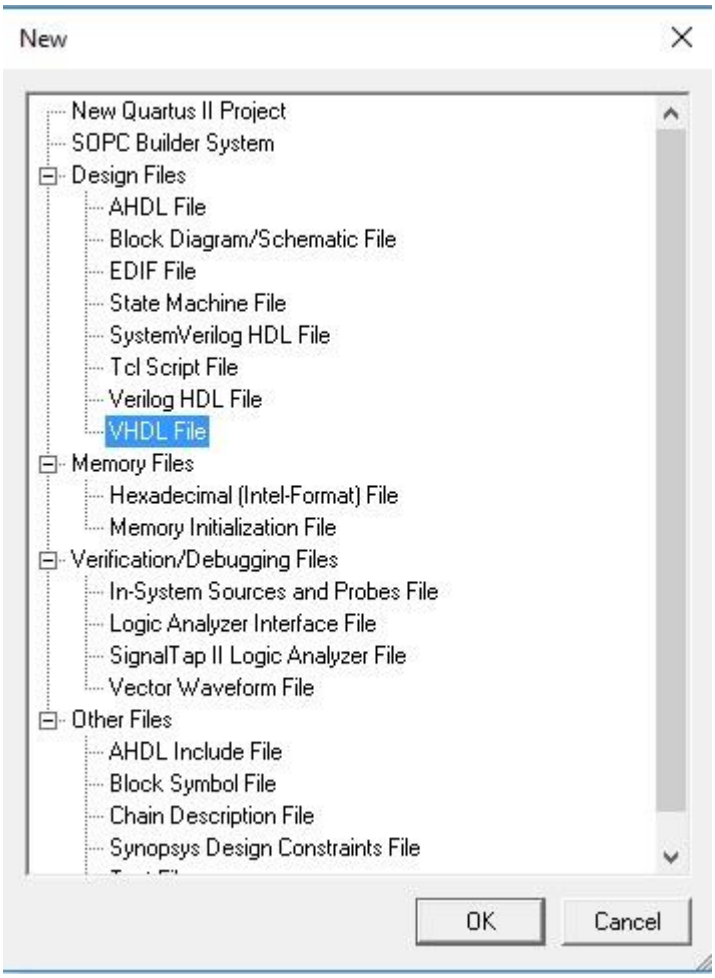
Στη συνέχεια ο χρήστης θα πρέπει να πατήσει Next ώστε να του εμφανιστεί η καρτέλα με το όνομα ‘Family’. Το FPGA που θα χρησιμοποιηθεί σ’ αυτή την εργασία είναι το Cyclone II και στην καρτέλα με το όνομα ‘Available Devices’ επιλέγεται η συσκευή EPC2C35F672C6. *(Εικόνα 4)*



Εικόνα 4 Ορισμός προγραμματιζόμενης συσκευής

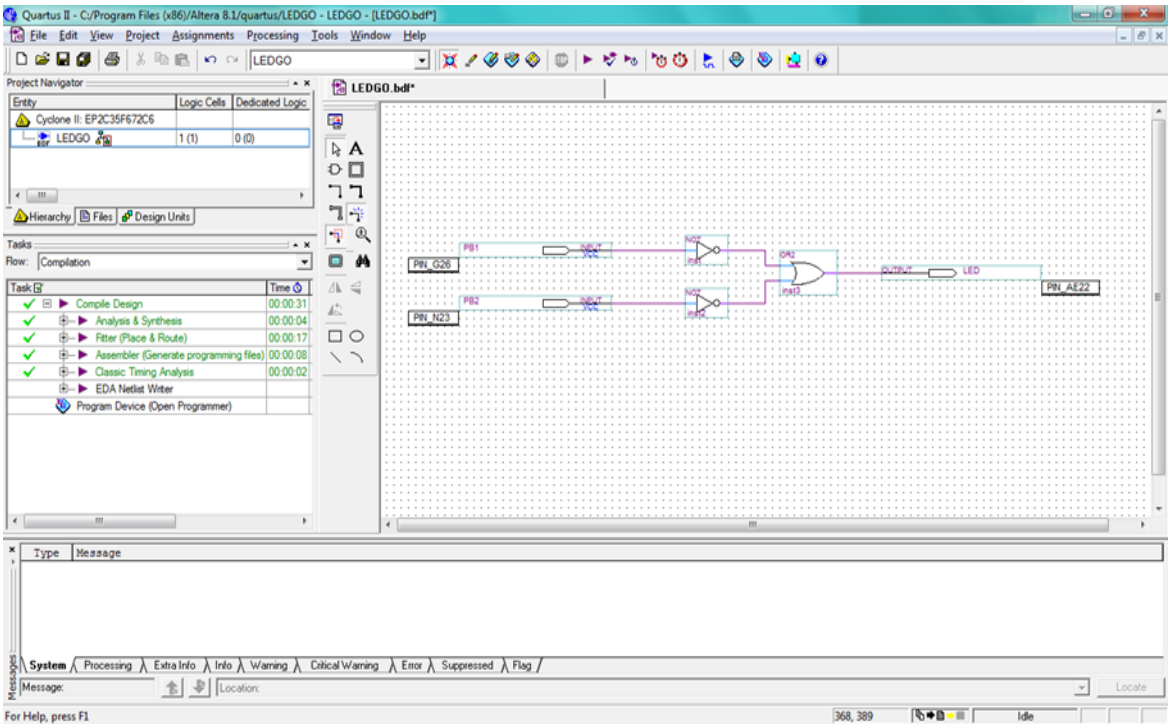
2.8.3 Εισαγωγή αρχείου

Αφού έχει δοθεί το όνομα του project στη συνέχεια ακολουθεί η εισαγωγή του κυκλώματος η οποία γίνεται με δύο τρόπους. Ο πρώτος τρόπος αναφέρεται στη σχηματική περιγραφή όπου επιλέγεται Block Diagram/Schematic File. Ενώ ο δεύτερος τρόπος είναι με τη χρήση μιας γλώσσας περιγραφής Verilog HDL ή VHDL. Η συγκεκριμένη επιλογή γίνεται από το μενού με την επιλογή File>New.



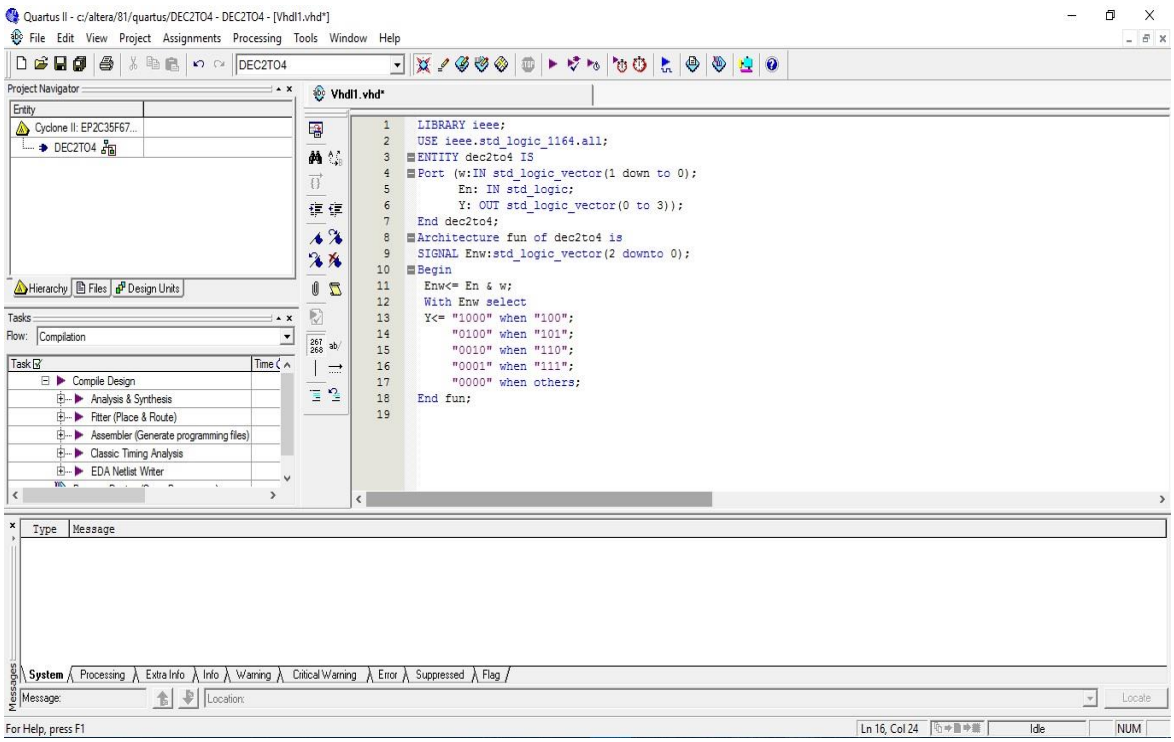
Εικόνα 5 Εισαγωγή αρχείου

Στην περίπτωση που το κύκλωμα περιγράφεται σχηματικά, για να τοποθετηθεί οποιοδήποτε σύμβολο μέσα στο πλέγμα πρέπει ο κέρσορας να βρίσκεται μέσα στο πλέγμα και κάνοντας δεξί κλικ Insert>Symbol (ή πατώντας διπλό αριστερό κλικ) εμφανίζονται υποκατηγορίες με τα εξής ονόματα : Megafunctions (περιέχει κυκλώματα, αποκωδικοποιητές κ.α.), Primitives (περιέχει τις λογικές πύλες), others. Με διπλό κλικ πάνω στις πύλες και επιλέγοντας Properties δίνονται τα ονόματα του κυκλώματος που σχεδιάστηκε.



Εικόνα 6 Σχηματική περιγραφή ενός κυκλώματος

Στην περίπτωση που το κύκλωμα περιγράφεται σε γλώσσα VHDL, εμφανίζεται ένα παράθυρο εισαγωγής κώδικα VHDL που ανοίγει στο δεξί μέρος της οθόνης με το όνομα vdhl.vhd.



Εικόνα 7 Κώδικας σε γλώσσα VHDL

2.8.4 Μετάφραση

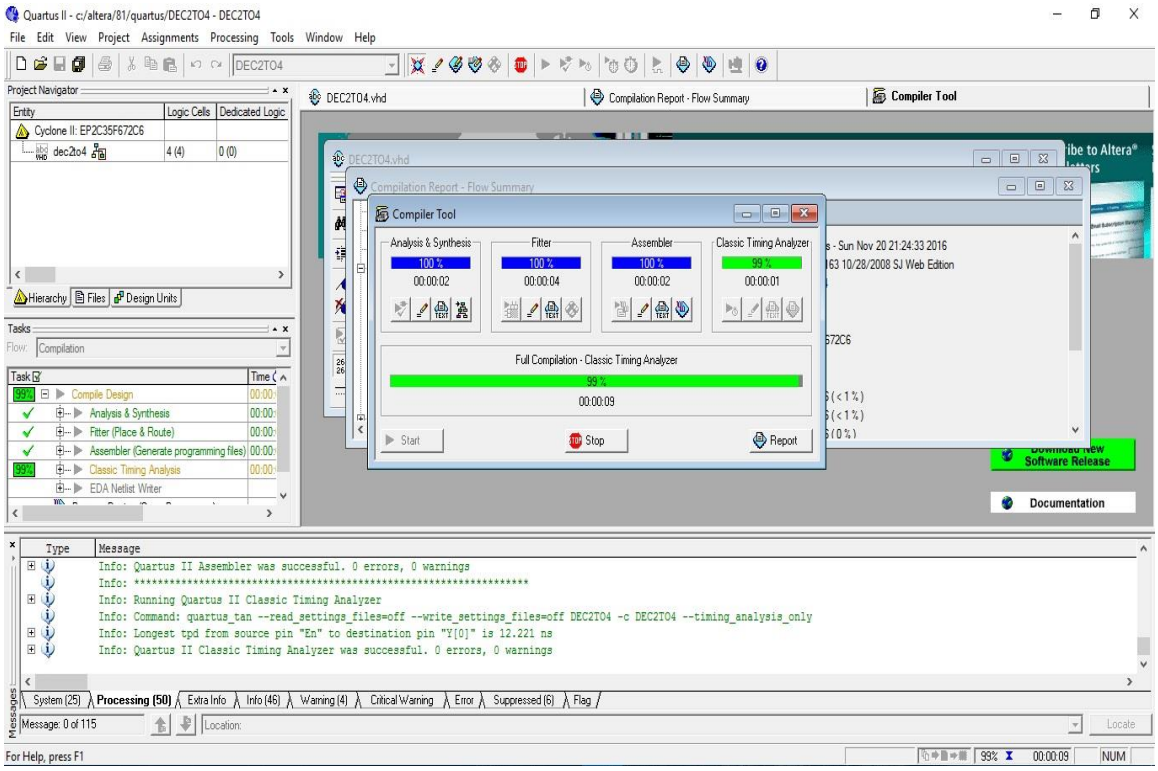
Η διαδικασία της μετάφρασης πραγματοποιείται μετά το τέλος της περιγραφής του κυκλώματος και της αποθήκευσης του. Ο μεταφραστής του Quartus II αποτελείται από ανεξάρτητα εργαλεία τα οποία ελέγχουν και αναλύουν τον κώδικα VHDL ή το σχηματικό διάγραμμα για λάθη. Επίσης, δημιουργούν μία λογική έκφραση για κάθε λογική συνάρτηση του κυκλώματος και απεικονίζουν το σχέδιο σε ένα FPGA ή ένα CLPD της ALTERA και δημιουργούν αρχεία εξόδων για προσομοίωση λειτουργίας (Simulation), χρονική ανάλυση (timing analysis), και προγραμματισμό των διατάξεων. Ο μεταφραστής αποτελείται από τα εργαλεία:

- Analysis and Synthesis
- Fitter
- Assembler
- Timing Analyzer

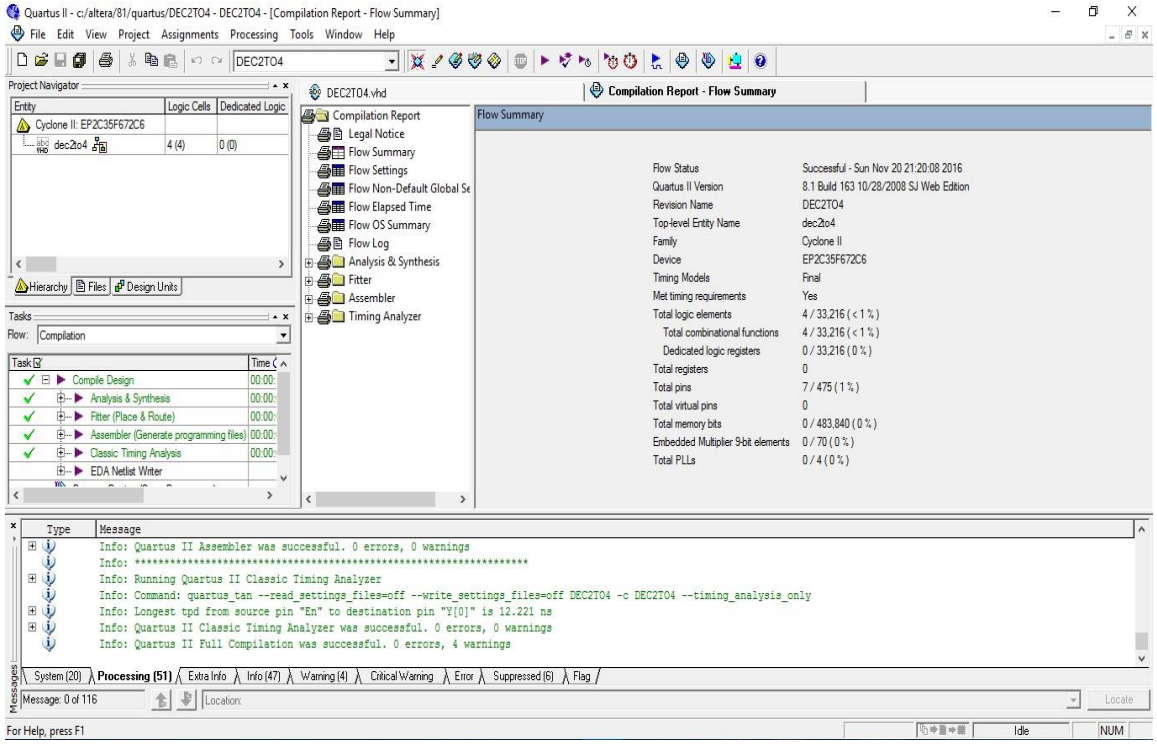
Μετά την ολοκλήρωση της εισαγωγής του κώδικα VHDL πρέπει να γίνεται έλεγχος για τυχόν ορθογραφικά ή συντακτικά λάθη στο πρόγραμμα. Επειδή η πλήρης μετάφραση μπορεί να διαρκεί αρκετό χρόνο το QUARTUS-II δίνει τη δυνατότητα ανάλυσης του κώδικα και

του προσδιορισμού τυχόν λαθών εκτελώντας μόνο το πρώτο βήμα. Με τις εντολές Processing-> Start->Start Analysis and Synthesis πραγματοποιείται η διαδικασία αυτή και αναλαμβάνει τη μετάφραση του προγράμματος (compilation). Μόλις ολοκληρωθεί η παραπάνω διαδικασία εμφανίζει στην οθόνη ένα πληροφοριακό μήνυμα για το αν ήταν επιτυχής η διαδικασία ή όχι. Στο κάτω μέρος της οθόνης εμφανίζονται διάφορα μηνύματα τα οποία χωρίζονται σε τρεις κατηγορίες: πληροφοριακά (info) με πράσινα γράμματα, προειδοποιητικά (warning) με μπλε γράμματα, και σφάλματα (errors) με κόκκινα γράμματα. Αν ο κώδικας περιέχει σφάλματα η διαδικασία ανάλυσης σταματάει και αναφέρεται ο συνολικός αριθμός των σφαλμάτων. Για την αποσφαλμάτωση του κώδικα πρέπει να προσδιοριστεί στο κάτω μέρος της οθόνης το πρώτο μήνυμα λάθους και με διπλό κλικ του αριστερού πλήκτρου του ποντικιού τοποθετείται αυτόματα ο δρομέας (cursor) στη γραμμή του κώδικα που παρουσιάζεται το σφάλμα. Η διαδικασία Start Analysis and Synthesis χρησιμοποιείται για να ελεγχθούν τυχόν εναπομείναντα λάθη. Πολλές φορές η διόρθωση ενός σφάλματος οδηγεί σε σημαντική ελάττωση του συνολικού αριθμού των σφαλμάτων.

Όταν ολοκληρωθεί η διόρθωση των σφαλμάτων η επόμενη ενέργεια που ακολουθεί είναι η μετάφραση του κώδικα. Για να ξεκινήσει η διαδικασία του μεταφραστή επιλέγονται οι εντολές Processing --> Start Compilation *(Εικόνα 8)*. Μόλις ο Compiler ολοκληρώσει τη μετάφραση, με την εντολή Processing --> Compilation Report η οθόνη εμφανίζει ένα παράθυρο στο οποίο δίνονται πληροφοριακά και στατιστικά στοιχεία για τη διαδικασία της μετάφρασης του κυκλώματος *(Εικόνα 9)*.



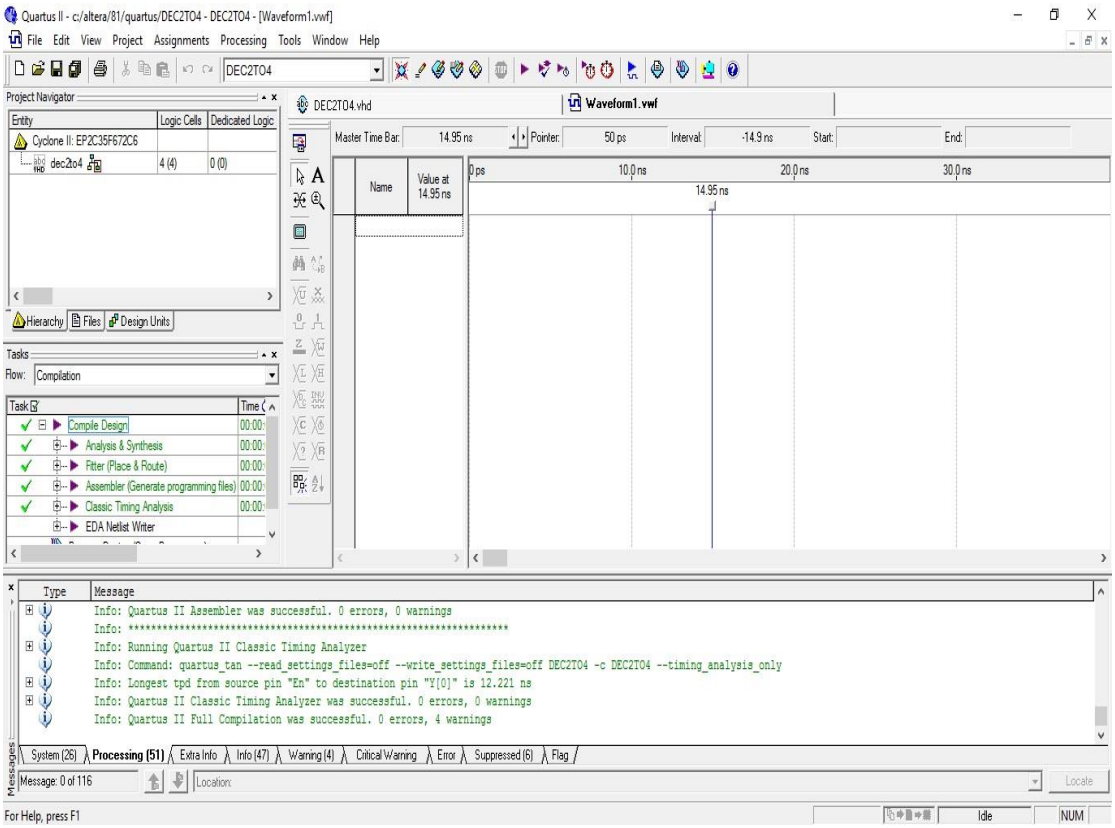
Εικόνα 8 Το εργαλείο Compiler



Εικόνα 9 Compilation Report

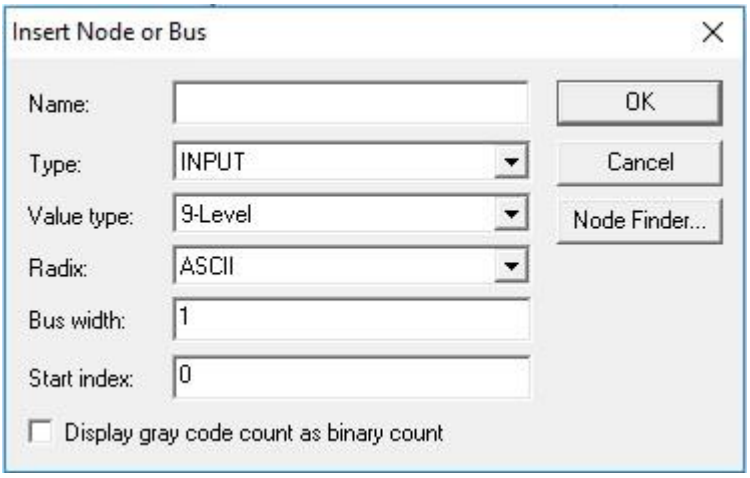
2.8.5 Προσομοίωση

Το Quartus II περιλαμβάνει ένα εργαλείο προσομοίωσης που χρησιμοποιείται για την προσομοίωση της συμπεριφοράς του σχεδιασμένου κυκλώματος. Πριν γίνει η προσομοίωση του κυκλώματος, είναι απαραίτητο να δημιουργηθούν οι κυματομορφές οι οποίες αντιπροσωπεύουν τα σήματα εισόδου. Για την σχεδίαση των διανυσμάτων δοκιμής χρησιμοποιείται ο Επεξεργαστής Κυματομορφών (Waveform Editor). Για να εκτελεστεί ο Waveform Editor επιλέγεται από το μενού File->New->Other Files->Vector Waveform File οπότε το Quartus θα μας εμφανίσει την *Εικόνα 10* που φαίνεται παρακάτω.

