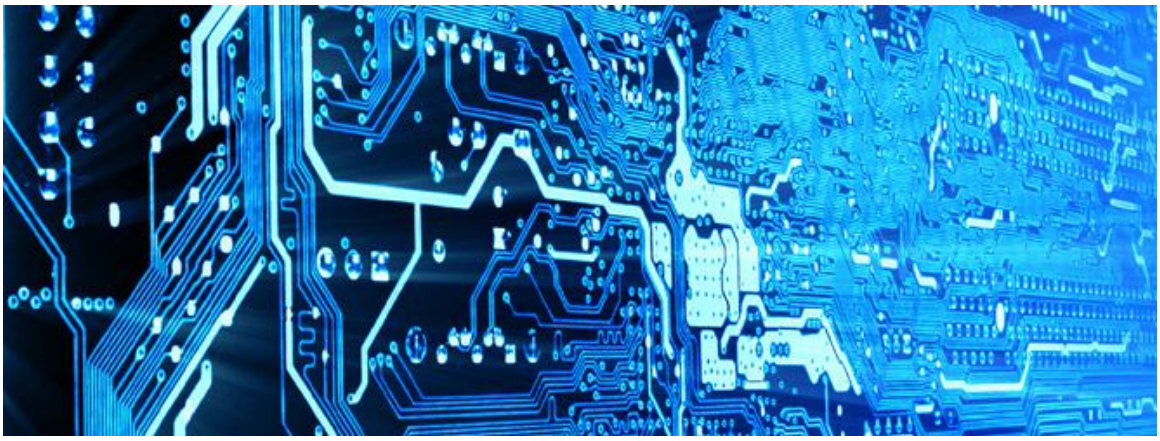
Τ.Ε.Ι. ΗΠΕΙΡΟΥ

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΗΠΕΙΡΟΥ

Τμήμα Τεχνολογίας Πληροφορικής και Τηλεπικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση ενός μικροεπεξεργαστή με VHDL κώδικα



Ντούσκα Ελπίδα

ΑΜ: 10155

Επιβλέπων καθηγητής: Βαρτζιώτης Φώτιος

Άρτα 2014

Περίληψη

Η γλώσσα περιγραφής υλικού VHDL είναι η πλέον διαδεδομένη γλώσσα για την προτυποποίηση και την προσομοίωση ψηφιακών συστημάτων. Η παρούσα πτυχιακή εργασία αποσκοπεί στο να παρέχει έναν εργαστηριακό οδηγό ψηφιακής σχεδίασης κυκλωμάτων με τη συγκεκριμένη γλώσσα με απώτερο σκοπό, την υλοποίηση ενός μικροεπεξεργαστή σε κώδικα VHDL.

Για τις ανάγκες της μελέτης, η εργασία χωρίζεται σε δύο μέρη: το θεωρητικό και το εργαστηριακό-πρακτικό. Το θεωρητικό μέρος αντλείται από την μελέτη βιβλιογραφίας και ηλεκτρονικών πηγών για την απόκτηση του απαραίτητου θεωρητικού υπόβαθρου των προγραμματιζόμενων ψηφιακών κυκλωμάτων (PLD, CPLD, FPGA), της γλώσσας VHDL (τρόπος σύνταξης, δομή, εντολές) καθώς και του λογισμικού Quartus II που αποτελεί ένα από τα πιο γνωστά εργαλεία σχεδίασης και σύνθεσης ψηφιακών κυκλωμάτων.