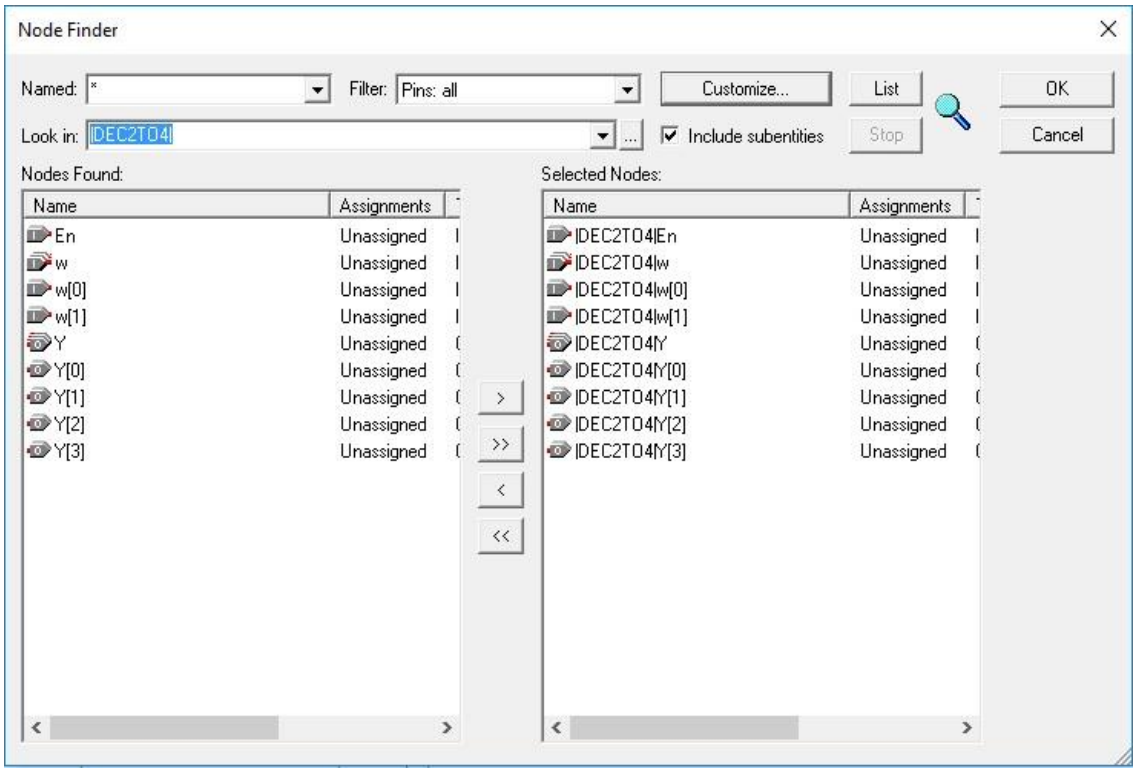


Εικόνα 10 Waveform Editor

Στη συνέχεια, πρέπει να συμπεριληφθούν οι κόμβοι εισόδου και εξόδου του κυκλώματος που θα προσομοιωθεί αυτό γίνεται με τη βοήθεια του προγράμματος Node Finder. Από το μενού επιλέγεται Edit->Insert Node or Bus εμφανίζεται η *Εικόνα 11*. Στο πλαίσιο με το όνομα Name δίνεται το όνομα ενός σήματος (pin), αλλά είναι πιο εύκολο να ανοιχτεί το παράθυρο της *Εικόνας 12* πατώντας κλικ στο κουμπί με την ετικέτα Node Finder.



Εικόνα 11 Παράθυρο διαλόγου εισαγωγής κόμβου ή διαύλου

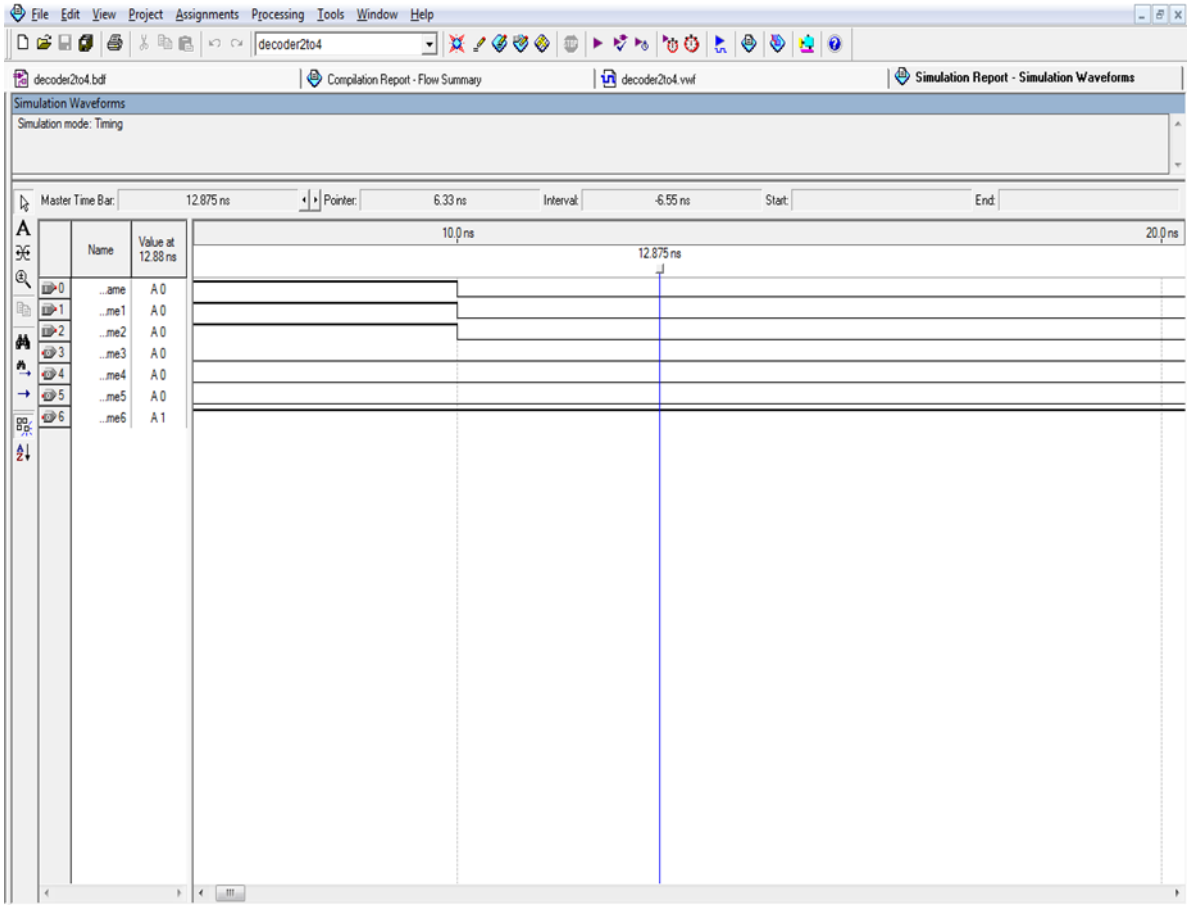


Εικόνα 12 Node Finder

Ο Node Finder απεικονίζει στην αριστερή μεριά του παραθύρου τους κόμβους καθώς και μία σειρά εργαλείων που χρησιμοποιούνται για να καθοριστούν οι λογικές τιμές των σημάτων εισόδου. Οι τιμές στις εξόδους θα δημιουργηθούν αυτόματα από τον εξομοιωτή. Χρησιμοποιώντας τα εργαλεία του Editor, υπάρχει η δυνατότητα δημιουργίας κατάλληλων κυματομορφών εισόδου, θέτοντας τις τιμές των εισόδων σε 1 ή 0, για συγκεκριμένα χρονικά

διαστήματα. Το αρχείο των διανυσμάτων εισόδου (Waveform vector file) αποθηκεύεται με όνομα Part1.wvf και πρέπει να έχει το ίδιο όνομα με το όνομα του entity.

Για να εκτελεστεί η προσομοίωση του κυκλώματος εκτελείται από το μενού η εντολή Processing->Start Simulation. Στη συνέχεια ανοίγει ένα παράθυρο που ονομάζεται Simulation Report στο αριστερό τμήμα του οποίου αναφέρονται στατιστικά στοιχεία για τη διαδικασία της προσομοίωσης και στο δεξί τμήμα του φαίνονται τα αποτελέσματα της προσομοίωσης της λειτουργίας του κυκλώματος *(Εικόνα 13)*. Στο σημείο αυτό, θα πρέπει να ελεγχθεί αν οι τιμές από την εκτέλεση της προσομοίωσης ανταποκρίνονται στις θεωρητικές τιμές που εμφανίζονται με βάση τον πίνακα αληθείας του κυκλώματος.

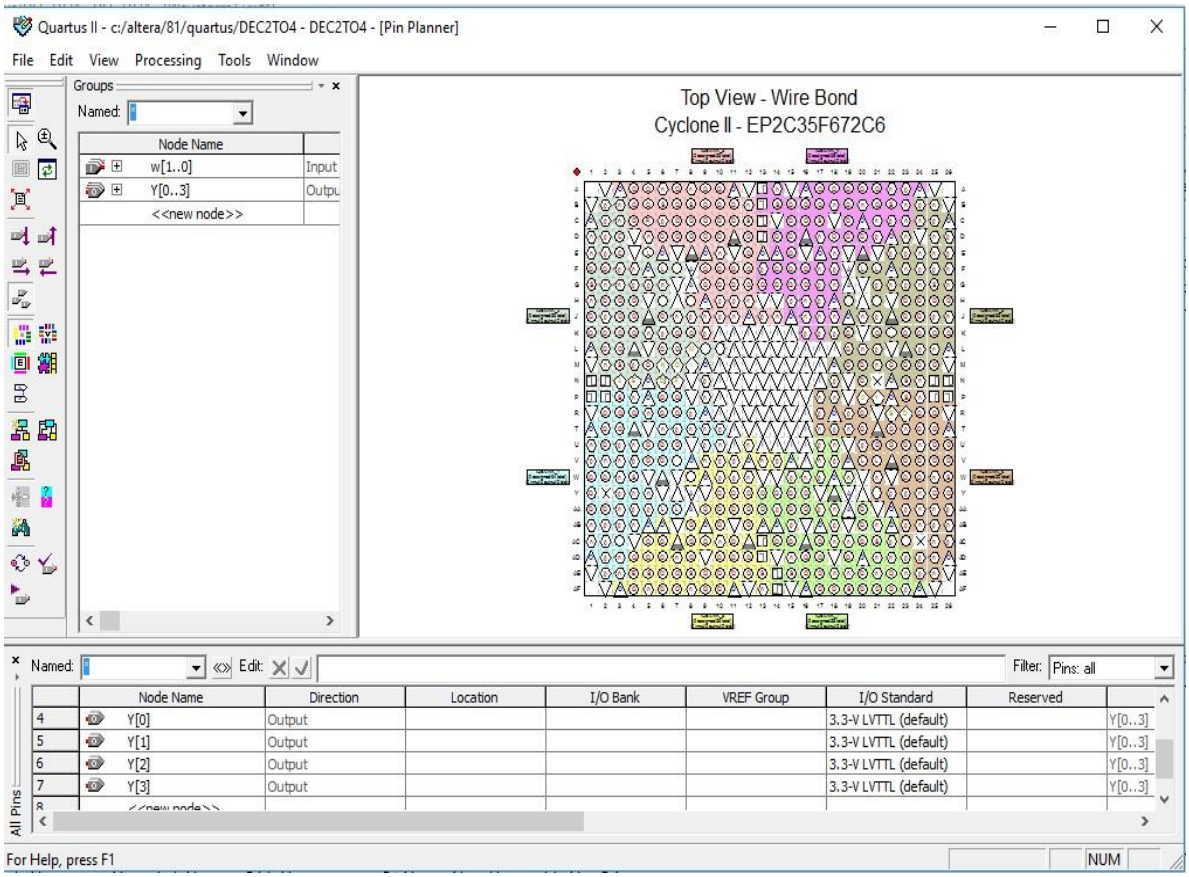


Εικόνα 13 Αρχείο Κυματομορφών

2.8.6 Ορισμός ακροδεκτών

Μόλις ολοκληρωθεί η διαδικασία της προσομοίωσης το επόμενο βήμα είναι η αντιστοίχιση των εισόδων αλλά και των εξόδων του κυκλώματος με συγκεκριμένους ακροδέκτες της διάταξης που πρόκειται να προγραμματιστεί. Η διάταξη που θα χρησιμοποιηθεί παρακάτω είναι ένα FPGA και ανήκει στην οικογένεια Cyclone II της Altera, συγκεκριμένα η διάταξη είναι EP2C35F672C6.

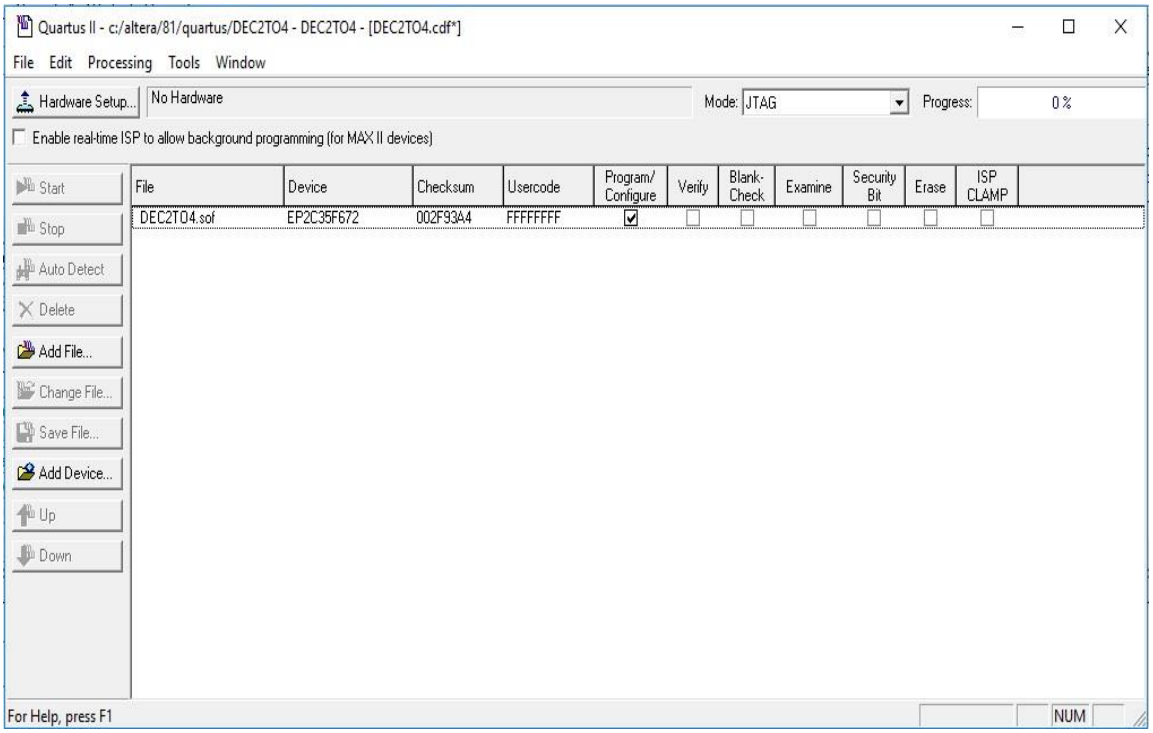
Με την επιλογή της εντολής Assignments->Pins ορίζονται οι ακροδέκτες του τσιπ στους οποίους θα συνδεθούν οι είσοδοι και οι έξοδοι του κυκλώματος. Στο πεδίο Node Name εμφανίζονται τα ονόματα που δόθηκαν προηγουμένως στους ακροδέκτες στο σχηματικό διάγραμμα. Κάνοντας διπλό κλικ στο πεδίο Location εμφανίζεται η αρίθμηση όλων των ακροδεκτών της συγκεκριμένης διάταξης. Οι επιλογές αυτές προκύπτουν από τα σχηματικά διαγράμματα του κυκλώματος που πρόκειται να προγραμματιστεί *(Εικόνα 14)*.



Εικόνα 14 Pin Planner

2.8.7 Προγραμματισμός κυκλώματος

Μετά την ολοκλήρωση του ορισμού των ακροδεκτών θα πρέπει να γίνει η μετάφραση ώστε να δημιουργηθεί το αρχείο *.sof το οποίο περιέχει όποιες πληροφορίες χρειάζονται για την διαμόρφωση του FPGA. Η διαμόρφωση γίνεται με τον Programmer, που βρίσκεται στο μενού Tools->Programmer *(Εικόνα 15)*. Θα πρέπει να γίνουν οι κατάλληλες επιλογές του υλικού (Hardware Setup) για την επιλογή του κατάλληλου κυκλώματος προγραμματισμού. Οι πιο συνηθισμένες επιλογές είναι ο οδηγός BYTE-BLASTER για προγραμματισμό μέσω της παράλληλης θύρας ή ο USB-BLASTER για προγραμματισμό μέσω της θύρας USB. Είναι απαραίτητο να επιλεγεί το τελικό αρχείο διαμόρφωσης (.sof) που δημιουργήθηκε κατά το τελικό στάδιο της μετάφρασης, καθώς και η επιλογή Program/Configure.



Εικόνα 15 Programmer

ΚΕΦΑΛΑΙΟ 3

Ψηφιακά Ολοκληρωμένα Κυκλώματα

3.1 Διατάξεις Προγραμματιζόμενης Λογικής (PLDs)

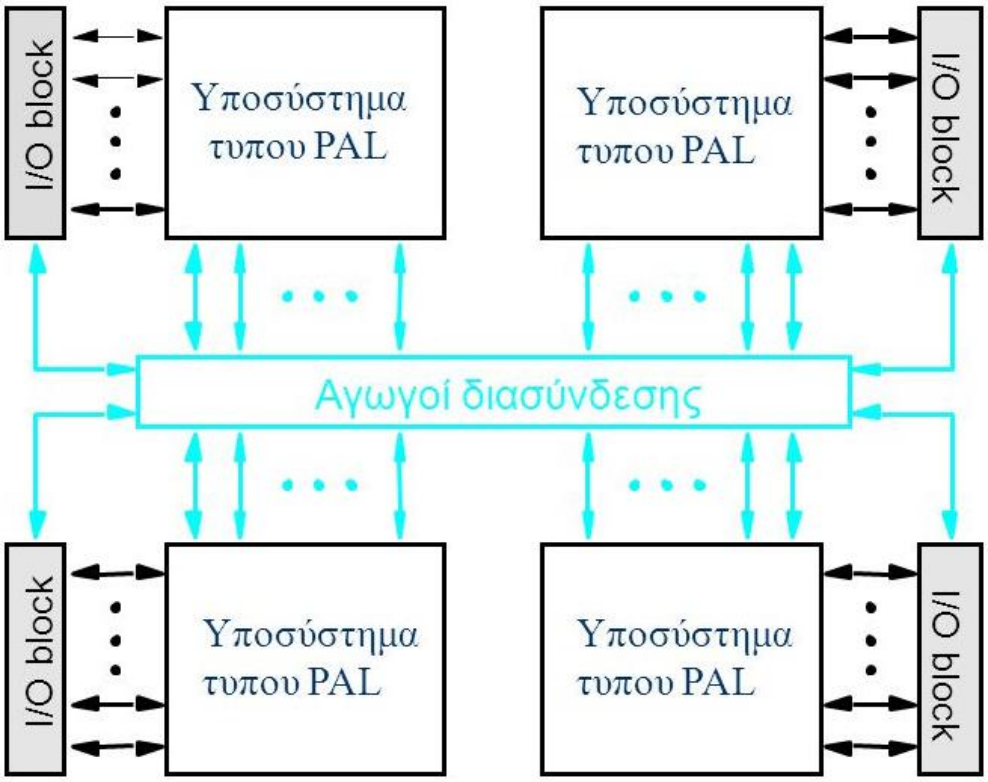
Στα μέσα της δεκαετίας του 1970 εμφανίστηκαν τα προγραμματιζόμενα λογικά στοιχεία (Programmable Logic Devices – PLDs). Η βασική ιδέα ήταν η κατασκευή συνδυαστικών λογικών κυκλωμάτων που να είναι προγραμματιζόμενα. Τα PLDs μπορούν να διαμορφωθούν από τον χρήστη με στόχο την υλοποίηση λογικών κυκλωμάτων. Ακόμη, είναι δυνατό να πραγματοποιήσουν οποιαδήποτε Boolean έκφραση ή συνάρτηση με τη χρήση των λογικών δομών που είναι ήδη υλοποιημένες. Από την άλλη μεριά, τα συνήθη ολοκληρωμένα κυκλώματα παρέχουν μια συγκεκριμένη λογική συνάρτηση η οποία δεν μπορεί να τροποποιηθεί ώστε να ικανοποιήσει συγκεκριμένη απαίτηση ενός σχεδιασμού. Τα τελευταία χρόνια ολοένα και περισσότεροι χρήστες χρησιμοποιούν τα PLDs σε διάφορες εφαρμογές καθώς το κόστος τους έχει μειωθεί σημαντικά λόγω της χρήσης νέων τεχνολογιών. Σημαντική παρατήρηση αποτελεί το γεγονός ότι όλα τα PLDs είναι δυνατό να προγραμματιστούν μόνο μία φορά με τη χρήση ασφαλειών ή αντισφαλειών, ή μπορούν να επαναπρογραμματιστούν χρησιμοποιώντας μνήμες EEPROM ή Flash. Τα PLDs προσφέρονται σε διάφορες αρχιτεκτονικές και με ένα μεγάλο πλήθος από τεχνολογίες μνημών για τον προγραμματισμό τους. Μία από τις σημαντικότερες κατηγορίες PLDs είναι τα FPGAs (Field Programmable Gate Arrays).

3.2 Πολύπλοκες Διατάξεις Προγραμματιζόμενης Λογικής (CLPDs)

Το CPLD (Complex Programmable Logic Devices) είναι μία Σύνθετη Προγραμματιζόμενη Λογική Διάταξη που αποτελείται από μια συλλογή απλών PLDs πάνω σε ένα chip. Για την σύνδεση των PLD μεταξύ τους καθώς και με τους ακροδέκτες εισόδου και εξόδου χρησιμοποιείται ένας προγραμματιζόμενος πίνακας διακοπτών. Ο σκοπός της δημιουργίας των CPLD ήταν η υλοποίηση κυκλωμάτων, που απαιτούν περισσότερες εισόδους και εξόδους σε σχέση με τα PLD.

Ένα CPLD αποτελείται από πολλές βαθμίδες κυκλωμάτων οι οποίες βρίσκονται στο ίδιο ολοκληρωμένο σύστημα και ο τρόπος με τον οποίο συνδέονται είναι οι εσωτερικές καλωδιώσεις. Στο *Σχήμα 1* παρουσιάζεται ένα παράδειγμα ενός CPLD το οποίο περιέχει τέσσερις βαθμίδες PLD που συνδέονται μεταξύ τους με εσωτερικές καλωδιώσεις. Επιπροσθέτως, κάθε βαθμίδα PLD συνδέεται και με ένα υποκύκλωμα, που ονομάζεται

βαθμίδα εισόδου/εξόδου (I/O BLOCK) και προσαρμόζεται σε έναν αριθμό ακροδεκτών εισόδου και εξόδου του ολοκληρωμένου κυκλώματος.



Σχήμα 1 Δομή ενός CPLD

Όπως και στα PLD έτσι και τα CLPD περιέχουν προγραμματιζόμενους διακόπτες της ίδιας μορφής. Οι συγκεκριμένοι διακόπτες προγραμματίζονται με την τοποθέτηση ενός ολοκληρωμένου κυκλώματος σε μια ειδική μονάδα προγραμματισμού. Εάν, ένα κύκλωμα CLPD είναι μεγάλο, η συγκεκριμένη μέθοδος προγραμματισμού θεωρείται αρκετά δύσκολη για δύο λόγους: πρώτον επειδή τα μεγάλα κυκλώματα CLPD διαθέτουν παραπάνω από 200 ακροδέκτες, οι οποίοι είναι εύθραυστοι και εύκαμπτοι. Κατά δεύτερον λόγο, το κύκλωμα CLPD πρέπει να τοποθετηθεί σε μια κατάλληλη βάση ώστε να πραγματοποιηθεί αλλά τέτοιες βάσεις είναι πολύ ακριβές και συνήθως κοστίζουν πιο ακριβά και από το κύκλωμα.