Το εργαστηριακό μέρος αντλείται από την μελέτη και υλοποίηση ψηφιακών κυκλωμάτων ώστε να είναι και πρακτικά εφικτή η σχεδίαση και προσομοίωση ενός μικροεπεξεργαστή. Παρουσιάζεται μια σειρά εφαρμογών με το αναπτυξιακό κύκλωμα DE2 της εταιρίας Altera, το οποίο βασίζεται στη διάταξη Cyclone II EP2C35F672C6 με αναφορά στις κυριότερες συσκευές εισόδου/εξόδου που περιλαμβάνει. Σχεδιάζονται με τη μορφή εργαστηριακών ασκήσεων μια σειρά ψηφιακών κυκλωμάτων όπως πολυπλέκτες, αποκωδικοποιητές, κωδικοποιητές, αθροιστές, καταχωρητές, απαριθμητές, τα οποία χρησιμοποιούνται ως υποκυκλώματα για να συνθέσουν έναν απλό επεξεργαστή. Ο επεξεργαστής που υλοποιείται έχει διάδρομο δεδομένων 16-bits, περιλαμβάνει οχτώ καταχωρητές, ALU για πρόσθεση και αφαίρεση, κύκλωμα ελέγχου που στηρίζεται σε έναν απαριθμητή και αποκωδικοποιητή και χαρακτηρίζεται από ένα απλό σύνολο τεσσάρων εντολών.

Όλες οι σχεδιάσεις που υλοποιήθηκαν, επιβεβαιώθηκαν σε επίπεδο προσομοίωσης, με το λογισμικό Quartus II v. 9.1.

«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 νομίμων συνεπειών»

Περιεχόμενα

Κατάλογος Πινάκων

Κατάλογος Προγραμμάτων

Κατάλογος Σχημάτων

Κεφάλαιο 1

Προγραμματιζόμενα Ψηφιακά Κυκλώματα

1.1. Προγραμματιζόμενες Λογικές Διατάξεις (PLDs)

Οι Προγραμματιζόμενες Λογικές Διατάξεις (Programmable Logic Devices - PLDs), είναι ψηφιακά κυκλώματα που δημιουργήθηκαν την δεκαετία του 1970 και τα οποία σε αντίθεση με τα μέχρι τότε υπάρχοντα τυπικά ολοκληρωμένα κυκλώματα (standard chips) που εκτελούσαν συγκεκριμένες λειτουργίες, παρείχαν την δυνατότητα της ανάπτυξης και υλοποίησης κυκλωμάτων και συστημάτων από τον ίδιο τον χρήστη, για να τα διαμορφώσει ανάλογα με τις ανάγκες του. Αυτή η ιδέα οδήγησε σε κυκλώματα οργανωμένα σαν πίνακες πυλών, με ενσωματωμένη τη βασική ενσύρματη λογική, ώστε να μην απαιτείται η φάση της μεταλλοποίησης, με δυνατότητα προγραμματισμού των πινάκων για να μπορούν να υλοποιούν οποιαδήποτε ψηφιακή λογική που μπορεί να εκφραστεί με την βοήθεια λογικών συναρτήσεων. Οι διατάξεις αυτές έχουν μια πολύ γενική δομή, που αποτελείται από επαναλαμβανόμενες κυψελίδες πυλών και περιλαμβάνουν προγραμματιζόμενους λογικούς διακόπτες, που επιτρέπουν στα εσωτερικά στοιχεία του ολοκληρωμένου κυκλώματος να οργανώνονται με διάφορους τρόπους. Οι διακόπτες προγραμματίζονται από τον τελικό χρήστη και όχι από τον κατασκευαστή, που σημαίνει ότι επιλέγοντας την κατάλληλη οργάνωση διακοπτών, είναι σε θέση να δημιουργήσει εσωτερικές συνδέσεις και να υλοποιήσει όποιες λειτουργίες επιθυμεί για την εκτέλεση μιας συγκεκριμένης εφαρμογής.