Για τους δύο λόγους που αναφέρθηκαν παραπάνω, οι διατάξεις CLPD υποστηρίζουν την τεχνική ISP σύμφωνα με την οποία η μητρική πλακέτα που περιέχει το κύκλωμα CLPD, περιέχει ένα μικρό συνδετήρα καθώς κι ένα καλώδιο που συνδέει το σύστημα του υπολογιστή με τον συγκεκριμένο συνδετήρα. Τα κυκλώματα CLPD που επιτρέπουν την

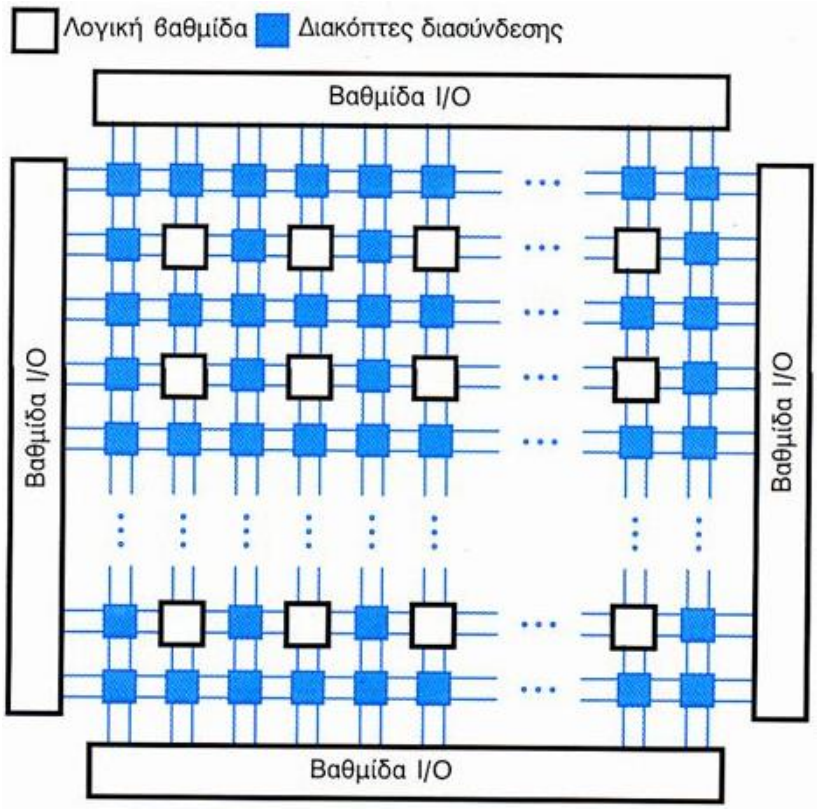
συγκεκριμένη μέθοδο προγραμματισμού έχουν τροποποιηθεί από το ινστιτούτο IEEE και ονομάζεται JTAG port. Η συγκεκριμένη θύρα χρησιμοποιεί τέσσερα καλώδια για να γίνει η μεταφορά της πληροφορίας από τον υπολογιστή και της διάταξης που επρόκειτο να πραγματοποιηθεί.

Μια τελευταία παρατήρηση που θα πρέπει να διατυπωθεί είναι ότι, από τη στιγμή που θα πραγματοποιηθεί ένα CLPD, η πληροφορία που έχει τοποθετηθεί σε αυτή διατηρείται μόνιμα ακόμη κι αν διακοπεί η τροφοδοσία του ολοκληρωμένου κυκλώματος.

3.3 Διατάξεις Πυλών Προγραμματιζόμενου Πεδίου (FPGA)

Τα FPGAs (Field Programmable Gate Arrays) είναι διατάξεις προγραμματιζόμενης λογικής που υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων, ενώ ταυτόχρονα επιτρέπουν πλήρη ελευθερία όσον αφορά τη διαδικασία σχεδίασης. Οι διατάξεις FGPA διαφέρουν σημαντικά σε σχέση με τα CPLD και τα PLD επειδή δεν περιέχουν πύλες AND και OR, αντιθέτως περιέχουν λογικές βαθμίδες για την υλοποίηση των ζητούμενων συναρτήσεων.

Η γενική δομή ενός FPGA περιέχει τρία είδη πόρων: τις λογικές βαθμίδες, τις γραμμές εσωτερικής διασύνδεσης και τις βαθμίδες εισόδου/εξόδου οι οποίες χρησιμοποιούνται για τη σύνδεση με τους ακροδέκτες της συσκευασίας και τους διακόπτες. Οι λογικές βαθμίδες οργανώνονται με τη μορφή δισδιάστατης σειράς και οι γραμμές διασύνδεσης οργανώνονται ως οριζόντια και κατακόρυφα κανάλια δρομολόγησης ανάμεσα στις γραμμές και στήλες των λογικών βαθμίδων. Τα κανάλια δρομολόγησης επιτρέπουν στις λογικές βαθμίδες να διασυνδέονται με πολλούς τρόπους. Στο *Σχήμα 2* παρουσιάζεται η γενική δομή ενός FPGA όπου τα τετράγωνα που υπάρχουν δίπλα από τις λογικές βαθμίδες εμπεριέχουν διακόπτες οι οποίοι συνδέουν τους ακροδέκτες εισόδου/εξόδου των λογικών βαθμίδων με τα καλώδια διασύνδεσης. Από την άλλη, τα τετράγωνα που βρίσκονται διαγώνια από τις λογικές βαθμίδες συνδέουν ένα καλώδιο διασύνδεσης με ένα άλλο.



Σχήμα 2 Γενική Δομή ενός FPGA

Οι γλώσσες προγραμματισμού που χρησιμοποιούνται για την περιγραφή ενός FPGA είναι η AHDL, Verilog και VHDL. Αν διακοπεί η τροφοδοσία του FPGA χάνει τον προγραμματισμό του, για το λόγο αυτό απαιτεί έναν μικροεπεξεργαστή από τον οποίο θα προγραμματίζεται κάθε φορά που επανέρχεται η τάση τροφοδοσίας. Ο προγραμματισμός του FPGA μπορεί να αλλάζει κάθε φορά που τροποποιείται το λογισμικό του μικροεπεξεργαστή.

3.4 Το Cyclone II FPGA

Στην παρούσα εργασία θα χρησιμοποιηθεί το Cyclone II EP2C35F672C6 της Altera το οποίο είναι ένα ολοκληρωμένο που ενσωματώνεται στην αναπτυξιακή πλατφόρμα DE2. Το συγκεκριμένο FPGA περιέχει 33.216 λογικά στοιχεία (Logic Elements-LEs), 35 ενσωματωμένους πολλαπλασιαστές, 4 PLLs και 475 ακίδες εισόδου/εξόδου. Το συγκεκριμένο FPGA έχει υλοποιηθεί σε wafer 300nm χρησιμοποιώντας την τεχνολογία TSMC’s 90nm.

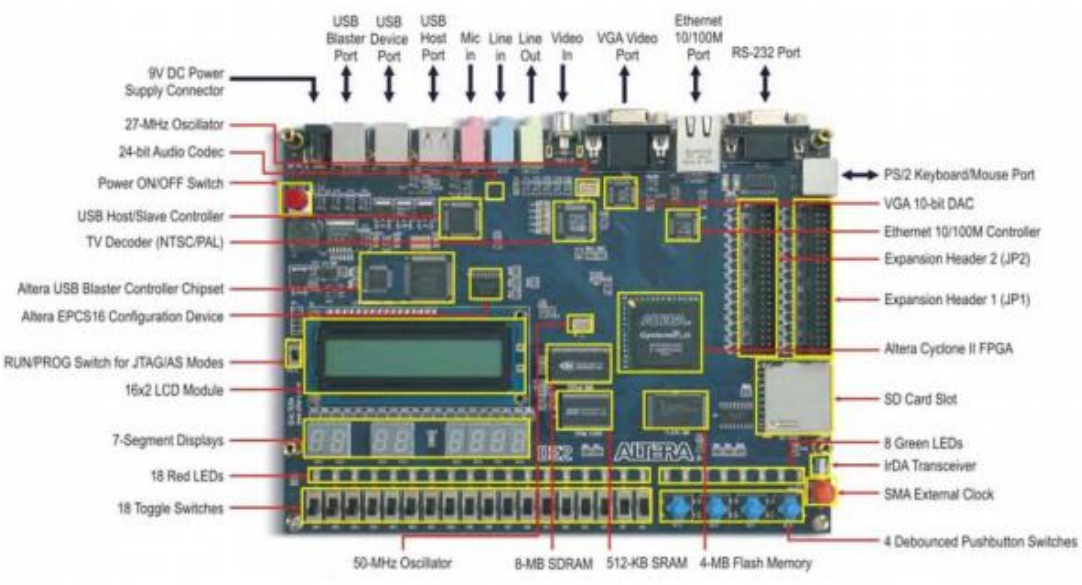
Η οικογένεια Cyclone II μειώνοντας το συνολικό μέγεθος πυριτίου, μπορεί να υποστηρίξει σύνθετα ψηφιακά συστήματα σε ένα ολοκληρωμένο με κόστος κατασκευής πολύ μικρότερο της τεχνολογίας κατασκευής ASIC. Επίσης, η Cyclone II των συγκεκριμένων ολοκληρωμένων προσφέρει 60% μεγαλύτερη απόδοση, με το ίδιο κόστος και με ταυτόχρονη μείωση κατανάλωσης της ενέργειας στο μισό σε σύγκριση με FPGA άλλων κατασκευαστών.

Η αρχιτεκτονική της οικογένειας Cyclone II είναι οργανωμένη σε δύο διαστάσεις σε γραμμές και στήλες. Μεταξύ των δύο διαστάσεων υπάρχουν διασυνδέσεις στις διάφορες λογικές δομές (LABs) τις ενσωματωμένες δομές μνήμης και τους ενσωματωμένους πολλαπλασιαστές. Οι λογικές δομές (LABs) αποτελούνται από 16 λογικά στοιχεία (LEs) η κάθε μία. Το LE είναι μια μικρή μονάδα λογικής που επιτρέπει την αποδοτική υλοποίηση στοιχειωδών λογικών συναρτήσεων. Υπάρχουν από 4.608 έως 68,416 LE σε ένα chip Cyclone II.

Η οικογένεια Cyclone II ενσωματώνει ένα κεντρικό δίκτυο ρολογιού και τέσσερα PLLs. Το κεντρικό δίκτυο ρολογιού αποτελείται από 16 γραμμές ρολογιού, οι οποίες οδηγούν ολόκληρο το FPGA. Επίσης, περιλαμβάνει δομές μνήμης διπλής εισόδου με συνολική χωρητικότητα 4K- bits. Οι συγκεκριμένες δομές βρίσκονται τοποθετημένες σε στήλες κατά μήκος του ολοκληρωμένου μεταξύ συγκεκριμένων LABs και έχουν τη δυνατότητα λειτουργίας σε συχνότητες μέχρι 260 MHz. Οι πολλαπλασιαστές που είναι τοποθετημένοι στο FPGA μπορούν να χρησιμοποιηθούν είτε ως δύο 9x9 πολλαπλασιαστές είτε ως ένας 16x16 πολλαπλασιαστής, με ταχύτητα έως και 250 MHz.

3.5 Βασικά χαρακτηριστικά του board DE2

Στο *Σχήμα 3* παρουσιάζεται η αναπτυξιακή πλακέτα DE2 με όλα τα υποσυστήματα, τις μνήμες καθώς και τα περιφερειακά που διαθέτει. Όπως φαίνεται και στο σχήμα, η πλατφόρμα διαθέτει μια σειρά από περιφερειακά για διάφορες εφαρμογές, όπως παραδείγματος χάριν εφαρμογές εικόνας, ήχου και μετάδοσης δεδομένων. Όλες αυτές οι εφαρμογές μπορούν να υλοποιηθούν μέσω αυτής της πλατφόρμας κάνοντας χρήση του Cyclone II FPGA που διαθέτει.



Σχήμα 3 Η αναπτυξιακή πλακέτα DE2 της Altera

Πιο αναλυτικά η κάρτα DE2 περιλαμβάνει:

• Το FPGA Cyclone II 2C35 της Altera.

• Τη συσκευή σειριακής μορφοποίησης EPCS16 της Altera.

• Το USB Blaster Port (on board) για προγραμματισμό. Υποστηρίζονται το JTAG και το Active Serial τύποι προγραμματισμού.

• SRAM μνήμη των 512 Kbyte.

• SDRAM μνήμη των 8 MByte.

• FLASH μνήμη των 4 MByte.

• Υποδοχή καρτών SD.

• Τέσσερις διακόπτες πλήκτρα (push buttons).

• Δεκαοχτώ κόκκινα LEDs.

• Εννέα πράσινα LEDs.

• Δύο κρυστάλλους στα 50MHz και 27MHz.

• Ένα Audio CODEC των 24-bits.

• Ένα VGA video DAC υψηλής ταχύτητας 10-bits.

• Μία οθόνη LCD 16 χαρακτήρων και 2 γραμμών.

• Ένα TV Decoder με TV είσοδο.

• Ένα 10/100 Ethernet ελεγκτή.

• Ένα USB Host/Slave ελεγκτή με δυνατότητα διασύνδεσης τύπου A και B.

• Ένα RS232 πομποδέκτη 9 ακίδων.

• Διασύνδεση ποντικιού/πληκτρολογίου τύπου PS/2.

• Ένα πομπό υπέρυθρων IrDA.

• Δύο κεφαλές επέκτασης (expansion headers) των 40 ακίδων η κάθε μία.

ΚΕΦΑΛΑΙΟ 4

Εργαστηριακές Ασκήσεις

4.1 Εισαγωγή

Στην παρούσα πτυχιακή εργασία παρατίθενται εργαστηριακές εφαρμογές (labs) που αφορούν την εκμάθηση της γλώσσας VHDL. Τα labs διατίθενται από την εταιρεία Altera και συγκεκριμένα από πανεπιστημιακό πρόγραμμα.

Οι συγκεκριμένες εφαρμογές επιτυγχάνονται με τη βοήθεια του προγράμματος Quartus II που διατίθεται από την εταιρεία Altera. Το Quartus II χρησιμοποιείται για την προσομοίωση και την υλοποίηση των κυκλωμάτων.

Για κάθε κύκλωμα αναλύονται τα κύρια χαρακτηριστικά του, η αναπαράσταση της λογικής του σχεδίασης, ο πίνακας αληθείας καθώς και το σύμβολο του. Επίσης, παρουσιάζεται και ο κώδικας VHDL του κάθε κυκλώματος μέσα από τον οποίο περιγράφεται το κάθε κύκλωμα.

4.2 Διακόπτες και Φωτεινοί ενδείκτες (Switches and Lights)

Το περιεχόμενο του κυκλώματος αναφέρεται σε εντολές αντιστοίχισης ενός αριθμού διακοπτών σε έναν αριθμό LEDs. Στο συγκεκριμένο κύκλωμα περιλαμβάνονται 18

διακόπτες που ονομάζονται SW17-0 που χρησιμοποιούνται ως είσοδοι και 18 κόκκινα LEDs με την ονομασία LEDR17-0 τα οποία αποτελούν τις εξόδους.

Ο κώδικας του κυκλώματος χωρίζεται σε τρία μέρη. Το πρώτο μέρος είναι το τμήμα των βιβλιοθηκών όπου δηλώνονται ποιες βιβλιοθήκες θα χρησιμοποιηθούν. Στο κύκλωμα δηλώνεται η βιβλιοθήκη IEEE.std_1164.all η οποία περιέχει τον ορισμό του τύπου std_logic_vector. Στο δεύτερο μέρος περιέχεται το τμήμα της οντότητας στο οποίο υλοποιείται το part1 που αποτελείται από τα σήματα εισόδου (SW17)-(SW0) καθώς και τα σήματα εξόδου (LEDR17)-(LEDR0). Η αρχιτεκτονική αποτελεί το τρίτο και τελευταίο μέρος του κυκλώματος που ονομάζεται structure και αναφέρεται στην οντότητα part1. Στο συγκεκριμένο τμήμα γίνεται η αντιστοίχιση κάθε διακόπτη SW σε κάθε LEDR.

Ο κώδικας VHDL:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY part1 IS
PORT (SW: IN STD_LOGIC_VECTOR (17 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (17 DOWNTO 0));
END part1;
ARCHITECTURE Structure OF part1 IS
BEGIN
LEDR <= SW;
END Structure;
```

Πρόγραμμα 1 Switches and Lights

Στο *Σχήμα 4*, φαίνεται η μορφή του κυκλώματος. Η κύρια έννοια της σύνταξης του κυκλώματος SW[17..0] και LEDR[17…0] είναι ότι τα σήματα εισόδου SW και τα σήματα εξόδου LEDR αντιστοιχούν σε 18 bits, τα οποία ονομάζονται SW[17], SW[16], ….., SW[0] και LEDR[17], LEDR[16], …., LEDR[0] αντίστοιχα.

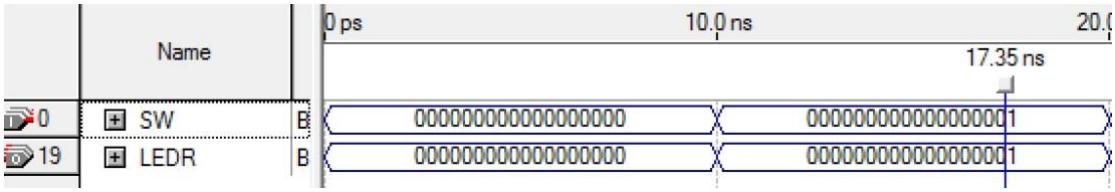
Σχήμα 4 Κύκλωμα αντιστοίχισης 18 SWs σε 18 LEDRs

Για να υλοποιηθεί το κύκλωμα χρειάζεται πρώτα να γίνει η αντιστοίχιση των ακροδεκτών, ώστε να αντιστοιχηθούν με το FPGA οι είσοδοι και οι έξοδοι που ορίστηκαν.

Signal Name	FPGA Pin No.	Signal Name	FPGA Pin No.
SW[0]	PIN_N25	LEDR[0]	PIN_AE23
SW[1]	PIN_N26	LEDR[1]	PIN_AF23
SW[2]	PIN_P25	LEDR[2]	PIN_AB21
SW[3]	PIN_AE14	LEDR[3]	PIN_AC22
SW[4]	PIN_AF14	LEDR[4]	PIN_AD22
SW[5]	PIN_AD13	LEDR[5]	PIN_AD23
SW[6]	PIN_AC13	LEDR[6]	PIN_AD21
SW[7]	PIN_C13	LEDR[7]	PIN_AC21
SW[8]	PIN_B13	LEDR[8]	PIN_AA14
SW[9]	PIN_A13	LEDR[9]	PIN_Y13
SW[10]	PIN_N1	LEDR[10]	PIN_AA13
SW[11]	PIN_P1	LEDR[11]	PIN_AC14
SW[12]	PIN_P2	LEDR[12]	PIN_AD15
SW[13]	PIN_T7	LEDR[13]	PIN_AE15
SW[14]	PIN_U3	LEDR[14]	PIN_AF13
SW[15]	PIN_U4	LEDR[15]	PIN_AE13
SW[16]	PIN_V1	LEDR[16]	PIN_AE12
SW[17]	PIN_V2	LEDR[17]	PIN_AD12

Πίνακας 5 Αντιστοίχιση Ακροδεκτών

Τα αποτελέσματα της προσομοίωσης του κυκλώματος δείχνουν ότι, όταν ένας από τους διακόπτες εισόδου είναι στην λογική κατάσταση 1, τότε στην ίδια κατάσταση θα βρίσκεται και η αντίστοιχη έξοδος. *(Σχήμα 5)*



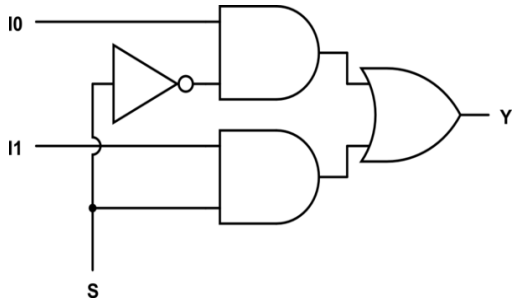
Σχήμα 5 Προσομοίωση του κυκλώματος Switches and Lights

4.3 Πολυπλέκτες (Multiplexers)

Ένα κύκλωμα πολυπλέκτη διαθέτει έναν αριθμό εισόδων δεδομένων (data inputs), μία ή περισσότερες εισόδους επιλογής δεδομένων (data select inputs) και μία έξοδο (output). Η κύρια εφαρμογή του πολυπλέκτη είναι η επιλογή μίας από τις πολλές πληροφορίες που

εφαρμόζονται στις εισόδους του και η μεταφορά της στην έξοδό του. Ο συμβολισμός 2^nx1 σημαίνει ότι ο πολυπλέκτης έχει 2^n εισόδους και μία έξοδο.

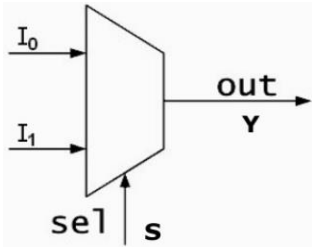
Στο *Σχήμα 6* παρουσιάζεται ένας πολυπλέκτης *δύο προς ένα* ο οποίος αποτελείται από δύο εισόδους, μία είσοδο επιλογής και μία έξοδο. Η είσοδος επιλογής (select input, s) επιλέγει ως έξοδο του πολυπλέκτη την είσοδο I0 ή την είσοδο I1. Αναλόγως με την τιμή της εισόδου επιλογής S, μία από τις εισόδους I0 και I1 μεταβιβάζεται στην έξοδο Y. Δηλαδή εάν S=0 τότε Y=I0 ενώ αν S=1 τότε Y=I1.



α) Κύκλωμα πολυπλέκτη 2 προς 1

S	Y
0	I0
1	I1

β) Πίνακας αληθείας

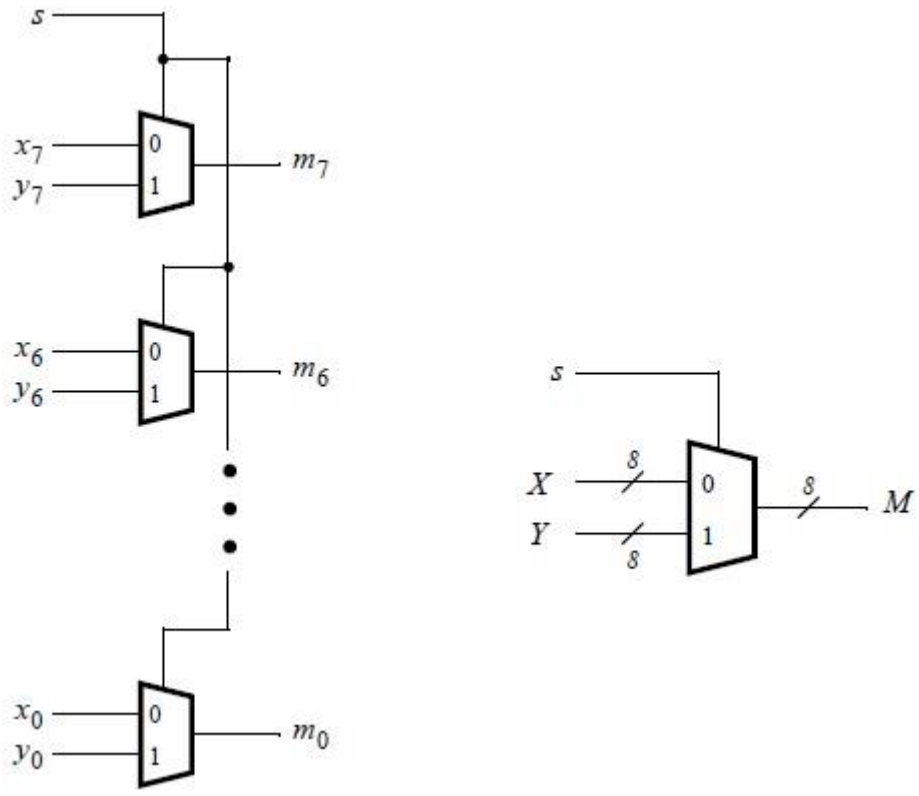


γ) Γραφικό σύμβολο

Σχήμα 6 Πολυπλέκτης 2:1

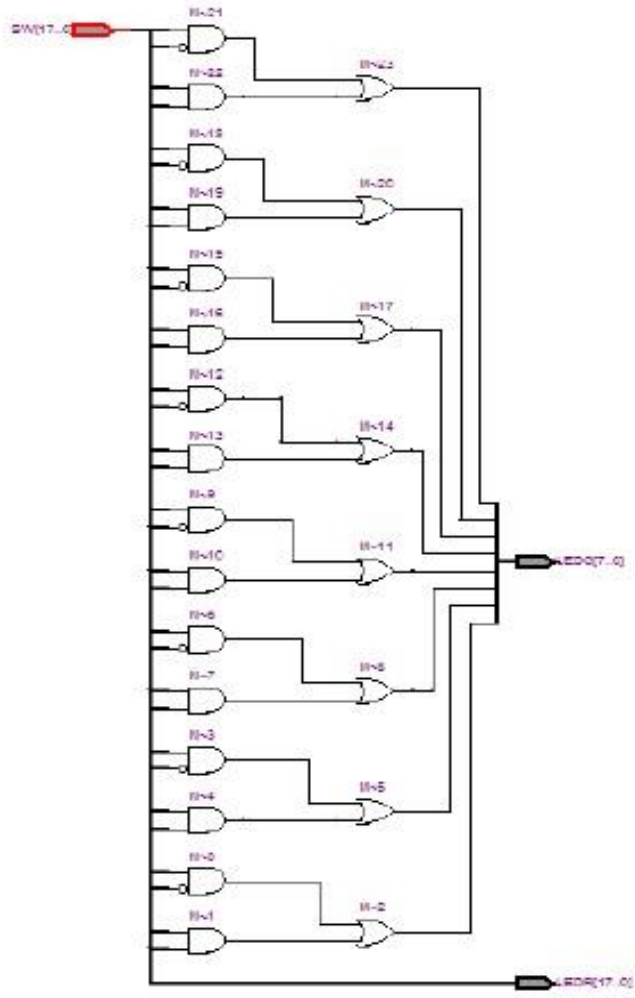
4.3.1 Πολυπλέκτης 2 σε 1 (8 bits)

Ένας πολυπλέκτης 2 σε 1 των 8 bits λειτουργεί σχεδόν με τον ίδιο τρόπο όπως και ένας απλός πολυπλέκτης. Η κυριότερη διαφορά είναι ότι ο συγκεκριμένος πολυπλέκτης αποτελείται από δύο εισόδους των 8 bits, X και Y και παράγει μια έξοδο των 8 bit M. Στο *Σχήμα 7* φαίνεται το λογικό κύκλωμα και το σύμβολο ενός πολυπλέκτη. Εάν s=0 τότε M=X ενώ εάν s=1 τότε M=Y.



Σχήμα 7 Πολυπλέκτης 2:1 των 8 bits

Στην αναπτυξιακή πλακέτα DE2 η είσοδος επιλογής s θα αντιστοιχηθεί με το διακόπτη SW17-0, η είσοδος X με το διακόπτη SW7-0 και η είσοδος Y με το διακόπτη SW15-8. Η έξοδος του κυκλώματος M θα συνδεθεί με το φωτεινό πράσινο διακόπτη LEDG7-0.



Εικόνα 16 Πολυπλέκτης 2:1 των 8 bits

Ο κώδικας VHDL για τον πολυπλέκτη χωρίζεται σε τρία τμήματα. Στη δήλωση της βιβλιοθήκης IEEE.std_logic_1164.all επειδή περιέχει τον τύπο std_logic_vector που χρησιμοποιείται για 8 bits. Στη δήλωση της οντότητας part 2 με εισόδους SW και εξόδους LEDR και LEDG και στη δήλωση της αρχιτεκτονικής με όνομα structure στην οποία υπάρχουν τέσσερα σήματα (sel,x,y,m). Το sel ορίζεται ως ο διακόπτης επιλογής. Για την περιγραφή ενός πολυπλέκτη χρησιμοποιείται η έκφραση:

m <= (NOT (s) AND x) OR (s AND y);

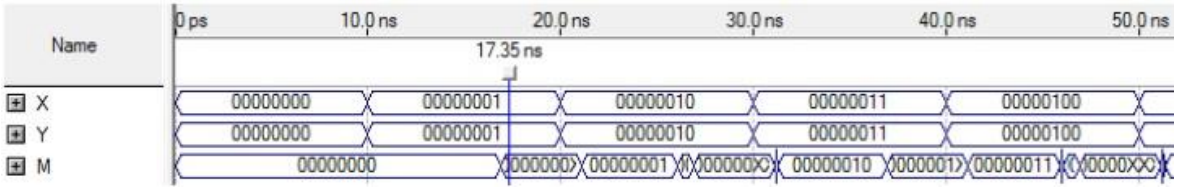
Ο κώδικας VHDL φαίνεται παρακάτω:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part2 IS
PORT (SW: IN STD_LOGIC_VECTOR (17 DOWNTO 0) ;
     LEDR: OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
```

```
      LEDG: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END part2;
ARCHITECTURE Structure OF part2 IS
SIGNAL S: STD_LOGIC;
SIGNAL X, Y, M: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
  LEDR <= SW;
  X <= SW (7 DOWNTO 0);
  Y <= SW (15 DOWNTO 8);
  S <= SW (17);
  M (0) <= (NOT (S) AND X (0)) OR (S AND Y (0));
  M (1) <= (NOT (S) AND X (1)) OR (S AND Y (1));
  M (2) <= (NOT (S) AND X (2)) OR (S AND Y (2));
  M (3) <= (NOT (S) AND X (3)) OR (S AND Y (3));
  M (4) <= (NOT (S) AND X (4)) OR (S AND Y (4));
  M (5) <= (NOT (S) AND X (5)) OR (S AND Y (5));
  M (6) <= (NOT (S) AND X (6)) OR (S AND Y (6));
  M (7) <= (NOT (S) AND X (7)) OR (S AND Y (7));
  LEDG (7 DOWNTO 0) <= M;
END Structure;
```