Σχήμα 1 - Συσκευή προγραμματισμού PLDs

Η ιδέα πίσω από τον προγραμματισμό της ψηφιακής λογικής στις προγραμματιζόμενες λογικές διατάξεις είναι η δημιουργία κατάλληλων διασυνδέσεων μέσω ενός προγραμματιζόμενου πίνακα διασυνδέσεων. Οι διασυνδέσεις μπορεί να δημιουργούνται μόνιμα (όπως σε μια προγραμματιζόμενη ROM), μπορεί όμως να είναι και επαναπρογραμματιζόμενες (όπως σε μια EEPROM). Αυτό σημαίνει πως τα περισσότερα PLDs μπορούν να διαγραφούν και να επαναπρογραμματιστούν πολλές φορές. Αυτή η δυνατότητα είναι σημαντική γιατί παρέχει τη δυνατότητα στον σχεδιαστή να κάνει διορθώσεις στο κύκλωμά του, για να διορθώσει τυχόν σχεδιαστικά λάθη ή για να συμπεριλάβει νέες λειτουργίες που δεν είχαν προβλεφθεί αρχικά. Από την άλλη πλευρά, οι διατάξεις αυτές έχουν το μειονέκτημα ότι οι προγραμματιζόμενοι διακόπτες που περιέχουν καταλαμβάνουν πολύτιμο χώρο επάνω στο ολοκληρωμένο κύκλωμα, μειώνοντας την ταχύτητα λειτουργίας των κυκλωμάτων που υλοποιούνται με αυτά.

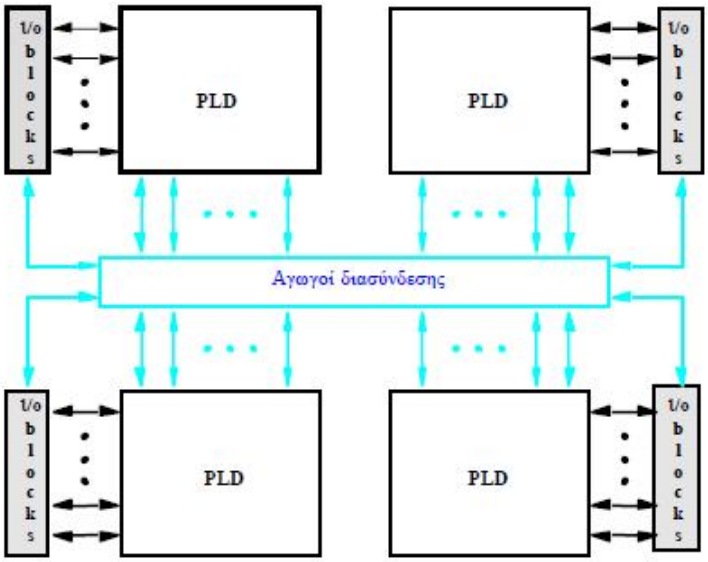
1.2. Σύνθετες Προγραμματιζόμενες Λογικές Διατάξεις (CPLD)

Οι Σύνθετες Προγραμματιζόμενες Λογικές Διατάξεις (Complex Programmable Logic Devices - CPLDs) δημιουργήθηκαν για να καλύψουν την ανάγκη για την υλοποίηση πιο σύνθετων και πιο μεγάλων ψηφιακών σχεδιάσεων σε διαμορφούμενα ολοκληρωμένα κυκλώματα, που απαιτούν περισσότερες εισόδους και εξόδους από ένα απλό PLD και χρειάζοναν πιο σύνθετες προγραμματιζόμενες διατάξεις.

Κάθε CPLD αποτελείται από πολλαπλά αντίγραφα (συλλογή) ενός απλού PLD, τα οποία είναι ενωμένα μεταξύ τους με έναν υπερσύνδεσμο καλωδίων πάνω σε ένα μοναδικό chip. Συνοδεύεται από μία δομή προγραμματιζόμενων διασυνδέσεων και από έναν αριθμό κυκλωμάτων εισόδου/εξόδου. Στο *Σχήμα 2* παρουσιάζεται η γενική δομή ενός CPLD, που αποτελείται από πολλά PLDs, τα οποία είναι συνδεδεμένα με ένα πίνακα καλωδίων, με κάθε PLD να συνδέεται και με έναν αριθμό ακροδεκτών που χρησιμεύουν σαν είσοδοι και έξοδοι του κυκλώματος. Ο αριθμός των ακροδεκτών διαφέρει ανάλογα με το τσιπ και τον κατασκευαστή.