Πρόγραμμα 2 An 8 bit wide 2-to-1 multiplexer

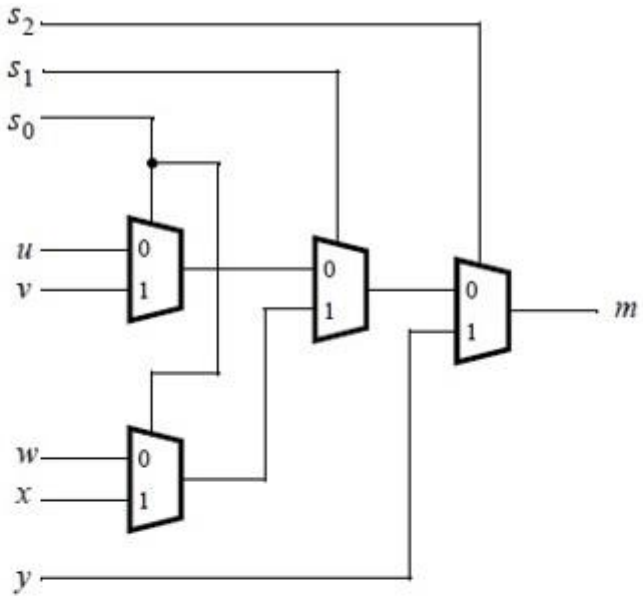
Στο *Σχήμα 8* φαίνεται η προσομοίωση του πολυπλέκτη 2:1 των 8 bits:



Σχήμα 8 Προσομοίωση πολυπλέκτη 2:1

4.3.2 Πολυπλέκτης 5 σε 1 (3 bits)

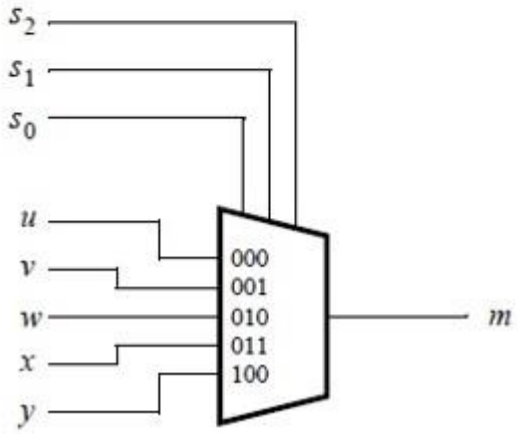
Το *Σχήμα 8* παρουσιάζει πως μπορεί κάποιος να δημιουργήσει έναν πολυπλέκτη 5 σε 1 χρησιμοποιώντας τέσσερις πολυπλέκτες 2 σε 1. Το κύκλωμα χρησιμοποιεί τρεις εισόδους επιλογής $S_2S_1S_0$ οι οποίες είναι των 3 bits και παράγει τον πίνακα αληθείας που φαίνεται παρακάτω.



α) Λογικό Κύκλωμα

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

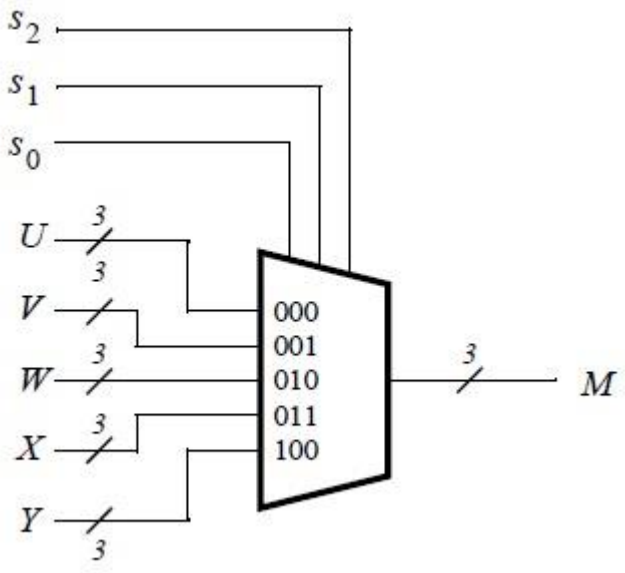
β) Πίνακας αληθείας



γ) Σύμβολο

Σχήμα 9 Πολυπλέκτης 5:1 των 3 bits

Στο *Σχήμα 9* παρουσιάζεται το σύμβολο του πολυπλέκτη 5 σε 1 των 3 bits.



`

Σχήμα 10 A 3 bit wide 5-to-1 multiplexer

Ο κώδικας VHDL φαίνεται παρακάτω:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part3 IS
PORT (SW: IN STD_LOGIC_VECTOR (17 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
LEDG: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END part3;
ARCHITECTURE Structure OF part3 IS
SIGNAL m_0, m_1, m_2: STD_LOGIC_VECTOR (1 TO 3);
SIGNAL S, U, V, W, X, Y, M: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
  S (2 DOWNTO 0) <= SW (17 DOWNTO 15);
  U <= SW (2 DOWNTO 0);
  V <= SW (5 DOWNTO 3);
  W <= SW (8 DOWNTO 6);
  X <= SW (11 DOWNTO 9);
  Y <= SW (14 DOWNTO 12);
  LEDR <= SW;
  m_0 (1) <= (NOT(S (0)) AND U (0)) OR (S (0) AND V (0));
  m_0 (2) <= (NOT(S (0)) AND W (0)) OR (S (0) AND X (0));
  m_0 (3) <= (NOT(S (1)) AND m_0 (1)) OR (S (1) AND m_0 (2));
  M (0) <= (NOT(S (2)) AND m_0 (3)) OR (S (2) AND Y (0));
  m_1 (1) <= (NOT(S (0)) AND U (1)) OR (S (0) AND V (1));
  m_1 (2) <= (NOT(S (0)) AND W (1)) OR (S (0) AND X (1));
  m_1 (3) <= (NOT(S (1)) AND m_1 (1)) OR (S (1) AND m_1 (2));
  M (1) <= (NOT(S (2)) AND m_1 (3)) OR (S (2) AND Y (1));
  m_2 (1) <= (NOT(S (0)) AND U (2)) OR (S (0) AND V (2));
  m_2 (2) <= (NOT(S (0)) AND W (2)) OR (S (0) AND X (2));
  m_2 (3) <= (NOT(S (1)) AND m_2 (1)) OR (S (1) AND m_2 (2));
  M (2) <= (NOT(S (2)) AND m_2 (3)) OR (S (2) AND Y (2));
  LEDG (2 DOWNTO 0) <= M;
END Structure;
```

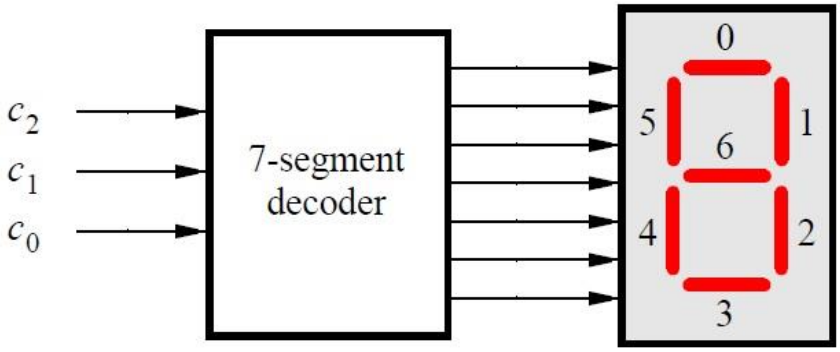
Πρόγραμμα 3 - A 3 bit multiplexer 5-to-1

4.4 Αποκωδικοποιητές και Κωδικοποιητές (Decoders and Encoders)

Ο αποκωδικοποιητής (decoder) είναι συνδυαστικό κύκλωμα που μετατρέπει κωδικοποιημένη δυαδική πληροφορία, η οποία έρχεται σε n γραμμές εισόδου, σε ισοδύναμη πληροφορία που τοποθετείται σε διακριτές γραμμές εξόδου, των οποίων το πλήθος μπορεί να είναι 2^n. Ένας κωδικοποιητής εκτελεί την αντίθετη λειτουργία ενός αποκωδικοποιητή, δηλαδή κωδικοποιεί πληροφορίες σε μία πιο συμπαγή μορφή.

4.4.1 Αποκωδικοποιητής επτά τομέων (7 Segment Decoder)

Το *Σχήμα 11* παρουσιάζει έναν αποκωδικοποιητή επτά τομέων που έχει τις εισόδους c2c1c0 οι οποίες είναι 3 bit. Ο αποκωδικοποιητής παράγει επτά εξόδους οι οποίες χρησιμοποιούνται για να απεικονίσουν έναν χαρακτήρα στον ενδείκτη επτά τομέων.



Σχήμα 11 Αποκωδικοποιητής 7 τομέων

Ο *Πίνακας 6* δείχνει ποιοι χαρακτήρες θα πρέπει να εμφανίζονται για κάθε τιμή των εισόδων c2c1c0. Για να είναι πιο απλή η σχεδίαση, μόνο τέσσερις χαρακτήρες περιλαμβάνονται στον πίνακα (συν τον ‘κενό’ χαρακτήρα, ο οποίος επιλέγεται για τους κωδικούς 100-111).

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

Πίνακας 6 Κωδικοί χαρακτήρων

Ο κώδικας VHDL υλοποιεί λογικές συναρτήσεις οι οποίες αντιπροσωπεύουν κυκλώματα, τα οποία απαιτούνται για την ενεργοποίηση καθενός από τους επτά ενδείκτες. Χρησιμοποιείται μια έκφραση Boolean για να προσδιοριστεί κάθε λογική συνάρτηση.

Το κύκλωμα ενός 7 segment decoder αποτελείται από μία είσοδο SW 3 bits, από μια έξοδο LEDR μεγέθους 3 bits και μια έξοδο HEX0 μεγέθους 7 bits. Κάθε bit της εξόδου αντιστοιχεί σε ένα LED του ενδείκτη επτά τομέων.

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY part4 IS
PORT (SW: IN STD_LOGIC_VECTOR (2 DOWNTO 0) ;
    LEDR: OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
    HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part4;
ARCHITECTURE Structure OF part4 IS
SIGNAL C: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
  LEDR <= SW;
  C (2 DOWNTO 0) <= SW (2 DOWNTO 0);
  HEX0 (0) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
  (NOT(C (2)) AND C (1) AND C (0)));
  HEX0 (1) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
  (NOT(C (2)) AND C (1) AND C (0)));
  HEX0 (2) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
  (NOT(C (2)) AND C (1) AND C (0)));
  HEX0 (3) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
  (NOT(C (2)) AND C (1) AND NOT(C (0))) OR
  (NOT(C (2)) AND C (1) AND C (0)));
  HEX0 (4) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
  (NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
```

```
  (NOT(C (2)) AND C (1) AND NOT(C (0))) OR (NOT(C (2)) AND C (1) AND C
(0)));
  HEX0 (5) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
  (NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
  (NOT(C (2)) AND C (1) AND NOT(C (0))) OR (NOT(C (2)) AND C (1) AND C
(0)));
  HEX0 (6) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
  (NOT(C (2)) AND NOT(C (1)) AND C (0)));
END Structure;
```

Πρόγραμμα 4 -7- Segment

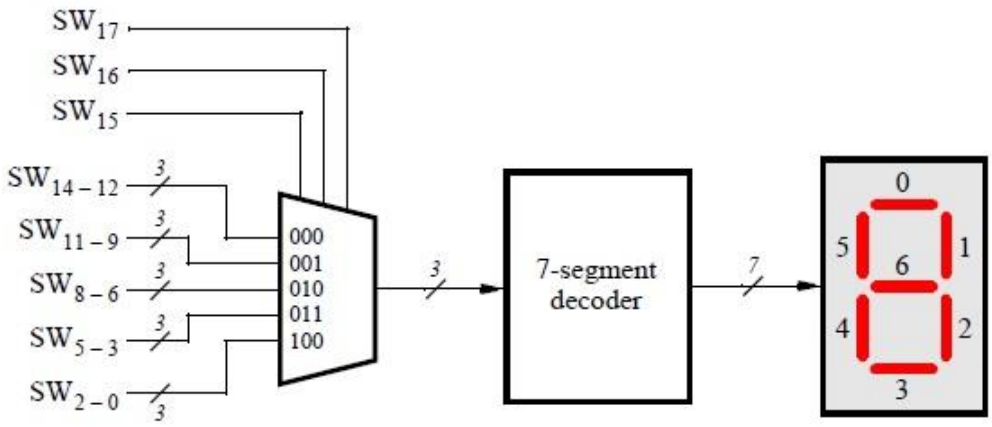
Οι είσοδοι c2c1c0 συνδέονται με τους διακόπτες SW2-0 και η έξοδος στην απεικόνιση επτά τομέων όπως φαίνεται στον *Πίνακα 7:*

Signal Name	FPGA Pin No.
SW[0]	PIN_N25
SW[1]	PIN_N26
SW[2]	PIN_P25
LEDR[0]	PIN_AE23
LEDR[1]	PIN_AF23
LEDR[2]	PIN_AB21
HEX0[0]	PIN_AF10
HEX0[1]	PIN_AB12
HEX0[2]	PIN_AC12
HEX0[3]	PIN_AD11
HEX0[4]	PIN_AE11
HEX0[5]	PIN_V14
HEX0[6]	PIN_V13

Πίνακας 7 Αντιστοίχιση ακροδεκτών

4.4.2 Αποκωδικοποιητής 7 segment

Το κύκλωμα που φαίνεται στο *Σχήμα 12* αποτελείται από έναν πολυπλέκτη 5:1, 3 bits ο οποίος επιλέγει ποιον από τους πέντε χαρακτήρες θα απεικονίσει ως ένδειξη 7 τομέων.



Σχήμα 12 Κύκλωμα επιλογής και ένδειξης 1:5 χαρακτήρων

Δηλαδή, το κύκλωμα εμφανίζει σε πέντε ενδείκτες 7 τομέων διαφορετικές λέξεις των πέντε χαρακτήρων. Οι πολυπλέκτες συνδέονται με τους χαρακτήρες με τέτοιο τρόπο ώστε να επιτρέπεται η περιστροφή μιας λέξης σε ολόκληρο τον ενδείκτη από τα δεξιά προς τα αριστερά όπως φαίνεται στον *Πίνακα 8*.

SW_{17} SW_{16} SW_{15}	Character pattern				
000	H	E	L	L	O
001	E	L	L	O	H
010	L	L	O	H	E
011	L	O	H	E	L
100	O	H	E	L	L

Πίνακας 8 Περιστροφή της λέξης HELLO σε 5 ενδείξεις

Ο κώδικας VHDL είναι ο εξής:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY part5 IS
PORT (SW: IN STD_LOGIC_VECTOR (17 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
HEX7, HEX6, HEX5, HEX4: OUT STD_LOGIC_VECTOR (0 TO 6);
HEX3, HEX2, HEX1, HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part5;
ARCHITECTURE Structure OF part5 IS
COMPONENT mux_3bit_5to1
PORT (S, U, V, W, X, Y: IN STD_LOGIC_VECTOR (2 DOWNTO 0) ;
M: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END COMPONENT;
COMPONENT char_7seg
PORT (C: IN STD_LOGIC_VECTOR (2 DOWNTO 0) ;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END COMPONENT;
```

```
SIGNAL Ch_Sel, Ch1, Ch2, Ch3, Ch4, Ch5, Blank: STD_LOGIC_VECTOR (2 DOWNTO
0);
SIGNAL H4_Ch, H3_Ch, H2_Ch, H1_Ch, H0_Ch: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
LEDR <= SW;
Ch_Sel <= SW (17 DOWNTO 15);
Ch1 <= SW (14 DOWNTO 12);
Ch2 <= SW (11 DOWNTO 9);
Ch3 <= SW (8 DOWNTO 6);
Ch4 <= SW (5 DOWNTO 3);
Ch5 <= SW (2 DOWNTO 0);
Blank <= "111";
M4: mux_3bit_5to1 PORT MAP (Ch_Sel, Ch1, Ch2, Ch3, Ch4, Ch5, H4_Ch);
M3: mux_3bit_5to1 PORT MAP (Ch_Sel, Ch2, Ch3, Ch4, Ch5, Ch1, H3_Ch);
M2: mux_3bit_5to1 PORT MAP (Ch_Sel, Ch3, Ch4, Ch5, Ch1, Ch2, H2_Ch);
M1: mux_3bit_5to1 PORT MAP (Ch_Sel, Ch4, Ch5, Ch1, Ch2, Ch3, H1_Ch);
M0: mux_3bit_5to1 PORT MAP (Ch_Sel, Ch5, Ch1, Ch2, Ch3, Ch4, H0_Ch);
H7: char_7seg PORT MAP (Blank, HEX7);
H6: char_7seg PORT MAP (Blank, HEX6);
H5: char_7seg PORT MAP (Blank, HEX5);
H4: char_7seg PORT MAP (H4_Ch, HEX4);
H3: char_7seg PORT MAP (H3_Ch, HEX3);
H2: char_7seg PORT MAP (H2_Ch, HEX2);
H1: char_7seg PORT MAP (H1_Ch, HEX1);
H0: char_7seg PORT MAP (H0_Ch, HEX0);
END Structure;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY mux_3bit_5to1 IS
PORT (S, U, V, W, X, Y: IN STD_LOGIC_VECTOR (2 DOWNTO 0) ;
M: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
END mux_3bit_5to1;
ARCHITECTURE Behavior OF mux_3bit_5to1 IS
SIGNAL m_0, m_1, m_2: STD_LOGIC_VECTOR (1 TO 3);
BEGIN
m_0 (1) <= (NOT(S (0)) AND U (0)) OR (S (0) AND V (0));
m_0 (2) <= (NOT(S (0)) AND W (0)) OR (S (0) AND X (0));
m_0 (3) <= (NOT(S (1)) AND m_0 (1)) OR (S (1) AND m_0 (2));
M (0) <= (NOT(S (2)) AND m_0 (3)) OR (S (2) AND Y (0));
m_1 (1) <= (NOT(S (0)) AND U (1)) OR (S (0) AND V (1));
m_1 (2) <= (NOT(S (0)) AND W (1)) OR (S (0) AND X (1));
m_1 (3) <= (NOT(S (1)) AND m_1 (1)) OR (S (1) AND m_1 (2));
M (1) <= (NOT(S (2)) AND m_1 (3)) OR (S (2) AND Y (1));
m_2 (1) <= (NOT(S (0)) AND U (2)) OR (S (0) AND V (2));
m_2 (2) <= (NOT(S (0)) AND W (2)) OR (S (0) AND X (2));
m_2 (3) <= (NOT(S (1)) AND m_2 (1)) OR (S (1) AND m_2 (2));
M (2) <= (NOT(S (2)) AND m_2 (3)) OR (S (2) AND Y (2));
END Behavior;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY char_7seg IS
PORT (C: IN STD_LOGIC_VECTOR (2 DOWNTO 0) ;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END char_7seg;
ARCHITECTURE Behavior OF char_7seg IS
BEGIN
form
Display (0) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
(NOT(C (2)) AND C (1) AND C (0)));
Display (1) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
(NOT(C (2)) AND C (1) AND C (0)));
```