Τα περισσότερα CPLDs περιέχουν προγραμματιζόμενους διακόπτες που μπορούν να προγραμματιστούν με την τοποθέτηση του ολοκληρωμένου κυκλώματος σε μία ειδική μονάδα προγραμματισμού. Η μέθοδος αυτή είναι δύσκολη σε μεγάλα κυκλώματα CPLD, με πάνω από 200 εύθραυστους και εύκαμπτους ακροδέκτες, τα οποία πρέπει να τοποθετηθούν σε κατάλληλη υποδοχή για να πραγματοποιηθεί ο προγραμματισμός.

Οι βάσεις για την αφαίρεση και την επανατοποθέτηση του κυκλώματος στην πλακέτα είναι πολύ ακριβές και κοστίζουν συχνά περισσότερο και από το ίδιο το κύκλωμα. Για τους λόγους αυτούς οι διατάξεις CPLD υποστηρίζουν την τεχνική ISP (In-System Programming). Με αυτόν τον τρόπο το ολοκληρωμένο κύκλωμα δεν χρειάζεται να αφαιρεθεί από την μητρική πλακέτα (PCB), αφού αυτή περιλαμβάνει έναν συνδετήρα (θύρα JTAG – Join Test Action Group) και ένα καλώδιο που συνδέει την πλακέτα με τον υπολογιστή. Μέσω αυτού του καλωδίου μεταφέρονται τα αρχεία προγραμματισμού που έχουν δημιουργηθεί από κάποιο πρόγραμμα σχεδίασης (CAD), από τον υπολογιστή στην πλακέτα και στη συνέχεια στο CPLD.



Σχήμα 2 - Γενική Δομή ενός CPLD

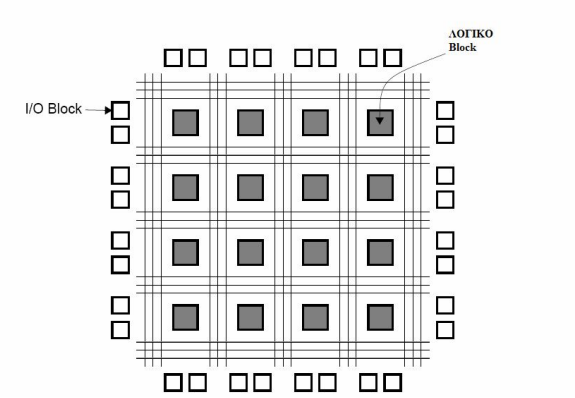
Εφόσον προγραμματιστεί ένα CPLD, διατηρεί μόνιμα την πληροφορία που έχει τοποθετηθεί σε αυτήν, ακόμη και αν διακοπεί η τροφοδοσία του ολοκληρωμένου κυκλώματος. Η διαδικασία αυτή ονομάζεται μη-πτητικός προγραμματισμός.

Τα CPLDs χρησιμοποιούνται για την υλοποίηση πολλών ειδών ψηφιακών κυκλωμάτων και πολύπλοκων σχεδιασμών (όπως ελεγκτές δικτύων και γραφικών). Η απόδοσή τους σε επιφάνεια και ταχύτητα είναι προβλέψιμη εξαιτίας των αρχιτεκτονικών που χρησιμοποιούνται.

1.3. Διατάξεις Πυλών Προγραμματιζόμενων στο Πεδίο (FPGA)

Οι Διατάξεις Πυλών Προγραμματιζόμενων στο Πεδίο (Field Programmable Gate Array - FPGA) αποτελούν μια επέκταση των CPLDs προς μεγαλύτερες πυκνότητες ολοκλήρωσης και δυνατότητα εκτέλεσης πολύ σύνθετης λογικής. Είναι πιο ευέλικτα προγραμματιστικά και επιτρέπουν την ανάπτυξη πιο σύνθετων εφαρμογών.

Τα FPGAs αποτελούνται από λογικές βαθμίδες (blocks) και πόρους διασύνδεσης (*Σχήμα 3*). Η λογική έχει διασπαστεί σε ένα μεγάλο αριθμό προγραμματιζόμενων λογικών δομικών μονάδων, η κάθε μία από τις οποίες είναι μικρότερη από μια διάταξη PLD, οι οποίες είναι κατανεμημένες σε ολόκληρο το τσιπ μέσα σε μια περιοχή προγραμματιζόμενων διασυνδέσεων με ολόκληρο τον πίνακα να περιβάλλεται από προγραμματιζόμενες δομικές μονάδες εισόδου/εξόδου. Η προγραμματιζόμενη λογική δομική μονάδα σε μια διάταξη FPGA έχει λιγότερες δυνατότητες απ' ότι μια τυπική διάταξη PLD, αλλά το τσιπ FPGA περιέχει πολύ μεγαλύτερο αριθμό λογικών δομικών μονάδων σε σχέση με τον αριθμό των PLDs που περιέχει μια διάταξη CPLD.



Σχήμα 3 - Γενική Δομή ενός FPGA

Τα FPGAs είναι ψηφιακά ολοκληρωμένα κυκλώματα τα οποία περιέχουν προγραμματιζόμενα μπλοκ ψηφιακής λογικής. Η διαμόρφωση του FPGA γίνεται μέσω προγραμματισμού του από τον χρήστη, για την υλοποίηση σχεδόν οποιασδήποτε ψηφιακής λειτουργίας. Πολλά FPGAs παράγονται με δυνατότητα να λειτουργήσουν σε διάφορες συχνότητες, ανάλογα με την επιτρεπόμενη ταχύτητα διάδοσης του σήματος (speed grade). Αυτά ξεχωρίζουν από το γράμμα 'A' στο τέλος της ονομασίας τους, και έναν αριθμό που χαρακτηρίζει το πόσο αργό ή γρήγορο είναι ένα FPGA (π.χ. η διάταξη EPF10K70A-2 είναι πιο αργή από την EPF10K70A-1).

1.4. Cyclone II FPGAs

Η οικογένεια κυκλωμάτων Cyclone II ανήκει στην ομάδα των FPGAs και κατασκευάζονται από την Altera. Είναι κυκλώματα βασισμένα σε τεχνολογία 90–nm και βασίζονται σε μνήμη, με μέγιστη χωρητικότητα τα 68.000 λογικά στοιχεία (LEs). Περιέχουν μέχρι και 1.1 Mbits ενσωματωμένης RAM. Αποτελούν την συνέχεια της πολύ επιτυχημένης οικογένειας Cyclone της Altera, επεκτείνοντας το πλήθος των λογικών στοιχείων, της ενσωματωμένης μνήμης και τους χρήσιμους ακροδέκτες I/O. Τα κυκλώματα αυτά προσφέρουν υψηλή ταχύτητα και χαμηλό κόστος και επιτρέπουν την σχεδίαση σύνθετων συστημάτων. Αυτό που κάνει τα Cyclone II να ξεχωρίζουν από τα άλλα FPGAs είναι το γεγονός ότι παρέχουν μεγάλες χωρητικότητες χωρίς να αυξάνουν το μέγεθος του κυκλώματος ή την κατανάλωση ισχύος, διασφαλίζοντας ταυτόχρονα μείωση του κόστους. Το χαμηλό κόστος κατασκευής και οι αναβαθμισμένες δυνατότητες του Cyclone II FPGA το καθιστούν ιδιανική λύση για μια ευρεία γκάμα εφαρμογών. Στον Πίνακα 1 παρουσιάζεται ολόκληρη η οικογένεια Cyclone II της Altera. Η Altera προσφέρει επίσης χαμηλού κόστους σειριακές συσκευές διαμόρφωσης για να διαμορφώνουν διατάξεις Cyclone II.