```
Display (2) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
(NOT(C (2)) AND C (1) AND C (0)));
Display (3) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
(NOT(C (2)) AND C (1) AND NOT(C (0))) OR
(NOT(C (2)) AND C (1) AND C (0)));
Display (4) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
(NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
(NOT(C (2)) AND C (1) AND NOT(C (0))) OR (NOT(C (2)) AND C (1) AND C (0)));
Display (5) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
(NOT(C (2)) AND NOT(C (1)) AND C (0)) OR
(NOT(C (2)) AND C (1) AND NOT(C (0))) OR (NOT(C (2)) AND C (1) AND C (0)));
Display (6) <= NOT ((NOT(C (2)) AND NOT(C (1)) AND NOT(C (0))) OR
(NOT(C (2)) AND NOT(C (1)) AND C (0)));
END Behavior;
```

Πρόγραμμα 5 -7- segment decoder

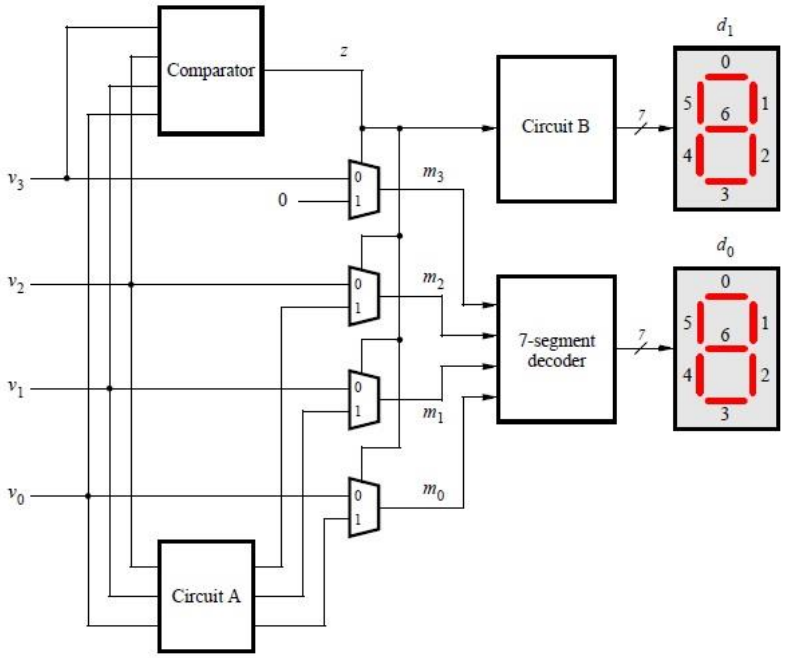
4.4.3 Κωδικοποιητής BCD σε δεκαδικό 7 segment

Ένας κωδικοποιητής BCD (binary coded decimal) μετατρέπει έναν αριθμό ο οποίος είναι γραμμένος σε δυαδικό κώδικα BCD στον αντίστοιχο δεκαδικό αριθμό για απεικόνιση σε ενδείκτη επτά τομέων. Για το λόγο αυτό, ο κωδικοποιητής αυτός είναι γνωστός και ως μετατροπέας. Αυτά τα επτά σήματα χρησιμοποιούνται για να οδηγήσουν τα LEDs του ενδείκτη. Οι δίοδοι αυτοί συμβολίζονται με τους αριθμούς 0 ως 6.

Binary value	Decimal digits	
0000	0	0
0001	0	1
0010	0	2
...	...	...
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Πίνακας 9 Τιμές μετατροπής δυαδικού σε δεκαδικό

Το κύκλωμα αποτελείται από έναν συγκριτή ο οποίος συγκρίνει δύο αριθμούς, τους οποίους τους δέχεται ως είσοδο. Ο σχεδιασμός του κυκλώματος παρουσιάζεται στο *Σχήμα 13*.



Σχήμα 13 Μετατροπέας δυαδικού σε δεκαδικό

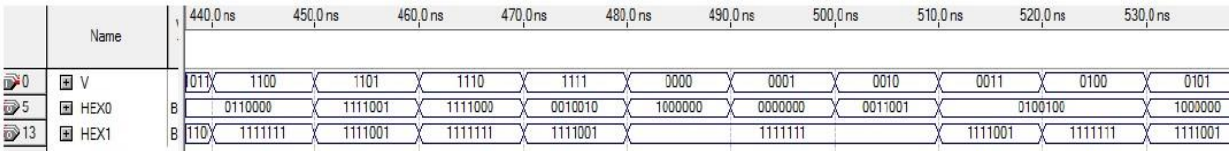
Ο κώδικας αποτελείται από πολυπλέκτες, το κύκλωμα A, το κύκλωμα B καθώς και αποκωδικοποιητή 7 τομέων. Διαθέτει μια είσοδο V (0-4 bits), μια έξοδο U (0-4 bits) και μια έξοδο Z. Στον κώδικα υπάρχουν λογικές εκφράσεις χωρίς καμία δήλωση IF-ELSE, CASE. Ο ρόλος του συγκριτή είναι να ελέγχει αν η τιμή της εισόδου V είναι μεγαλύτερη του 9 κι αν χρησιμοποιεί την έξοδο του συγκριτή για τον έλεγχο του ενδείκτη 7 τομέων. Αν η τιμή είναι μεγαλύτερη από 9 τότε η έξοδος παίρνει την τιμή 1, ενώ αν η τιμή είναι μεγαλύτερη τότε η έξοδος παίρνει το κενό. Για να απεικονιστούν οι τιμές των δεκαδικών ψηφίων οι διακόπτες SW3-0 συνδέονται στην είσοδο V και οι ενδείκτες HEX1 και HEX0 συνδέονται στον 7 segment.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part6 IS
PORT (SW: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
HEX1, HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part6;
ARCHITECTURE Structure OF part6 IS
COMPONENT bcd7seg
PORT (B: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
H: OUT STD_LOGIC_VECTOR (0 TO 6));
END COMPONENT;
SIGNAL V, M: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL B: STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL z: STD_LOGIC;
```

```
BEGIN
V <= SW;
z <= (V (3) AND V (2)) OR (V (3) AND V (1));
B (2) <= V (2) AND V (1);
B (1) <= V (2) AND NOT (V (1));
B (0) <= (V (1) AND V (0)) OR (V (2) AND V (0));
M (3) <= NOT (z) AND V (3);
M (2) <= (NOT (z) AND V (2)) OR (z AND B (2));
M (1) <= (NOT (z) AND V (1)) OR (z AND B (1));
M (0) <= (NOT (z) AND V (0)) OR (z AND B (0));
Circuit_D: bcd7seg PORT MAP (M, HEX0);
HEX1 <= ('1' & NOT (z) & NOT (z) & "1111");
END Structure;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY bcd7seg IS
PORT (B: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
H: OUT STD_LOGIC_VECTOR (0 TO 6));
END bcd7seg;
ARCHITECTURE Structure OF bcd7seg IS
BEGIN
H (0) <= (B (2) AND NOT (B (0))) OR
(NOT (B (3)) AND NOT (B (2)) AND NOT (B (1)) AND B (0));
H (1) <= (B (2) AND NOT (B (1)) AND B (0)) OR
(B (2) AND B (1) AND NOT (B (0)));
H (2) <= (NOT (B (2)) AND B (1) AND NOT (B (0)));
H (3) <= (NOT (B (2)) AND NOT (B (1)) AND B (0)) OR
(B (2) AND NOT (B (1)) AND NOT (B (0))) OR (B (2) AND B (1) AND B (0));
H (4) <= (NOT (B (1)) AND B (0)) OR (NOT (B (3)) AND B (0)) OR
(NOT (B (3)) AND B (2) AND NOT (B (1)));
H (5) <= (B (1) AND B (0)) OR (NOT (B (2)) AND B (1)) OR
(NOT (B (3)) AND NOT (B (2)) AND B (0));
H (6) <= (B (2) AND B (1) AND B (0)) OR (NOT (B (3)) AND NOT (B (2)) AND
NOT (B (1)));
END Structure;
```

Πρόγραμμα 6 BCD to decimal converter

Στο *Σχήμα 14*, παρουσιάζεται η σωστή λειτουργία του κωδικοποιητή, αφού οι τιμές στις εξόδους ανταποκρίνονται στις τιμές του πίνακα αληθείας.



Σχήμα 14 Προσομοίωση κωδικοποιητή BCD

4.5 Αθροιστές

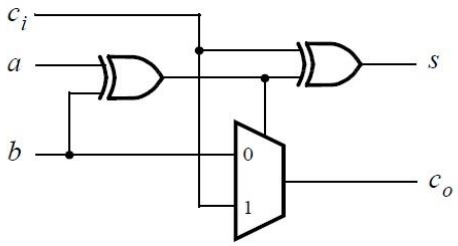
Ο αθροιστής είναι ένα ψηφιακό κύκλωμα με εισόδους δύο ή περισσότερα σήματα και ως έξοδος ορίζεται το άθροισμα των σημάτων αυτών. Οι αθροιστές χωρίζονται σε δύο

κατηγορίες σε αναλογικούς και σε ψηφιακούς. Οι αναλογικοί αθροιστές βασίζονται στους τελεστικούς ενισχυτές ενώ από την άλλη οι ψηφιακοί αθροιστές βασίζονται σε ψηφιακά λογικά κυκλώματα. Οι αθροιστές είναι απαραίτητοι μέσα στους μικροεπεξεργαστές ως μέρος της αριθμητικής και λογικής μονάδα (ALU) καθώς εκεί γίνεται η επεξεργασία και ο χειρισμός των δυαδικών αριθμών. Επίσης, είναι χρήσιμοι και σε άλλα μέρη ενός επεξεργαστή όπως για παράδειγμα για τον υπολογισμό διευθύνσεων, πίνακα δεικτών. Βασικά ψηφιακά κυκλώματα των αθροιστών αποτελούν ο ημιαθροιστής (half adder) και ο πλήρης αθροιστής (full adder). Το συνδυαστικό κύκλωμα που εκτελεί την πρόσθεση δύο bit ονομάζεται ημιαθροιστής (half adder). Το κύκλωμα του περιλαμβάνει δύο δυαδικές εισόδους και δύο δυαδικές εξόδους. Οι μεταβλητές εισόδου παίρνουν τις τιμές των bit των προσθετέων ενώ στις μεταβλητές εξόδου πρέπει να ανατίθενται από το κύκλωμα του ημιαθροιστή οι τιμές του αθροίσματος και του κρατούμενου. Ο πλήρης αθροιστής είναι ένα συνδυαστικό κύκλωμα που παράγει το αριθμητικό άθροισμα τριών bit. Το κύκλωμα αυτό αποτελείται από τρεις εισόδους και δύο εξόδους. Δύο από τις μεταβλητές εισόδου, παριστάνουν τα δύο bit της ίδιας τάξης που θα προστεθούν ενώ η τρίτη είσοδος παριστάνει το κρατούμενο από την προηγούμενη πρόσθεση. Η μία από τις δύο μεταβλητές εξόδου δίνει την τιμή του bit του αθροίσματος ενώ η άλλη δίνει το κρατούμενο εξόδου.

4.5.1 Πλήρης Αθροιστής

Ο πλήρης αθροιστής μπορεί να υλοποιηθεί με δύο ημιαθροιστές, από τους οποίους ο πρώτος προσθέτει τα ci και a και ο δεύτερος προσθέτει στο πρώτο άθροισμα το κρατούμενο εισόδου. Το τελικό κρατούμενο προκύπτει με μια πύλη OR, που εξάγει μονάδα αν οποιαδήποτε από τις δύο βαθμίδες πρόσθεσης παράγει κρατούμενο 1.

Στο *Σχήμα 15* απεικονίζονται το λογικό διάγραμμα, το σύμβολο και ο πίνακας αληθείας του πλήρη αθροιστή. Το κύκλωμα έχει τρεις εισόδους (ci, a, b), όπου ci είναι το κρατούμενο της προηγούμενης πρόσθεσης, και δύο εξόδους (s, co) όπου s είναι το άθροισμα και co το κρατούμενο από την εκτέλεση της πρόσθεσης.



α) Λογικό κύκλωμα

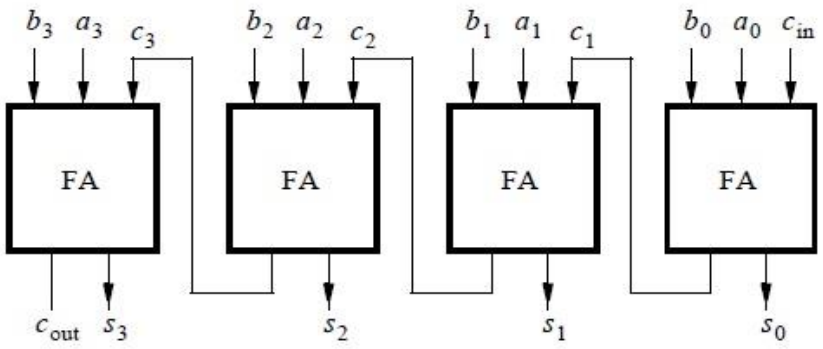
β) Σύμβολο

b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

γ) Πίνακας αληθείας

Σχήμα 15 Πλήρης αθροιστής

Ο πλήρης αθροιστής που περιγράφεται παρακάτω, έχει εύρος εισόδων και εξόδων 4 bits και αποτελείται από τέσσερις πλήρης αθροιστές. Ως είσοδο δέχεται δύο αριθμούς 4 bits και ένα κρατούμενο του 1 bit. Το άθροισμα των αριθμών που προκύπτει στην έξοδο είναι επίσης 4 bits ενώ το κρατούμενο που παράγεται από την πρόσθεση είναι 1 bit. Ο συγκεκριμένος τύπος κυκλώματος ονομάζεται Ripple-Carry-Adder από τον τρόπο με τον οποίο διαδίδεται στην επόμενη βαθμίδα πρόσθεσης το κρατούμενο.



Σχήμα 16 Four bit ripple carry adder

Στον κώδικα VHDL η οντότητα part 3 αποτελείται από σήμα εισόδου SW (7-0), από σήματα εξόδου Ledr (7-0) και Ledg (7-0). Το τμήμα της αρχιτεκτονικής αποτελείται από ένα υποκύκλωμα με την ονομασία fa μέσα στο οποίο υπάρχουν τρεις είσοδοι (a,b,ci) και δύο έξοδοι (s,co). Η είσοδος ci λειτουργεί ως κρατούμενο εισόδου. Η έξοδος s δίνει το άθροισμα των εισόδων a και b ενώ η έξοδος co είναι το κρατούμενο της εξόδου. Επίσης, στο τμήμα της αρχιτεκτονικής δηλώνονται τρία σήματα 3 bits που θα χρησιμοποιηθεί για την μεταφορά του κρατουμένου μεταξύ των αθροιστών. Στο κύριο σώμα της αρχιτεκτονικής γίνεται περιγραφή τεσσάρων στιγμιότυπων με ονόματα bit0, bit1, bit2 και bit3. Μετά το όνομα του στιγμιότυπου ακολουθεί το όνομα του υποκυκλώματος. Τα σήματα του υποκυκλώματος αντιστοιχίζονται με τα σήματα που έχουν οριστεί στο κυρίως κύκλωμα.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part7 IS
PORT (SW: IN STD_LOGIC_VECTOR (7 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
LEDG: OUT STD_LOGIC_VECTOR (4 DOWNTO 0));
END part7;
ARCHITECTURE Structure OF part7 IS
COMPONENT fa
PORT (a, b, ci: IN STD_LOGIC;
s, co: OUT STD_LOGIC);
END COMPONENT;
SIGNAL A, B, S: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL C: STD_LOGIC_VECTOR (4 DOWNTO 1);
BEGIN
A <= SW (7 DOWNTO 4);
B <= SW (3 DOWNTO 0);
bit0: fa PORT MAP (A (0), B (0), '0', S (0), C (1));
bit1: fa PORT MAP (A (1), B (1), C (1), S (1), C (2));
```

```
bit2: fa PORT MAP (A (2), B (2), C (2), S (2), C (3));
bit3: fa PORT MAP (A (3), B (3), C (3), S (3), C (4));
LEDR <= SW;
LEDG <= (C (4) & S);
END Structure;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY fa IS
PORT (a, b, ci: IN STD_LOGIC;
s, co: OUT STD_LOGIC);
END fa;
ARCHITECTURE Structure OF fa IS
SIGNAL a_xor_b: STD_LOGIC;
BEGIN
a_xor_b <= a XOR b;
s <= a_xor_b XOR ci;
co <= (NOT (a_xor_b) AND b) OR (a_xor_b AND ci);
END Structure;
```

Πρόγραμμα 7 Four bit ripple carry adder

Για την υλοποίηση του κώδικα VHDL θα πρέπει να γίνει η παρακάτω αντιστοίχιση ώστε οι είσοδοι και οι έξοδοι του κυρίως κυκλώματος που ορίστηκαν να αντιστοιχηθούν με ακροδέκτες της διάταξης FPGA (Πίνακας 15). Οι δύο αριθμοί των εισόδων a, b παίρνουν τιμή με τη βοήθεια των διακοπτών SW του αναπτυξιακού.

Signal Name	FPGA Pin No.
SW[0]	PIN_N25
SW[1]	PIN_N26
SW[2]	PIN_P25
SW[3]	PIN_AE14
SW[4]	PIN_AF14
SW[5]	PIN_AD13
SW[6]	PIN_AC13
SW[7]	PIN_C13

Signal Name	FPGA Pin No.
LEDR[0]	PIN_AE23
LEDR[1]	PIN_AF23
LEDR[2]	PIN_AB21
LEDR[3]	PIN_AC22
LEDR[4]	PIN_AD22
LEDR[5]	PIN_AD23
LEDR[6]	PIN_AD21
LEDR[7]	PIN_AC21
LEDG[0]	PIN_AA14
LEDG[1]	PIN_Y13
LEDG[2]	PIN_AA13
LEDG[3]	PIN_AC14
LEDG[4]	PIN_AD15
LEDG[5]	PIN_AE15

Πίνακας 10 Αντιστοίχιση ακροδεκτών

4.5.2 Αθροιστής Ψηφίων BCD

Το κύκλωμα ενός BCD αθροιστή προσθέτει δύο δεκαδικά ψηφία σε κώδικα BCD και ένα πιθανό κρατούμενο από κάποια προηγούμενη βαθμίδα, το άθροισμα των οποίων στην έξοδο δεν μπορεί να ξεπερνά τον αριθμό 19. Αυτό δημιουργείται επειδή κάθε ψηφίο εισόδου δεν πρέπει να ξεπερνά το 9(1001) με αποτέλεσμα αν τροφοδοτηθεί ένας δυαδικός αθροιστής 4 bits με δύο ψηφία BCD θα σχηματίσει το άθροισμα δυαδικά και το αποτέλεσμα που θα παράγει θα κυμαίνεται από 0 μέχρι 9. Για την υλοποίηση ενός τέτοιου αθροιστή χρησιμοποιούνται υποκυκλώματα, ενός αθροιστή 4 bits(fa), ενός μετατροπέα δυαδικού σε δεκαδικό (BCD_decimal) και ενός μετατροπέα από BCD σε ενδείκτη επτά τομέων(BCD 7 segment).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part8 IS
PORT (SW: IN STD_LOGIC_VECTOR (8 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (8 DOWNTO 0);
LEDG: OUT STD_LOGIC_VECTOR (8 DOWNTO 0);
HEX7, HEX6, HEX5, HEX4: OUT STD_LOGIC_VECTOR (0 TO 6);
HEX3, HEX2, HEX1, HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part8;
ARCHITECTURE Structure OF part8 IS
COMPONENT fa
PORT (a, b, ci: IN STD_LOGIC;
s, co: OUT STD_LOGIC);
END COMPONENT;
COMPONENT bcd_decimal
PORT (V: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
z: BUFFER STD_LOGIC;
M: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END COMPONENT;
COMPONENT bcd7seg
PORT (B: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
H: OUT STD_LOGIC_VECTOR (0 TO 6));
END COMPONENT;
SIGNAL A, B, S: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL Cin: STD_LOGIC;
SIGNAL C: STD_LOGIC_VECTOR (4 DOWNTO 1);
SIGNAL S0: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL S0_M: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL S1: STD_LOGIC;
BEGIN
A <= SW (7 DOWNTO 4);
B <= SW (3 DOWNTO 0);
Cin <= SW (8);
bit0: fa PORT MAP (A (0), B (0), Cin, S (0), C (1));
bit1: fa PORT MAP (A (1), B (1), C (1), S (1), C (2));
bit2: fa PORT MAP (A (2), B (2), C (2), S (2), C (3));
bit3: fa PORT MAP (A (3), B (3), C (3), S (3), C (4));
LEDR <= SW;
LEDG (4 DOWNTO 0) <= (C (4) & S);
H_6: bcd7seg PORT MAP (A, HEX6);
```

```
HEX7 <= ("1111111");
H_4: bcd7seg PORT MAP (B, HEX4);
HEX5 <= "1111111";
LEDG (8) <= (A (3) AND A (2)) OR (A (3) AND A (1)) OR
(B (3) AND B (2)) OR (B (3) AND B (1));
LEDG (7 DOWNTO 5) <= "000";
BCD_S: bcd_decimal PORT MAP (S, S1, S0);
S0_M (3) <= (NOT(C (4)) AND S0 (3)) OR (C (4) AND S0 (1));
S0_M (2) <= (NOT(C (4)) AND S0 (2)) OR (C (4) AND NOT (S0 (1)));
S0_M (1) <= (NOT(C (4)) AND S0 (1)) OR (C (4) AND NOT (S0 (1)));
S0_M (0) <= S0 (0);
H_0: bcd7seg PORT MAP (S0_M, HEX0);
HEX1 <= ('1' & NOT (S1 OR C (4)) & NOT (S1 OR C (4)) & "1111");
HEX2 <= "1111111";
HEX3 <= "1111111";
END Structure;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY fa IS
PORT (a, b, ci: IN STD_LOGIC;
s, co: OUT STD_LOGIC);
END fa;
ARCHITECTURE Structure OF fa IS
SIGNAL a_xor_b: STD_LOGIC;
BEGIN
a_xor_b <= a XOR b;
s <= a_xor_b XOR ci;
co <= (NOT (a_xor_b) AND b) OR (a_xor_b AND ci);
END Structure;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY bcd_decimal IS
PORT (V: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
z: BUFFER STD_LOGIC;
M: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END bcd_decimal;
ARCHITECTURE Structure OF bcd_decimal IS
SIGNAL B: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
z <= (V (3) AND V (2)) OR (V (3) AND V (1));
B (2) <= V (2) AND V (1);
B (1) <= V (2) AND NOT (V (1));
B (0) <= (V (1) AND V (0)) OR (V (2) AND V (0));
M (3) <= NOT (z) AND V (3);
M (2) <= (NOT (z) AND V (2)) OR (z AND B (2));
M (1) <= (NOT (z) AND V (1)) OR (z AND B (1));
M (0) <= (NOT (z) AND V (0)) OR (z AND B (0));
END Structure;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY bcd7seg IS
PORT (B: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
H: OUT STD_LOGIC_VECTOR (0 TO 6));
END bcd7seg;
ARCHITECTURE Behavior OF bcd7seg IS
BEGIN
H (0) <= (B (2) AND NOT (B (0))) OR
(NOT (B (3)) AND NOT (B (2)) AND NOT (B (1)) AND B (0));
H (1) <= (B (2) AND NOT (B (1)) AND B (0)) OR
(B (2) AND B (1) AND NOT (B (0)));
H (2) <= (NOT (B (2)) AND B (1) AND NOT (B (0)));
```

```
H (3) <= (NOT (B (2)) AND NOT (B (1)) AND B (0)) OR
(B (2) AND NOT (B (1)) AND NOT (B (0))) OR (B (2) AND B (1) AND B (0));
H (4) <= (NOT (B (1)) AND B (0)) OR (NOT (B (3)) AND B (0)) OR
(NOT (B (3)) AND B (2) AND NOT (B (1)));
H (5) <= (B (1) AND B (0)) OR (NOT (B (2)) AND B (1)) OR
(NOT (B (3)) AND NOT (B (2)) AND B (0));
H(6) <= (B(2) AND B(1) AND B(0)) OR (NOT(B(3)) AND NOT(B(2)) AND NOT(B(1)));
END Behavior;
```

Πρόγραμμα 8 Αθροιστής ψηφίων BCD

4.6 Ακολουθιακά Κυκλώματα

Υπάρχει μια κατηγορία λογικών κυκλωμάτων, στα οποία οι τιμές των εξόδων δεν εξαρτώνται μόνο από τις τρέχουσες τιμές των εισόδων, αλλά και από την προηγούμενη συμπεριφορά του κυκλώματος. Τέτοια κυκλώματα είναι τα στοιχεία αποθήκευσης που μπορούν να αποθηκεύσουν τις τιμές των λογικών σημάτων. Όταν αλλάξουν οι τιμές των εισόδων του κυκλώματος, οι νέες τιμές μπορεί να διατηρήσουν το κύκλωμα στην υπάρχουσα κατάσταση ή να το οδηγήσουν σε μια καινούρια. Καθώς ο χρόνος κυλά ένα κύκλωμα περνά από ένα σύνολο καταστάσεων, που είναι αποτέλεσμα των αλλαγών στις εισόδους του. Τα κυκλώματα που έχουν αυτή τη συμπεριφορά ονομάζονται *ακολουθιακά κυκλώματα (sequential circuits).*

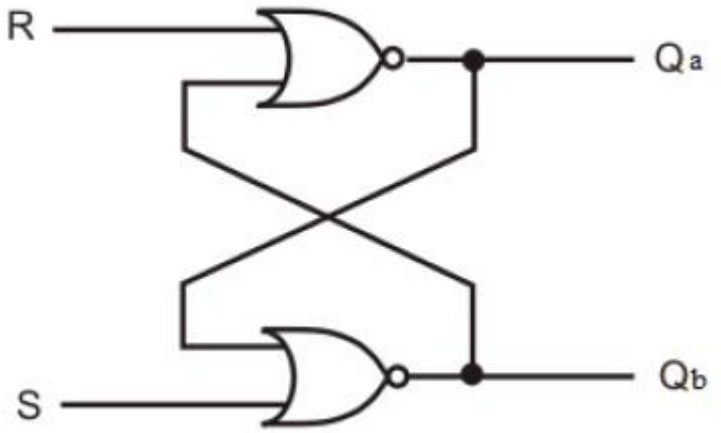
Τα ακουλουθιακά κυκλώματα χωρίζονται σε δύο κατηγορίες, τα σύγχρονα και τα ασύγχρονα. Σύγχρονο ονομάζεται ένα ακολουθιακό κύκλωμα, στην περίπτωση που υπάρχει ένα ωρολογιακό σήμα με σκοπό να ελέγχει τη λειτουργία του ακολουθιακού κυκλώματος. Ενώ, ασύγχρονο ονομάζεται ένα ακολουθιακό κύκλωμα στην περίπτωση που δεν χρησιμοποιείται ωρολογιακό σήμα.

Τα σύγχρονα ακολουθιακά κυκλώματα υλοποιούνται με τη βοήθεια συνδυαστικής λογικής και ενός ή περισσότερων flip-flops. Ένα flip-flop μπορεί να χρησιμοποιηθεί ως κύτταρο μνήμης καθώς μπορεί να διατηρηθεί σε μια κατάσταση έως ότου κάποιο κατάλληλο σήμα εισόδου το κάνει να αλλάξει κατάσταση (αποθήκευση 1 bit πληροφορίας).

Τα ασύγχρονα ακολουθιακά κυκλώματα αποτελούνται από λογικές πύλες που προκαλούν καθυστέρηση διάδοσης στα σήματα που διαδίδονται μέσα από αυτές και ονομάζονται μανδαλωτές (latches).

4.6.1 Μανδαλωτής RS

Το κύκλωμα του μανδαλωτή RS πραγματοποιείται με δύο πύλες NOR όπως φαίνεται παρακάτω:



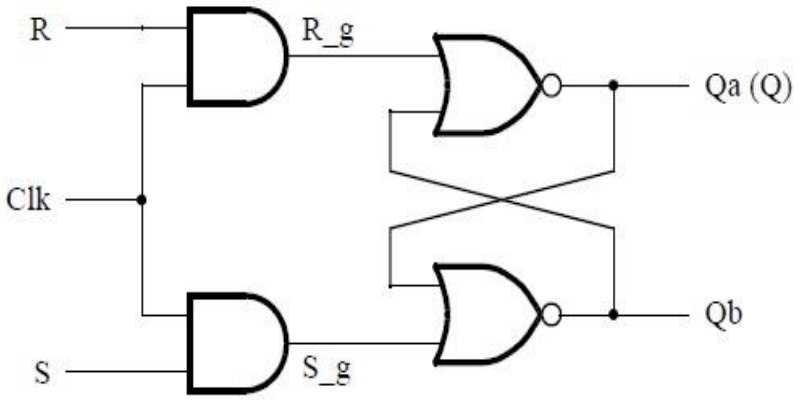
Σχήμα 17 Μανδαλωτής RS

Ο μανδαλωτής έχει δύο εισόδους, που χαρακτηρίζονται ως R (reset) και S (set), καθώς και δύο εξόδους που καθορίζουν την κατάσταση του κυκλώματος. Η κατάσταση του μανδαλωτή Qa=1 και Qb=0 χαρακτηρίζεται ως κατάσταση set ενώ η κατάσταση Qa=0 και Qb=1 χαρακτηρίζεται ως κατάσταση reset. Οι έξοδοι αποτελούν συμπλήρωμα η μία της άλλης. Όταν και οι δύο είσοδοι πάρουν την τιμή 1 ταυτόχρονα, τότε προκύπτει μία μη επιτρεπτή κατάσταση. Η λειτουργία του μανδαλωτή παρουσιάζεται στον Πίνακα 11.

Λειτουργία του μανδαλωτή SR				
Είσοδοι		Έξοδοι		Κατάσταση
S	R	Qa	Qb	
1	0	1	0	Set
0	0	1	0	Store
0	1	0	1	Reset
0	0	0	1	Store
1	1	0	0	Μη επιτρεπτή

Πίνακας 11 Η λειτουργία του RS latch

Αν οι είσοδοι R και S οδηγηθούν από δυο πύλες AND, όπου η μια είσοδος συνδέεται σε κοινή πηγή παλμών ρολογιού τότε έχουμε έναν χρονιζόμενο μανδαλωτή. Με την είσοδο clock καθορίζεται πότε θα αλλαχθεί η κατάσταση του χρονιζόμενου μανδαλωτή. Έτσι, όταν το clock=0 τότε οι είσοδοι R_g και S_g θα είναι μηδέν. Ενώ όταν το clock=1 τότε οι είσοδοι R_g και S_g θα είναι ίδιες με τις εισόδους R και S. *(Σχήμα 18)*



Σχήμα 18 Χρονιζόμενος μανδαλωτής RS

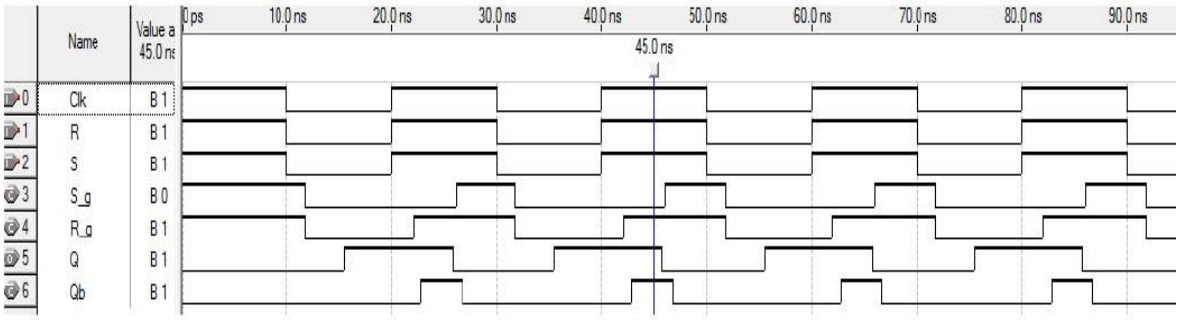
Ο κώδικας VHDL που περιγράφει αυτό το κύκλωμα είναι ο εξής:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY part9 IS
PORT (Clk, R, S: IN STD_LOGIC;
Q: OUT STD_LOGIC);
END part9;
ARCHITECTURE Structural OF part9 IS
```

```
SIGNAL R_g, S_g, Qa, Qb: STD_LOGIC;
ATTRIBUTE keep: Boolean;
ATTRIBUTE keep of R_g, S_g, Qa, Qb: signal is true;
BEGIN
R_g <= R AND Clk;
S_g <= S AND Clk;
Qa <= NOT (R_g OR Qb);
Qb <= NOT (S_g OR Qa);
Q <= Qa;
END Structural;
```

Πρόγραμμα 9 VHDL code RS latch

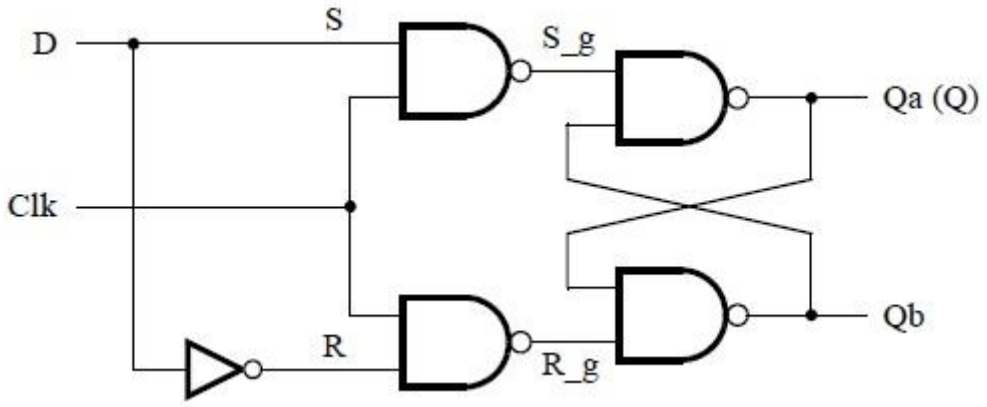
Με την προσομοίωση επιβεβαιώνεται η σωστή λειτουργία του μανδαλωτή:



Σχήμα 19 Προσομοίωση RS latch

4.6.2 Μανδαλωτής τύπου D

Ένας τρόπος να αντιμετωπιστεί το πρόβλημα της ανεπιθύμητης συμπεριφοράς λόγω της μη επιτρεπτής κατάστασης στον μανδαλωτή SR είναι να διασφαλιστεί ότι οι είσοδοι S και R δεν θα παίρνουν την τιμή 1 ταυτόχρονα. Αυτό επιτυγχάνεται με χρήση του D latch.



Σχήμα 20 D latch

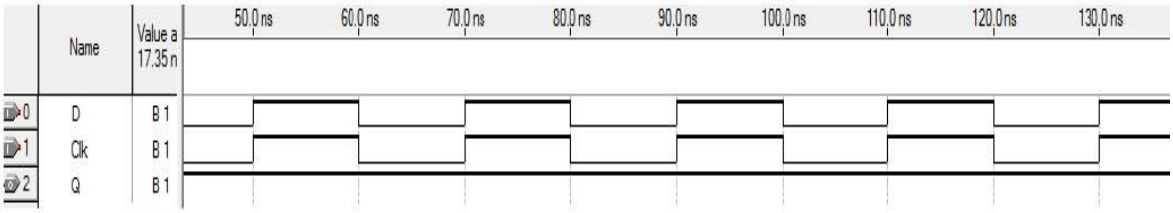
Το κύκλωμα έχει μόνο δύο εισόδους: την D (data) και την Clc (clock). Η τιμή της εισόδου D γίνεται αντιληπτή όταν Clk = 1. Όταν D = 1, η έξοδος Q παίρνει την τιμή 1, οπότε το κύκλωμα βρίσκεται σε κατάσταση set. Όταν D = 0, η έξοδος Q παίρνει τιμή 0, οπότε το κύκλωμα μεταβαίνει σε κατάσταση reset.

Παρακάτω απεικονίζεται ο κώδικας VHDL:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY D_latch IS
PORT (Clk, D: IN STD_LOGIC;
Q: OUT STD_LOGIC);
END D_latch;
ARCHITECTURE Behavior OF D_latch IS
SIGNAL R, R_g, S_g, Qa, Qb: STD_LOGIC;
ATTRIBUTE keep: Boolean;
ATTRIBUTE keep of R, R_g, S_g, Qa, Qb: signal is true;
BEGIN
R <= NOT D;
S_g <= NOT (D AND Clk);
R_g <= NOT (R AND Clk);
Qa <= NOT (S_g AND Qb);
Qb <= NOT (R_g AND Qa);
Q <= Qa;
END Behavior;
```

Πρόγραμμα 10 VHDL code RS latch

Στο *Σχήμα 21* φαίνεται η προσομοίωση του D latch:

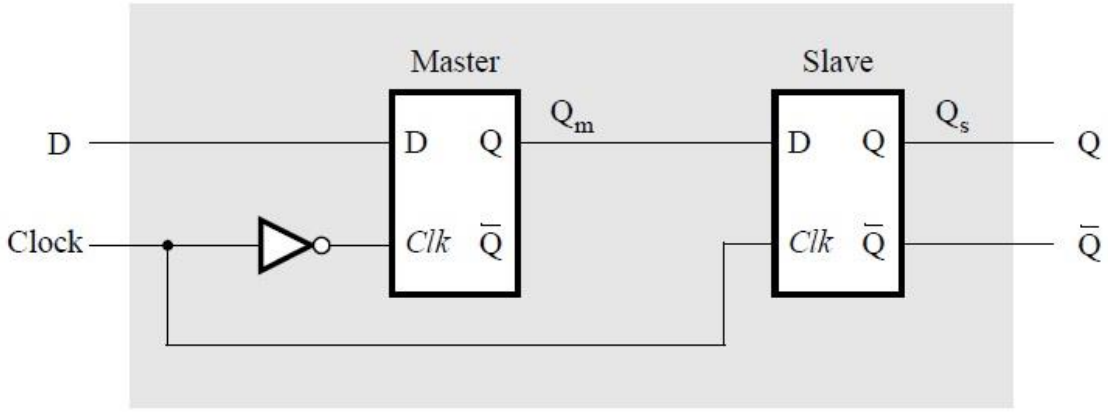


Σχήμα 21 Προσομοίωση D latch

4.6.3 D flip-flop

Κυκλώματα τα οποία αποτελούνται από μία είσοδο δεδομένων D, μία είσοδο ρολογιού Clk και δύο εξόδους Q και QN ονομάζονται Flip flop. Οι μανδαλωτές που περιγράφηκαν παραπάνω όπως και τα Flip flop αποτελούν τα πιο μικρά στοιχεία αποθήκευσης με χωρητικότητα 1 bit. Υπάρχουν δύο κατηγορίες Flip flop. Η πρώτη κατηγορία είναι αυτή που

η έξοδος τους επηρεάζεται από την είσοδο κατά το θετικό μέτωπο του παλμού του ρολογιού, δηλαδή η ανανέωση της εξόδου γίνεται την στιγμή της μετάβασης του ρολογιού από την κατάσταση Low στην κατάσταση High. Η δεύτερη κατηγορία FF είναι αυτή που η έξοδός τους επηρεάζεται από την είσοδο κατά την αρνητική ακμή του ρολογιού, δηλαδή, κατά την μετάβαση του ρολογιού από την High κατάσταση στην Low. Ένα τέτοιο κύκλωμα παρουσιάζεται παρακάτω και ονομάζεται D Flip flop master slave.



Σχήμα 22 Κύκλωμα D flip flop master slave

Το παραπάνω κύκλωμα αποτελείται από δύο μανδαλωτές D. Ο πρώτος μανδαλωτής, ο οποίος ονομάζεται master αλλάζει κατάσταση όταν Clock=1 ενώ ο δεύτερος μανδαλωτής ονομάζεται slave και αλλάζει κατάσταση όταν Clock=0. Το κύκλωμα λειτουργεί με τέτοιο τρόπο ώστε όταν ο ωρολογιακός παλμός έχει υψηλή τιμή, ο μανδαλωτής master να παρακολουθεί την τιμή του σήματος της εισόδου D αλλά ο μανδαλωτής slave να μην αλλάζει κατάσταση. Άρα, η τιμή της εξόδου Qm παρακολουθεί τις αλλαγές της εισόδου, ενώ η έξοδος Qs παραμένει σταθερή. Ο μανδαλωτής master θα σταματήσει να παρακολουθεί τις αλλαγές της εισόδου D όταν το ωρολογιακό σήμα αλλάξει στην τιμή 0. Ταυτόχρονα, ο μανδαλωτής slave αποκρίνεται στην τιμή Qm και μεταβάλλει την έξοδό του ανάλογα. Ο μανδαλωτής slave μπορεί να υποστεί το πολύ μία αλλαγή κατά τη διάρκεια ενός ωρολογιακού παλμού εφόσον η τιμή Qm δεν αλλάζει όσο το Clock=0.

Ο κώδικας VHDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part10 IS
PORT (SW: IN STD_LOGIC_VECTOR (1 DOWNTO 0) ;
LEDR: OUT STD_LOGIC_VECTOR (0 TO 0));
```

```
END part10;
ARCHITECTURE Structural OF part10 IS
COMPONENT D_latch
PORT (Clk, D: IN STD_LOGIC;
Q: OUT STD_LOGIC);
END COMPONENT;
SIGNAL Qm, Qs: STD_LOGIC;
BEGIN
U1: D_latch PORT MAP (NOT SW (1), SW (0), Qm);
U2: D_latch PORT MAP (SW (1), Qm, Qs);
LEDR (0) <= Qs;
END Structural;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY D_latch IS
PORT (Clk, D: IN STD_LOGIC;
Q: OUT STD_LOGIC);
END D_latch;
ARCHITECTURE Behavior OF D_latch IS
SIGNAL R, R_g, S_g, Qa, Qb: STD_LOGIC;
ATTRIBUTE keep: Boolean;
ATTRIBUTE keep of R, R_g, S_g, Qa, Qb: signal is true;
BEGIN
R <= NOT D;
S_g <= NOT (D AND Clk);
R_g <= NOT (R AND Clk);
Qa <= NOT (S_g AND Qb);
Qb <= NOT (R_g AND Qa);
Q <= Qa;
END Behavior;
```

Πρόγραμμα 11 Master slave D flip flop

4.6.4 Καταχωρητής 16 bits

Οι καταχωρητές αποτελούνται από ένα σύνολο Flip flops, τα οποία είναι κατάλληλα συνδεδεμένα μεταξύ τους και δέχονται ένα κοινό ωρολογιακό σήμα. Σε κάθε Flip flop μπορεί να αποθηκευτεί 1 bit πληροφορίας. Σε έναν καταχωρητή ο οποίος αποτελείται από n Flip flops μπορούν να αποθηκευτούν n bits πληροφορίας. Για παράδειγμα, για την δημιουργία ενός καταχωρητή χωρητικότητας 8 bits θα πρέπει να χρησιμοποιηθούν οκτώ Flip flops.

Η εισαγωγή δεδομένων σ' έναν καταχωρητή μπορεί να γίνει είτε σειριακά, είτε παράλληλα. Στην πρώτη περίπτωση σε κάθε ωρολογιακό παλμό εισάγεται σε ένα bit, ενώ στην δεύτερη περίπτωση εισάγονται και τα n bits με έναν ωρολογιακό παλμό. Το ίδιο ισχύει και για την εξαγωγή των δεδομένων. Τα δεδομένα μπορούν να εξαχθούν ή παράλληλα ή σειριακά.

Ο κώδικας των 16 bits καταχωρητή αποτελείται από τρεις εισόδους clock και Resetn που είναι 1 bits και η R που έχει εύρος 16 bits. Η έξοδος Q είναι κι αυτή 16 bits. Η είσοδος

Resetn λειτουργεί ως ασύγχρονη είσοδος μηχανισμού. Στο κομμάτι της αρχιτεκτονικής έχουμε μια διαδικασία στην οποία συμβαίνουν τα εξής: αν η είσοδος Resetn=0 τότε ο καταχωρητής δεν λειτουργεί και στην έξοδο παίρνουμε 0. Αν η είσοδος Resetn=1 τότε ο καταχωρητής λειτουργεί κανονικά και η έξοδος εξαρτάται από το ρολόι του κυκλώματος.

Κώδικας VHDL:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY part11 IS
PORT (SW: IN STD_LOGIC_VECTOR (15 DOWNTO 0) ;
KEY: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
HEX7, HEX6, HEX5, HEX4,
HEX3, HEX2, HEX1, HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part11;
ARCHITECTURE Behavior OF part11 IS
COMPONENT hex7seg
PORT (hex: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END COMPONENT;
COMPONENT regne
PORT (R: IN STD_LOGIC_VECTOR (15 DOWNTO 0) ;
Clock, Resetn: IN STD_LOGIC;
Q: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END COMPONENT;
SIGNAL A, B: STD_LOGIC_VECTOR (15 DOWNTO 0);
BEGIN
A_reg: regne PORT MAP (SW, KEY (1), KEY (0), A);
B <= SW;
digit_7: hex7seg PORT MAP (A (15 DOWNTO 12), HEX7);
digit_6: hex7seg PORT MAP (A (11 DOWNTO 8), HEX6);
digit_5: hex7seg PORT MAP (A (7 DOWNTO 4), HEX5);
digit_4: hex7seg PORT MAP (A (3 DOWNTO 0), HEX4);
digit_3: hex7seg PORT MAP (B (15 DOWNTO 12), HEX3);
digit_2: hex7seg PORT MAP (B (11 DOWNTO 8), HEX2);
digit_1: hex7seg PORT MAP (B (7 DOWNTO 4), HEX1);
digit_0: hex7seg PORT MAP (B (3 DOWNTO 0), HEX0);
END Behavior;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY regne IS
PORT (R: IN STD_LOGIC_VECTOR (15 DOWNTO 0) ;
Clock, Resetn: IN STD_LOGIC;
Q: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END regne;
ARCHITECTURE Behavior OF regne IS
BEGIN
PROCESS (Clock, Resetn)
BEGIN
IF (Resetn = '0') THEN
Q <= (OTHERS => '0');
ELSIF (Clock'EVENT AND Clock = '1') THEN
Q <= R;
END IF;
END PROCESS;
```

```
END Behavior;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY hex7seg IS
PORT (hex: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END hex7seg;
ARCHITECTURE Behavior OF hex7seg IS
BEGIN
PROCESS (hex)
BEGIN
CASE hex IS
WHEN "0000" => display <= "0000001";
WHEN "0001" => display <= "1001111";
WHEN "0010" => display <= "0010010";
WHEN "0011" => display <= "0000110";
WHEN "0100" => display <= "1001100";
WHEN "0101" => display <= "0100100";
WHEN "0110" => display <= "1100000";
WHEN "0111" => display <= "0001111";
WHEN "1000" => display <= "0000000";
WHEN "1001" => display <= "0001100";
WHEN "1010" => display <= "0001000";
WHEN "1011" => display <= "1100000";
WHEN "1100" => display <= "0110001";
WHEN "1101" => display <= "1000010";
WHEN "1110" => display <= "0110000";
WHEN OTHERS => display <= "0111000";
END CASE;
END PROCESS;
END Behavior;
```

Πρόγραμμα 12 -16- bits register

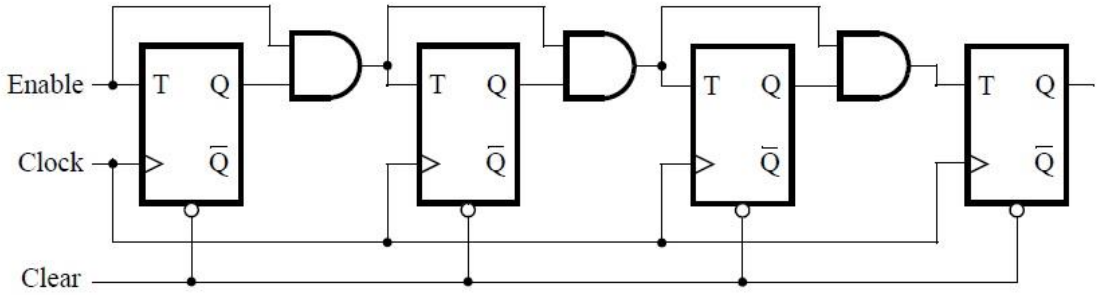
4.7 Απαριθμητές

Απαριθμητές ονομάζονται τα κυκλώματα που μπορούν να αυξήσουν ή να μειώσουν την τιμή της εξόδου τους κατά ένα, όταν στην είσοδο του ρολογιού δέχονται παλμό clock. Τα κυκλώματα των απαριθμητών χρησιμοποιούνται για να μετρούν πόσες φορές εμφανίστηκε κάποιο γεγονός, για την απαρίθμηση διαδοχικών εργασιών σε ένα σύστημα καθώς και για την παραγωγή χρονικών καθυστερήσεων. Η κατασκευή τέτοιων κυκλωμάτων γίνεται με τη χρήση καταχωρητών και με Flip flops διαφόρων ειδών.

4.7.1 Απαριθμητής 4 bits

Ο απαριθμητής τεσσάρων bits έχει μια είσοδο μηδενισμού και μια είσοδο ενεργοποίησης. Η είσοδος clear είναι η είσοδος μηδενισμού και η είσοδος enable είναι η είσοδος ενεργοποίησης. Όταν η είσοδος clear είναι μηδέν τότε η έξοδος count του κυκλώματος μηδενίζεται. Ενώ αν η είσοδος clear είναι ίση με τη μονάδα τότε η είσοδος ενεργοποίησης

enable είναι κι αυτή 1, τότε η τιμή της εξόδου του μετρητή θα αυξάνεται κατά ένα σε κάθε θετικό ωρολογιακό μέτωπο αλλιώς όταν η enable είναι μηδέν η τιμή της εξόδου παραμένει αμετάβλητη.



Σχήμα 23 -4- bit counter

Παρακάτω φαίνεται το πρόγραμμα του απαριθμητή 4 bits:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part12 IS
PORT (SW: IN STD_LOGIC_VECTOR (1 DOWNTO 0) ;
KEY: IN STD_LOGIC_VECTOR (0 DOWNTO 0);
HEX0: OUT STD_LOGIC_VECTOR (0 TO 6));
END part12;
ARCHITECTURE Behavior OF part12 IS
COMPONENT ToggleFF
PORT (T, Clock, Resetn: IN STD_LOGIC;
Q: OUT STD_LOGIC);
END COMPONENT;
COMPONENT hex7seg
PORT (hex: IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END COMPONENT;
SIGNAL Clock, Resetn: STD_LOGIC;
SIGNAL Count, Enable: STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
Clock <= KEY (0);
Resetn <= SW (0);
Enable (0) <= SW (1);
TFF0: ToggleFF PORT MAP (Enable (0), Clock, Resetn, Count (0));
Enable (1) <= Count (0) AND Enable (0);
TFF1: ToggleFF PORT MAP (Enable (1), Clock, Resetn, Count (1));
Enable (2) <= Count (1) AND Enable (1);
TFF2: ToggleFF PORT MAP (Enable (2), Clock, Resetn, Count (2));
Enable (3) <= Count (2) AND Enable (2);
TFF3: ToggleFF PORT MAP (Enable (3), Clock, Resetn, Count (3));
digit0: hex7seg PORT MAP (Count (3 DOWNTO 0), HEX0);
END Behavior;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY ToggleFF IS
PORT (T, Clock, Resetn: IN STD_LOGIC;
```

```
Q: OUT STD_LOGIC);
END ToggleFF;
ARCHITECTURE Behavior OF ToggleFF IS
SIGNAL T_out: STD_LOGIC;
BEGIN
PROCESS (Clock)
BEGIN
IF (Clock'EVENT AND Clock = '1') THEN
IF (Resetn = '0') THEN
T_out <= '0';
ELSIF (T = '1') THEN
T_out <= NOT T_out;
END IF;
END IF;
END PROCESS;
Q <= T_out;
END Behavior;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY hex7seg IS
PORT (hex: IN STD_LOGIC_VECTOR (3 DOWNTO 0;
Display: OUT STD_LOGIC_VECTOR (0 TO 6));
END hex7seg;
ARCHITECTURE Behavior OF hex7seg IS
BEGIN
PROCESS (hex)
BEGIN
CASE hex IS
WHEN "0000" => display <= "0000001";
WHEN "0001" => display <= "1001111";
WHEN "0010" => display <= "0010010";
WHEN "0011" => display <= "0000110";
WHEN "0100" => display <= "1001100";
WHEN "0101" => display <= "0100100";
WHEN "0110" => display <= "1100000";
WHEN "0111" => display <= "0001111";
WHEN "1000" => display <= "0000000";
WHEN "1001" => display <= "0001100";
WHEN "1010" => display <= "0001000";
WHEN "1011" => display <= "1100000";
WHEN "1100" => display <= "0110001";
WHEN "1101" => display <= "1000010";
WHEN "1110" => display <= "0110000";
WHEN OTHERS => display <= "0111000";
END CASE;
END PROCESS;
END Behavior;
```

Πρόγραμμα 13 -4- bit counter

ΚΕΦΑΛΑΙΟ 5

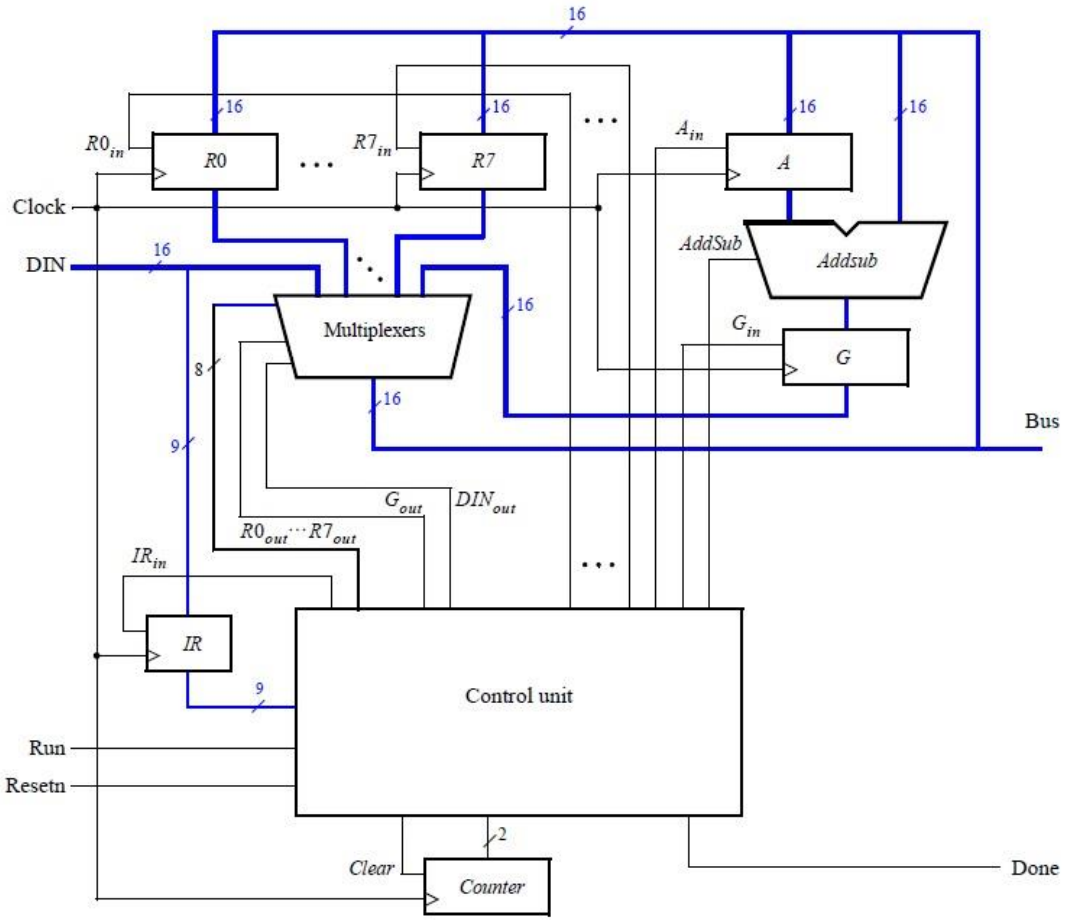
Σχεδίαση του επεξεργαστή

5.1 Περιγραφή της λειτουργίας του επεξεργαστή

Στο κεφάλαιο που ακολουθεί παρουσιάζεται η λειτουργία του κυκλώματος ενός απλού επεξεργαστή ο οποίος θα εκτελεί τέσσερις πράξεις:

- Πρόσθεση (add)
- Αφαίρεση (sub)
- Φόρτωση (mvi)
- Μετακίνηση (mv)

Στο *Σχήμα 24* που φαίνεται παρακάτω παρουσιάζεται ένα ψηφιακό σύστημα που αποτελείται από καταχωρητές των 16 bit, έναν πολυπλέκτη, έναν αθροιστή/αφαιρέτη και μια μονάδα ελέγχου. Τα δεδομένα εισάγονται στο σύστημα μέσω της εισόδου DIN η οποία είναι εύρους 16 bit. Τα δεδομένα μπορούν να μεταφέρονται μέσω του πολυπλέκτη στους διάφορους καταχωρητές. Ο πολυπλέκτης επιτρέπει επίσης, τη μεταφορά δεδομένων από τον έναν καταχωρητή στον άλλο.



Σχήμα 24 Digital System

Η πρόσθεση ή η αφαίρεση πραγματοποιείται με τη χρήση του πολυπλέκτη για να προστεθεί πρώτα ένας αριθμός των 16 bit στο δίαυλο και να φορτωθεί ο συγκεκριμένος αριθμός στον καταχωρητή A. Μόλις γίνει αυτό, ένας δεύτερος αριθμός των 16 bit τοποθετείται στον δίαυλο επικοινωνίας, το κύκλωμα του αθροιστή/αφαιρέτη εκτελεί την απαιτούμενη λειτουργία και το αποτέλεσμα φορτώνεται στον καταχωρητή G. Τα δεδομένα στον καταχωρητή G μπορούν να μεταφερθούν σε έναν από τους υπόλοιπους καταχωρητές.

Το κύκλωμα μπορεί να εκτελεί διαφορετικές λειτουργίες σε κάθε κύκλο ρολογιού, όπως ρυθμίζεται από τη μονάδα ελέγχου. Η μονάδα ελέγχου καθορίζει πότε τα δεδομένα θα τοποθετηθούν στο δίαυλο επικοινωνίας και ελέγχει σε ποιον από τους καταχωρητές θα φορτωθούν τα δεδομένα. Για παράδειγμα, αν η μονάδα ελέγχου επιβεβαιώσει τα σήματα R0out και Ain, τότε ο πολυπλέκτης θα τοποθετήσει τα περιεχόμενα του καταχωρητή R0 στο δίαυλο επικοινωνίας και αυτά τα δεδομένα θα φορτωθούν στον επόμενο ενεργό παλμό ρολογιού στον καταχωρητή A.

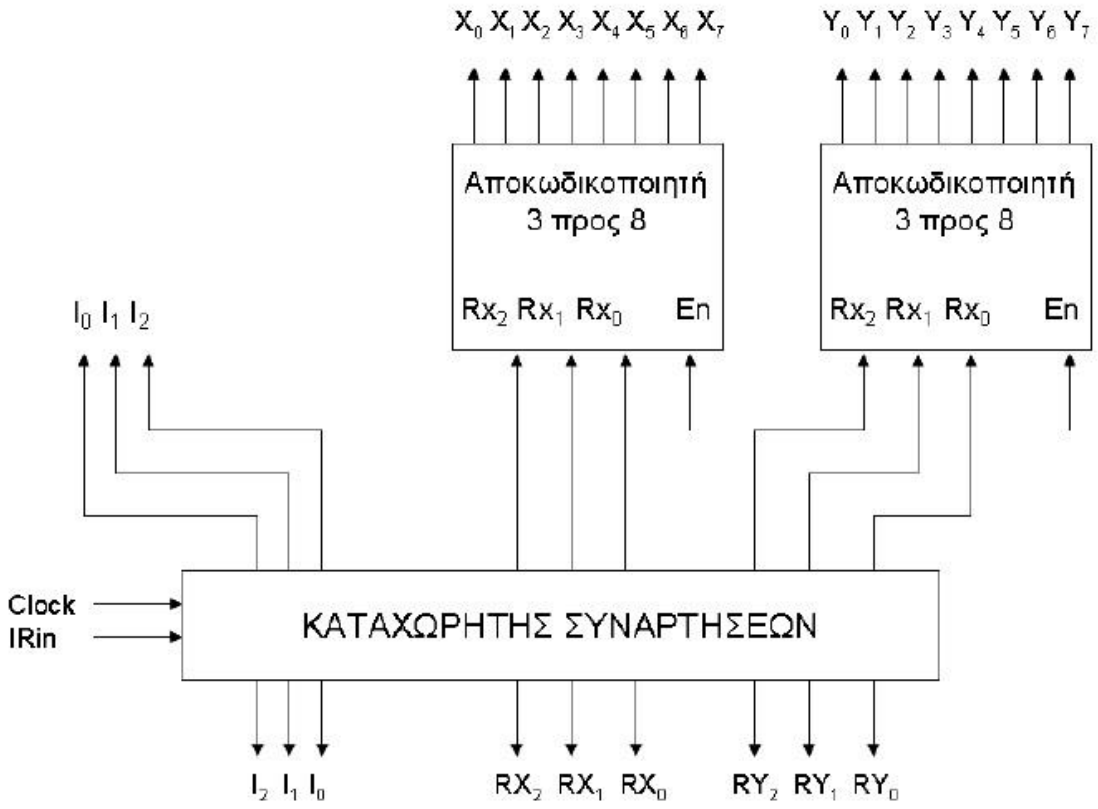
Ένα κύκλωμα όπως το παραπάνω συχνά ονομάζεται επεξεργαστής. Στον *Πίνακα 12* παρουσιάζονται οι οδηγίες του επεξεργαστή. Στην αριστερή στήλη βρίσκονται το όνομα μιας εντολής και του τελεστή της. Η έννοια της σύνθεσης RX ← [RY] είναι ότι τα περιεχόμενα του καταχωρητή RY φορτώνονται στον καταχωρητή RX. Με την εντολή mv (μετακίνηση) πραγματοποιείται η αντιγραφή δεδομένων από τον έναν καταχωρητή στον άλλο. Για την εντολή mvi (φόρτωση), η έκφραση RX ← D δηλώνει ότι η 16 bit σταθερά D έχει φορτωθεί στον καταχωρητή RX.

Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Πίνακας 12 Εντολές εκτέλεσης του επεξεργαστή

5.2 Περιγραφή των εντολών του επεξεργαστή

Το κύκλωμα ελέγχου του επεξεργαστή περιέχει κυκλώματα τα οποία είναι υπεύθυνα για τη σωστή λειτουργία του. Στην είσοδο IRin η οποία είναι μεγέθους 9 bits δηλώνεται η εντολή Instruction, δηλαδή η εντολή που θα εκτελέσει ο επεξεργαστής. Η είσοδος IRin αποθηκεύεται σε έναν καταχωρητή εντολής IR. Για τις λειτουργίες της μετακίνησης, της φόρτωσης, της πρόσθεσης και της αφαίρεσης χρησιμοποιούνται οι τιμές I2I1I0=000, I2I1I0=001, I2I1I0=010 και I2I1I0=011 αντίστοιχα. Με αυτό τον τρόπο χρησιμοποιούνται τα πρώτα τρία bits του κωδικού της εντολής για τον προσδιορισμό της πράξης.



Σχήμα 25 Καταχωρητής Εντολών

Για την επιλογή του καταχωρητή όπου θα μεταφερθούν τα δεδομένα χρησιμοποιούνται τα επόμενα τρία bits (Rx2, Rx1, Rx0) και τα τρία τελευταία bits (Ry2, Ry1, Ry0) χρησιμοποιούνται για την επιλογή του δεύτερου καταχωρητή (operands). Όπως, φαίνεται και στο *Σχήμα 25* χρησιμοποιούνται δύο αποκωδικοποιητές τρία προς οκτώ, οι οποίοι βρίσκονται στην έξοδο του καταχωρητή συνάρτησης. Υπάρχει ένα αποκωδικοποιητής για κάθε τριάδα bit Rx και Ry. Ο κάθε αποκωδικοποιητής επιλέγει έναν από τους οχτώ διαθέσιμους καταχωρητές (R0 – R7) του επεξεργαστή, με τη βοήθεια των σημάτων Xreg και Yreg που παράγει στην έξοδο.

Οι οχτώ καταχωρητές που υπάρχουν στο κύκλωμα αποτελούνται από τα σήματα ενεργοποίησης Rin (0,1,2,3,4,5,6,7). Με τη βοήθεια του πολυπλέκτη ο οποίος στον κώδικα VHDL υλοποιείται με μια δομή IF, οι έξοδοι των καταχωρητών μοιράζονται στο δίαυλο επικοινωνίας. Το σήμα επιλογής Sel έχει 10 bits, ώστε ενεργοποιεί κάθε φορά την έξοδο στη δίαυλο ενός από τους R0, R1, R2, R3, R4, R5, R6, R7, G και DIN (Δεδομένα).

Οι δυο αποκωδικοποιητές ενεργοποιούνται διαρκώς όταν η είσοδος ενεργοποίησης enable βρίσκεται μόνιμα στην κατάσταση 1. Όταν η είσοδος IRin είναι ίση με 1 τότε ο καταχωρητής συνάρτησης αποθηκεύει τις τιμές που έχουν οι είσοδοι. Μέσω εξωτερικών διακοπτών τύπου dip δίνονται στο σύστημα τα δεδομένα προς επεξεργασία καθώς και τα σήματα Clock, W και Resetn.

Το κύκλωμα ελέγχου περιέχει επίσης έναν απαριθμητή, ο οποίος χρησιμεύει στην δημιουργία των απαιτούμενων σημάτων ελέγχου σε κάθε βήμα. Ο απαριθμητής θα είναι των δύο bits και οι τιμές που μπορεί να πάρει η έξοδος T θα είναι: ‘00’, ‘01’, ‘10’, ‘11’ επειδή οι λειτουργίες της πρόσθεσης και της αφαίρεσης του επεξεργαστή απαιτούν μέχρι και τρεις κύκλους εκτέλεσης.

Στην είσοδο του απαριθμητή υπάρχει ένα σήμα Clock που παίρνει παλμούς από το εξωτερικό ρολόι και ένα σήμα Clear. Κάθε μία από τις εξόδους του αποκωδικοποιητή αντιπροσωπεύει ένα στάδιο της λειτουργίας. Όταν δεν εκτελείται κανένα στάδιο, ο απαριθμητής έχει την τιμή 00 και ενεργοποιείται η έξοδος T_0 του αποκωδικοποιητή. Ενώ, στην περίπτωση που ο απαριθμητής πάρει την τιμή 01, ενεργοποιείται η έξοδος T_1. Αυτό συμβαίνει όταν εκτελεστεί μία από τις εντολές μετακίνησης ή φόρτωσης (I=000 ή I=001). Η παραπάνω διαδικασία, έχει σαν αποτέλεσμα την παραγωγή των κατάλληλων σημάτων DINout για την φόρτωση του διαύλου μέσω του πολυπλέκτη με δεδομένα εισόδου (DIN) από τους εξωτερικούς διακόπτες εισόδου, και RXin για την φόρτωση του κατάλληλου καταχωρητή. Όταν εκτελείται η εντολή πρόσθεσης (I= ‘010’), τότε για T1=‘01’ τα δεδομένα του Rx εξάγονται στη δίαυλο (RXout=1) και μετακινούνται στον καταχωρητή A (Ain=1). Ο Rx θα εξάγει τα δεδομένα του στο bus, όταν το σήμα επιλογής Sel του πολυπλέκτη, που τροφοδοτεί τη δίαυλο, παίρνει την κατάλληλη τιμή.

Τα σήματα Ain, Gin και Gout δημιουργούνται στις λειτουργίες της πρόσθεσης και της αφαίρεσης. Στο στάδιο T_2=10, ο Ry εξάγει τα δεδομένα του στο δίαυλο, ώστε να μπορούν να αθροιστούν με τα δεδομένα του A (RYout=1) και ο καταχωρητής G θα φορτωθεί με το αποτέλεσμα της πρόσθεσης (ή της αφαίρεσης, ανάλογα με την τιμή του σήματος AddSub), οπότε Gin=1.

Στο τρίτο και τελευταίο στάδιο, όπου T_3=11, ο καταχωρητής G εξάγει το αποτέλεσμα της πράξης (add ή sub) στο δίαυλο (Gout=1).

Κάθε φορά που ολοκληρώνεται μία εντολή του επεξεργαστή, θα δημιουργείται από το κύκλωμα ελέγχου το σήμα Done το οποίο θα παίρνει την τιμή 1. Η είσοδος IR θα στέλνει

τα δεδομένα της στους αποκωδικοποιητές του κυκλώματος ελέγχου κάθε φορά που ο χρήστης θα δίνει εξωτερικά το σήμα W.

Το σήμα Clear χρησιμοποιείται για να διασφαλίσει ότι η τιμή του απαριθμητή παραμένει μηδέν όταν δεν εκτελείται καμία λειτουργία. Το συγκεκριμένο σήμα χρησιμοποιείται σαν είσοδος στον απαριθμητή. Το σήμα IRin δίνει την εντολή να φορτωθούν τα περιεχόμενα της εντολής στον καταχωρητή συναρτήσεων.

Στον *Πίνακα 13* φαίνονται τα σήματα ελέγχου του επεξεργαστή που πραγματοποιούνται σε κάθε στάδιο.

	T_0 = '00'	T_1 = '01'	T_2 = '10'	T_3 = '11'
Μετακίνηση (**mv**)	-	Rout←[Y] Rin=[X] Done=1	-	-
Φόρτωση (**mv**)	-	DINout=1 Rin=[X] Done=1	-	-
Πρόσθεση (**add**)	-	Rout←[X] Ain=1	Rout←[Y] Gin=1 AddSub=0	Gout=1 Rin←[X] Done=1
Αφαίρεση (**sub**)	-		Rout←[Y] Gin=1 AddSub=1	

Πίνακας 13 Σήματα ελέγχου του επεξεργαστή σε κάθε στάδιο

5.3 Κυκλώματα που συνθέτουν τον επεξεργαστή

Για την υλοποίηση του κυκλώματος του επεξεργαστή χρησιμοποιούνται μικρότερα κυκλώματα τα οποία είναι ένας απαριθμητής (upcount), ένας αποκωδικοποιητής τρία προς οκτώ (dec3to8) και ένας καταχωρητής (regn).

5.3.1 Ο απαριθμητής (upcount)

Ένας απαριθμητής αποτελείται από δύο εισόδους. Η μια είσοδος αποτελεί την είσοδο μηδενισμού του απαριθμητή και η άλλη είσοδος το clock. Έχει και μια έξοδο η οποία είναι δυο bits. Όταν η είσοδος clear γίνεται ίση με τη μονάδα ο απαριθμητής μηδενίζεται και η

έξοδος Q παίρνει την τιμή ‘00’. Ενώ, όταν η είσοδος clear γίνει 0 για κάθε θετικό ωρολογιακό παλμό η έξοδος Q(count) θα αυξάνεται κατά μια μονάδα.

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY upcount IS
PORT (Clear, Clock: IN STD_LOGIC;
Q: OUT STD_LOGIC_VECTOR (1 DOWNTO 0));
END upcount;
ARCHITECTURE Behavior OF upcount IS
SIGNAL Count: STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
PROCESS (Clock)
 BEGIN
   IF (Clock'EVENT AND Clock = '1') THEN
     IF Clear THEN
       Count <= “00”;
     ELSE
       Count <= Count + 1;
     END IF;
  END IF;
END PROCESS;
Q <= Count;
END Behavior;
```

Πρόγραμμα 14 Απαριθμητής

5.3.2 Αποκωδικοποιητής 3x8

Η χρήση του αποκωδικοποιητή γίνεται ώστε να επιλεγεί μια έξοδος με βάση την πληροφορία των γραμμών επιλογής. Το κύκλωμα ενός αποκωδικοποιητή διαθέτει μια είσοδο ενεργοποίησης, η οποία όταν είναι 1 ενεργοποιεί τις εξόδους ενώ όταν είναι 0 απενεργοποιεί τις εξόδους. Στον κώδικα του αποκωδικοποιητή ο συνδυασμός των εισόδων En και W που χρησιμοποιούνται στο σώμα της αρχιτεκτονικής μέσα σε μια διαδικασία και καθορίζουν ποια έξοδος θα ενεργοποιείται.

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY decoder3to8 IS
PORT (W: IN STD_LOGIC_VECTOR (2 DOWN TO 0) ;
     En: IN STD_LOGIC;
     Y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decoder3to8;
ARCHITECTURE Behavior OF decoder3to8 IS
BEGIN
 PROCESS (W, En)
 BEGIN
   IF (En = '1') THEN
      CASE W IS
        WHEN "000" => Y <= "10000000";
        WHEN "001" => Y <= "01000000";
        WHEN "010" => Y <= "00100000";
```

```
        WHEN "011" => Y <= "00010000";
        WHEN "100" => Y <= "00001000";
        WHEN "101" => Y <= "00000100";
        WHEN "110" => Y <= "00000010";
        WHEN "111" => Y <= "00000001";
      END CASE;
   ELSE
      Y <= "00000000";
   END IF;
END PROCESS;
END Behavior;
```

Πρόγραμμα 15 Αποκωδικοποιητής 3x8

5.3.3 Καταχωρητής

Το παρακάτω κύκλωμα του καταχωρητή αποτελείται από n bits. Το μέγεθος ενός καταχωρητή είναι μεταβλητό και προσδιορίζεται από την τιμή που δίνουμε στο n. Όταν η είσοδος ενεργοποίησης Rin ισούται με 1 τότε τα flip flop φορτώνουν την τιμή της εισόδου R, ενώ όταν η είσοδος Rin είναι ίση με το μηδέν τα flip flop διατηρούν την προηγούμενη τιμή που τους είχε δοθεί.

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
ENTITY regn IS
 GENERIC (n: INTEGER:= 16);
PORT (R: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0);
Rin, Clock: IN STD_LOGIC;
Q: BUFFER STD_LOGIC_VECTOR (n-1 DOWNTO 0));
END regn;
ARCHITECTURE Behavior OF regn IS
BEGIN
 PROCESS (Clock)
 BEGIN
   IF (Clock'EVENT AND Clock = '1') THEN
     IF Rin='1' THEN
       Q <= R;
     END IF;
  END IF;
END PROCESS;
END Behavior;
```

Πρόγραμμα 16 Καταχωρητής regn

5.4 Κυρίως πρόγραμμα του επεξεργαστή

Σε αυτή την ενότητα παρουσιάζεται ο κώδικας VHDL του επεξεργαστή ο οποίος χωρίζεται σε τρία τμήματα. Στο πρώτο τμήμα δηλώνονται οι βιβλιοθήκες, στο δεύτερο τμήμα δηλώνεται η οντότητα όπου περιέχονται οι είσοδοι και οι έξοδοι του κυκλώματος και στο

τρίτο τμήμα δηλώνεται η αρχιτεκτονική όπου γίνεται η περιγραφή της λειτουργίας του επεξεργαστή.

Η δήλωση των βιβλιοθηκών είναι η εξής:

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

Στο τμήμα της οντότητας δηλώνονται οι είσοδοι και οι έξοδοι του επεξεργαστή. Η οντότητα έχει το όνομα Proc

```
ENTITY proc IS
PORT ( DIN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
Resetn, Clock, Run : IN STD_LOGIC;
Done : BUFFER STD_LOGIC;
BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0));
END proc;
```

Το τμήμα της αρχιτεκτονικής έχει το όνομα Behavior και αναφέρεται στην οντότητα Proc. Στο συγκεκριμένο τμήμα δηλώνονται τα υποκυκλώματα που θα υλοποιήσουν τον επεξεργαστή τα οποία είναι ένας απαριθμητής, ένας αποκωδικοποιητής και ένας καταχωρητής.

```
ARCHITECTURE Behavior OF proc IS
COMPONENT upcount
PORT ( Clear, Clock : IN STD_LOGIC;
Q : BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0));
END COMPONENT;
COMPONENT dec3to8
PORT ( W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
En : IN STD_LOGIC;
Y : OUT STD_LOGIC_VECTOR(0 TO 7));
END COMPONENT;
COMPONENT regn
GENERIC (n : INTEGER := 16);
PORT ( R : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
Rin, Clock : IN STD_LOGIC;
Q : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END COMPONENT;
SIGNAL Rin, Rout : STD_LOGIC_VECTOR(0 TO 7);
SIGNAL Sum : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL Clear, High, IRin, DINout, Ain, Gin, Gout, AddSub : STD_LOGIC;
SIGNAL Tstep_Q : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL I : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL Xreg, Yreg : STD_LOGIC_VECTOR(0 TO 7);
SIGNAL R0, R1, R2, R3, R4, R5, R6, R7, A, G : STD_LOGIC_VECTOR(15 DOWNTO
0);
SIGNAL IR : STD_LOGIC_VECTOR(1 TO 9);
SIGNAL Sel : STD_LOGIC_VECTOR(1 to 10); -- bus selector
```

Στο σώμα της αρχιτεκτονικής χρησιμοποιείται ένα στιγμιότυπο Tstep το οποίο αναφέρεται στον απαριθμητή. Ο απαριθμητής παίρνει σαν είσοδο το σήμα Clear το οποίο ορίζεται ως εξής:

```
Clear <= NOT(Resetn) OR Done OR (NOT(Run) AND NOT(Tstep_Q(1)) AND
NOT(Tstep_Q(0)));
```

Η παραπάνω έκφραση δηλώνει ότι όταν το Clear είναι ίσο με τη μονάδα (δηλαδή όταν μία από τις τρεις εισόδους της πύλης or είναι ίση με τη μονάδα) τότε ο απαριθμητής θα μηδενίζεται.

Το επόμενο σήμα που ακολουθεί είναι το I το οποίο παίρνει ως τιμή τα τρία πρώτα bits της συνάρτησης IR:

```
I <= IR(1 TO 3);
```

Τα επόμενα στιγμιότυπα decX και decY χρησιμοποιούνται για την αποκωδικοποίηση του σήματος count και έχουν ως εξόδους τις T0, T1, T2, T3 και I0, I1, I2,I3 αντίστοιχα. Οι αποκωδικοποιητές θα είναι μόνιμα ενεργοποιημένοι όταν στην είσοδο ενεργοποίησης το σήμα High πάρει την τιμή 1.

```
High <= '1';
Clear <= NOT(Resetn) OR Done OR (NOT(Run) AND NOT(Tstep_Q(1)) AND
NOT(Tstep_Q(0)));
Tstep: upcount PORT MAP (Clear, Clock, Tstep_Q);
I <= IR(1 TO 3);
decX: dec3to8 PORT MAP (IR(4 TO 6), High, Xreg);
decY: dec3to8 PORT MAP (IR(7 TO 9), High, Yreg);
```

Η διαδικασία με το όνομα controlsignals, περιέχει τα σήματα T, I, Xreg και Yreg. Στη διαδικασία αυτή, ορίζεται η λειτουργία του κυκλώματος ελέγχου με βάση (α) το στάδιο της τωρινής κατάστασης και με (β) την πράξη που εκτελείται.

Ο συγκεκριμένος τρόπος περιγραφής του κυκλώματος δημιουργεί δύο εμφωλευμένες εντολές CASE. Υπάρχουν δύο επίπεδα εντολών CASE. Στο πρώτο επίπεδο υπολογίζονται οι πιθανές τιμές του σήματος T. Στο δεύτερο επίπεδο εξετάζονται οι τιμές που παίρνει το σήμα I για κάθε τιμή του T. Αρχικά, η τιμή του T θα είναι ‘00’ και στην περίπτωση αυτή κανένα σήμα δεν θα ενεργοποιηθεί καθώς καμία εντολή δεν ενεργοποιείται.

Όταν το T πάρει την τιμή ‘01’, τότε ενεργοποιούνται οι τέσσερις πράξεις και γίνεται έλεγχος των σημάτων που θα δημιουργηθούν για την κάθε πράξη στο δεύτερο επίπεδο CASE. Για I=‘000’ (μετακίνηση) ενεργοποιούνται τα σήματα Rout, Rin και Done. Τα σήματα DINout, Rin και Done χρησιμοποιούνται για την πράξη της φόρτωσης δηλαδή, όταν I=‘001’. Η

έξοδος Done δεν ενεργοποιείται στις πράξεις της αφαίρεσης και της πρόσθεσης καθώς απαιτούνται τρία ωρολογιακά βήματα ενώ για την μεταφορά και τη φόρτωση απαιτείται μόνο ένα ωρολογιακό βήμα.

Στο στάδιο όπου το T θα πάρει την τιμή ‘10’ μόνο η πρόσθεση και η αφαίρεση μπορούν να ενεργοποιήσουν κάποιο σήμα, κι έτσι οι τιμές που θα πάρει το σήμα I είναι οι ‘010’ και ‘011’.

Στο τελευταίο στάδιο δίνεται στο Done η τιμή ‘1’, η οποία σηματοδοτεί το τέλος της πράξης. Οι τιμές που μπορεί να πάρει το I είναι της πρόσθεσης και της αφαίρεσης. Η διαδικασία τελειώνει όταν ολοκληρωθούν όλες οι πιθανές τιμές που μπορούν να πάρουν τα σήματα I και T.

```
-- Instruction Table
-- 000: mv Rx,Ry : Rx <- [Ry]
-- 001: mvi Rx,#D : Rx <- D
-- 010: add Rx,Ry : Rx <- [Rx] + [Ry]
-- 011: sub Rx,Ry : Rx <- [Rx] - [Ry]
-- OPCODE format: III XXX YYY, where
-- III = instruction, XXX = Rx, and YYY = Ry. For mvi,
-- a second word of data is loaded from DIN
--
controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
BEGIN
Done <= '0'; Ain <= '0'; Gin <= '0'; Gout <= '0'; AddSub <= '0';
IRin <= '0'; DINout <= '0'; Rin <= "00000000"; Rout <= "00000000";
CASE Tstep_Q IS
WHEN "00" => -- store DIN in IR as long as Tstep_Q = 0
IRin <= '1';
WHEN "01" => -- define signals in time step T1
CASE I IS
WHEN "000" => -- mv Rx,Ry
Rout <= Yreg;
Rin <= Xreg;
Done <= '1';
WHEN "001" => -- mvi Rx,#D
-- data is required to be on DIN
DINout <= '1';
Rin <= Xreg;
Done <= '1';
WHEN "010" => -- add
Rout <= Xreg;
Ain <= '1';
-- WHEN "011" => -- sub
WHEN OTHERS => -- sub
Rout <= Xreg;
Ain <= '1';
-- WHEN OTHERS => ;
END CASE;
WHEN "10" => -- define signals in time step T2
CASE I IS
WHEN "010" => -- add
Rout <= Yreg;
Gin <= '1';
```

```
-- WHEN "011" => -- sub
WHEN OTHERS => -- sub
Rout <= Yreg;
AddSub <= '1';
Gin <= '1';
-- WHEN OTHERS => ;
END CASE;
WHEN "11" => -- define signals in time step T3
CASE I IS
WHEN "010" => -- add
Gout <= '1';
Rin <= Xreg;
Done <= '1';
-- WHEN "011" => -- sub
WHEN OTHERS => -- sub
Gout <= '1';
Rin <= Xreg;
Done <= '1';
-- WHEN OTHERS => ;
END CASE;
END CASE;
END PROCESS;
```

Στα επόμενα στιγμιότυπα (reg_0 ως reg_A) δηλώνονται οι καταχωρητές που χρησιμοποιήθηκαν στις παραπάνω περιπτώσεις (R0,R1,R2,R3,R4,R5,R6,R7,A).

```
reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
reg_1: regn PORT MAP (BusWires, Rin(1), Clock, R1);
reg_2: regn PORT MAP (BusWires, Rin(2), Clock, R2);
reg_3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
reg_4: regn PORT MAP (BusWires, Rin(4), Clock, R4);
reg_5: regn PORT MAP (BusWires, Rin(5), Clock, R5);
reg_6: regn PORT MAP (BusWires, Rin(6), Clock, R6);
reg_7: regn PORT MAP (BusWires, Rin(7), Clock, R7);
reg_A: regn PORT MAP (BusWires, Ain, Clock, A);
```

Η τιμή του σήματος AddSub καθορίζει ποια πράξη θα εκτελέσει ο αθροιστής/αφαιρέτης. Ανάλογα με την τιμή του AddSub, στο στιγμιότυπο reg_IR καθορίζεται η πράξη που θα εκτελεστεί μεταξύ του καταχωρητή A και του αριθμού που βρίσκεται στο δίαυλο. Το αποτέλεσμα της πράξης θα αποθηκευτεί στον καταχωρητή G (reg_G).

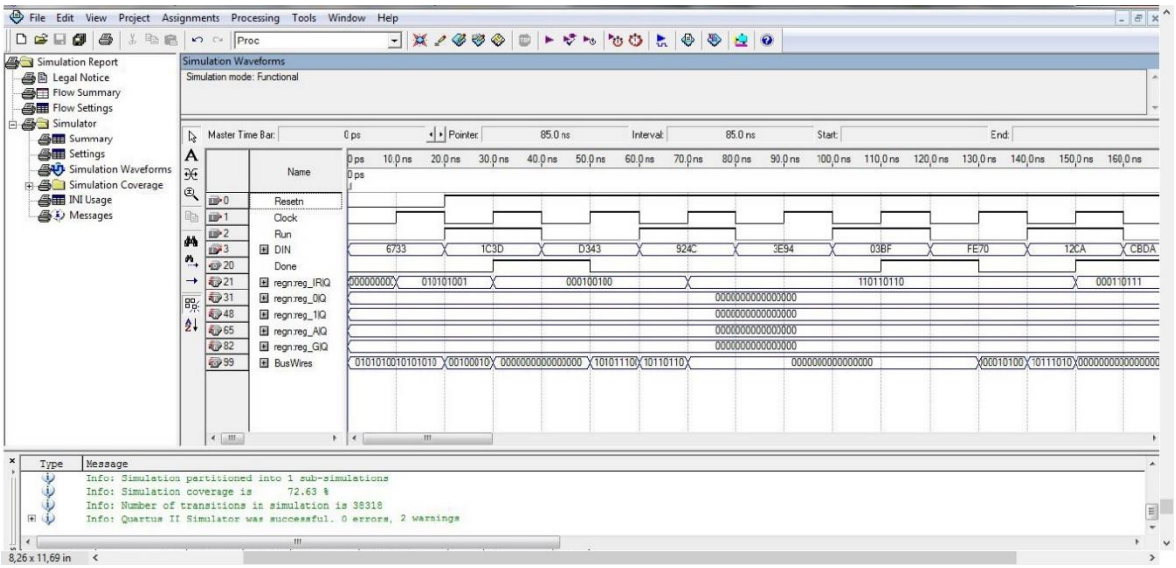
```
reg_IR: regn GENERIC MAP (n => 9) PORT MAP (DIN(15 DOWNTO 7), IRin,
Clock, IR);
-- alu
alu: PROCESS (AddSub, A, BusWires)
BEGIN
IF AddSub = '0' THEN
Sum <= A + BusWires;
ELSE
Sum <= A - BusWires;
END IF;
END PROCESS;
reg_G: regn PORT MAP (Sum, Gin, Clock, G);
```

Στο τελευταίο κομμάτι του κώδικα περιγράφεται η δίαυλος επικοινωνίας (bus), αντιστοιχίζοντας τα σήματα επιλογής του πολυπλέκτη που την υλοποιεί. Ο δίαυλος θα πάρει τα περιεχόμενα του αντίστοιχου καταχωρητή ανάλογα με την τιμή του σήματος Sel.

```
-- define the internal processor bus
Sel <= Rout & Gout & DINout;
busmux: PROCESS (Sel, R0, R1, R2, R3, R4, R5, R6, R7, G, DIN)
BEGIN
IF Sel = "1000000000" THEN
BusWires <= R0;
ELSIF Sel = "0100000000" THEN
BusWires <= R1;
ELSIF Sel = "0010000000" THEN
BusWires <= R2;
ELSIF Sel = "0001000000" THEN
BusWires <= R3;
ELSIF Sel = "0000100000" THEN
BusWires <= R4;
ELSIF Sel = "0000010000" THEN
BusWires <= R5;
ELSIF Sel = "0000001000" THEN
BusWires <= R6;
ELSIF Sel = "0000000100" THEN
BusWires <= R7;
ELSIF Sel = "0000000010" THEN
BusWires <= G;
ELSE
BusWires <= DIN;
END IF;
END PROCESS;
END Behavior;
```

5.5 Προσομοίωση του επεξεργαστή

Στο *Σχήμα 26* παρουσιάζεται η προσομοίωση του επεξεργαστή η οποία εκτελείται μετά την επιτυχή διαδικασία της μετάφρασης. Κάθε ωρολογιακός παλμός του διαγράμματος κατά τον οποίο η είσοδος Run έχει τεθεί σε ‘1’ σημαίνει την έναρξη κάποιας λειτουργίας. Η είσοδος ενεργοποίησης του επεξεργαστή είναι η είσοδος Resetn η οποία αρχικά θα πάρει την τιμή ‘0’ και στη συνέχεια θα γίνει ίση με τη μονάδα ώστε να ξεκινήσει η λειτουργία του κυκλώματος του επεξεργαστή.



Σχήμα 26 Λειτουργική προσομοίωση του επεξεργαστή

5.6 Υλοποίηση και αντιστοίχιση των ακροδεκτών

Η διάταξη του κυκλώματος είναι ένα FPGA που ανήκει στην οικογένεια Cyclone II της εταιρίας Altera και συγκεκριμένα η διάταξη EP2C35F672C6.

Ορίζονται εννέα διακόπτες (push-buttons) της αναπτυξιακής πλακέτας DE2 οι οποίοι δίνουν τιμές στην συνάρτηση IR, δηλαδή στο opcode της εντολής. Η εντολή που εκτελείται θα δίνεται με τη βοήθεια αυτών των διακοπτών. Επίσης, ορίζονται δεκαέξι toggle switches που χρησιμοποιούνται σαν πηγή δεδομένων της εισόδου (DIN). Οι συγκεκριμένοι διακόπτες οδηγούνται στους κατάλληλους καταχωρητές του κυκλώματος με τη βοήθεια εντολών.

Υπάρχουν τρεις διακόπτες όπου ο ένας ορίζεται ως Clock, άλλος ένας διακόπτης είναι το εξωτερικό σήμα Run και ένας ακόμη διακόπτης είναι το σήμα Resetn.

Οι τιμές των bits της διαύλου επικοινωνίας (bus) απεικονίζονται σε 16 φωτεινούς ενδείκτες (LEDs) εξόδου, στους οποίους θα εμφανιστεί η τιμή που περνά στη δίαυλο σε κάθε ωρολογιακό βήμα. Το αποτέλεσμα της πράξης θα εμφανίζεται στους ακροδέκτες του διαύλου, στο τέλος κάθε πρόσθεσης ή αφαίρεσης, δηλαδή όταν Done=1. Μετά το τέλος κάθε εντολής και αφού ολοκληρωθούν τα βήματα του μετρητή, ένα τελευταίο LED θα απεικονίζει το σήμα Done.

ΚΕΦΑΛΑΙΟ 6

Συμπεράσματα

Στην παρούσα εργασία παρουσιάστηκε η γλώσσα VHDL και πως είναι εφικτό ένας χρήστης να συντάξει κώδικα με σκοπό την υλοποίηση ενός ψηφιακού κυκλώματος. Η σχεδίαση ψηφιακών κυκλωμάτων παρουσιάστηκε με τη βοήθεια του προγράμματος Quartus II.

Με τη σχεδίαση αρχικά πιο απλών και στη συνέχεια πιο σύνθετων κυκλωμάτων ένας αρχάριος χρήστης θα μπορεί να κατανοήσει την ψηφιακή σχεδίαση μέσω διαμορφούμενων κυκλωμάτων FPGAs.

Οι διατάξεις προγραμματιζόμενης λογικής CPLDs και FPGAs, είναι κυκλώματα τα οποία είναι κατάλληλα για ανάπτυξη των εφαρμογών και προτυποποίηση. Το συμπέρασμα αυτό προήλθε από το γεγονός ότι το υλικό των συγκεκριμένων κυκλωμάτων είναι σε θέση να διαμορφώνεται κατάλληλα, με στόχο την ψηφιακή λογική που σχεδιάζει ο χρήστης.

Τέλος, αξίζει να αναφερθεί ότι η VHDL αποτελεί μια πρότυπη γλώσσα η οποία είναι ανεξάρτητη από την τεχνολογία και τον κατασκευαστή ενός κυκλώματος, με άλλα λόγια η περιγραφή της μπορεί να μεταφερθεί και να χρησιμοποιηθεί για οποιαδήποτε τεχνολογία και κατασκευαστή.

Βιβλιογραφία

Brown Stephen & Vranesic Zvonko, 2002, *Σχεδίαση Ψηφιακών Συστημάτων με τη γλώσσα VHDL*, Εκδόσεις Τζιόλα

Pedroni A. Volnei, 2007, *Σχεδιασμός Κυκλωμάτων με τη VHDL,* Εκδόσεις Κλειδάριθμος

Καλόμοιρος Ιωάννης, 2010, *Προηγμένα Ψηφιακά Συστήματα*, Τμήμα Πληροφορικής και Επικοινωνιών, Τομέας Αρχιτεκτονικής Υπολογιστών και Βιομηχανικών Εφαρμογών, Σχολή Τεχνολογικών Εφαρμογών, ΤΕΙ Σερρών

Καλόμοιρος Ιωάννης, 2012, *Εισαγωγή στη γλώσσα VHDL*, Τμήμα Πληροφορικής και Επικοινωνιών, Σχολή Τεχνολογικών Εφαρμογών, ΤΕΙ Σερρών

Καλόμοιρος Ιωάννης, 2014, *Ψηφιακά Κυκλώματα*, Εργαστηριακή Άσκηση 4 Δυαδικός Αθροιστής - Αφαιρέτης

Καραγιώργας Νικόλαος- Τριανταφυλλόπουλος Σταύρος, *Σχεδιασμός Συστημάτων VLSI,* Μεταπτυχιακό Πρόγραμμα ΟΣΥΛ

Κυριάκης - Μπιτζάρος Ευστάθιος, *Σχεδίαση Ψηφιακών Συστημάτων*, Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε, ΑΕΙ Πειραιά

Μιχαηλίδου Ευθυμία - Μιχαηλίδου Χριστίνα, 2010, *Σχεδίαση Εφαρμογών Ψηφιακών Συστημάτων με τη γλώσσα VHDL,* Τμήμα Βιομηχανικής Πληροφορικής, Σχολή Τεχνολογικών Εφαρμογών, ΤΕΙ Καβάλας

Ντούσκα Ελπίδα, 2014, *Υλοποίηση ενός μικροεπεξεργαστή με VHDL κώδικα*, Τμήμα Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών, ΤΕΙ Ηπείρου

Ξαρχάκος Γ. Πέτρος, 2007, *Σχεδίαση Κυκλωμάτων VLSI για χαμηλή κατανάλωση ισχύος*, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Υπολογιστών και Πληροφορικής, Εθνικό Μετσόβιο Πολυτεχνείο

Πρίσκας Θεόδωρος, 2012, *Σχεδίαση ενός 8-bit μικροεπεξεργαστή (του μP 8085) σε VHDL και υλοποίηση σε FPGAs*, Ειδική Επιστημονική Εργασία, Πανεπιστήμιο Πατρών, Τμήμα Φυσικής, Πάτρα

Ροκαδάκη Φρειδερίκη, 2014, *Σχεδίαση Ψηφιακών Κυκλωμάτων ενός πλήρους αθροιστή και ενός αποκωδικοποιητή 2 σε 4 με τη χρήση του Quartus II,* Εργασία μαθήματος Τεχνικού Λόγου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε, ΤΕΙ Ηπείρου

Σαράης Μ. Παντελής, 2015, *Ανάπτυξη Πλατφόρμας Ελέγχου Ολοκληρωμένων Κυκλωμάτων Υψηλών Ταχυτήτων σε FPGA*, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής, Εθνικό Μετσόβιο Πολυτεχνείο

Τζότα Γεωργία, 2013, *Υλοποίηση του αλγορίθμου ευθυγράμμισης ακολουθίας Smith-Waterman σε ενσωματωμένη πλατφόρμα FPGA*, Τμήμα Πληροφορικής, Πανεπιστήμιο Πειραιά

Ηλεκτρονικές Πηγές

1) https://el.wikipedia.org/wiki/Γλώσσα_περιγραφής_υλικού

2) https://en.wikipedia.org/wiki/Programmable_Array_Logic

3) https://www.csd.uoc.gr/~hy225/01a/exer05.html

4) http://users.ics.forth.gr/kateveni/225/06a/verilog_basics.pdf

5) http://www.verilog.com/

6) https://en.wikipedia.org/wiki/ModelSim

7) http://www.cadlab.tuc.gr/courses/cad/chap1.pdf

8) http://www.apel.ee.upatras.gr/dic/fpga/fpga_tutorial.pdf

9) https://el.wikipedia.org/wiki/FPGA

10) http://digilab.teipir.gr/site_data/digital/lab/askisi1.pdf

11) http://www.eng.ucy.ac.cy/ece211/Lecture/Quartus_Tutorial.pdf