Device	LAB Columns	LAB Rows	LEs	PLLs	M4K Memory Blocks	Embedded Multiplier Blocks
EP2C5	24	13	4,608	2	26	13
EP2C8	30	18	8,256	2	36	18
EP2C20	46	26	18,752	4	52	26
EP2C35	60	35	33,216	4	105	35
EP2C50	74	43	50,528	4	129	86
EP2C70	86	50	68,416	4	250	150

Πίνακας 1 - Οικογένεια Cyclone II FPGA της Altera

Η αρχιτεκτονική της οικογένειας Cyclone II είναι οργανωμένη σε δύο διαστάσεις, υλοποιώντας την ψηφιακή λογική βασιζόμενη σε στήλες (columns) και γραμμές (rows) από λογικά στοιχεία. Οι διασυνδέσεις στηλών και γραμμών διάφορων ταχυτήτων διασυνδέουν μεταξύ τους τις λογικές δομές (LABs), τις ενσωματωμένες δομές μνήμης και τους ενσωματωμένους πολλαπλασιαστές. Κάθε λογική δομή (LAB) αποτελείται από 16 λογικά στοιχεία (LEs) τα οποία αποτελούν μια μικρή μονάδα λογικής που επιτρέπει την υλοποίηση λογικών συναρτήσεων του χρήστη. Όλες οι λογικές δομές είναι ομαδοποιημένες σε στήλες και γραμμές σε όλη την επιφάνεια του ολοκληρωμένου.

Οι διατάξεις Cyclone II περιέχουν από 4,608 μέχρι 68,416 LEs. Επιπλέον παρέχουν ένα κεντρικό δίκτυο μονάδων χρονισμού και PLLs. Το δίκτυο χρονισμού αποτελείται από 16 γραμμές ρολογιού, οι οποίες οδηγούν ολόκληρο το FPGA, για όλους τους πόρους μιας διάταξης, όπως τα στοιχεία εισόδου/εξόδου (IOEs), λογικά στοιχεία (LEs), ενσωματωμένους πολλαπλασιαστές και δομές μνήμης. Τα PLLs παρέχουν γενικού σκοπού χρονισμό, όπως και εξωτερικές εξόδους για υποστήριξη I/O υψηλής ταχύτητας. Τα M4K είναι δομές μνήμης διπλής εισόδου με συνολική χωρητικότητα 4K-bits, παρέχουν εύρος λέξης μέχρι 36 bits, συχνότητες λειτουργίας μέχρι 260 MHz, και είναι τοποθετημένες σε στήλες κατά μήκος του ολοκληρωμένου μεταξύ συγκεκριμένων LABs. Οι συσκευές Cyclone II προσφέρουν από 119 εώς 1,152 Kbits ενσωματωμένης μνήμης.

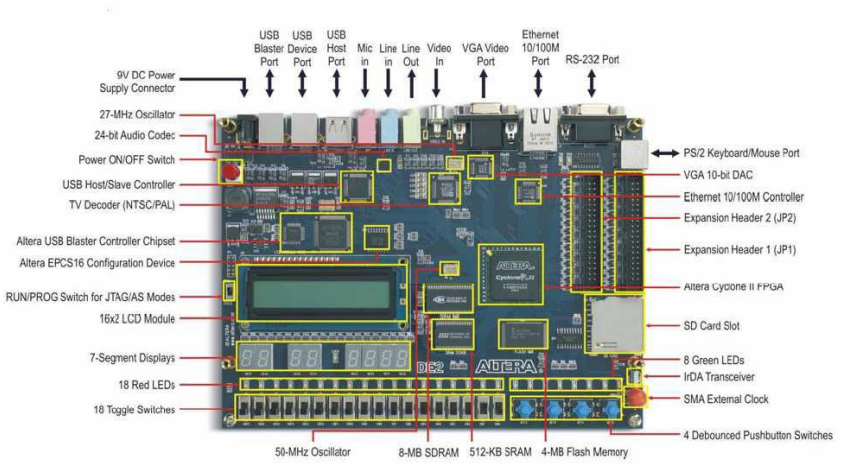
Οι ενσωματωμένοι πολλαπλασιαστές μπορούν να χρησιμοποιηθούν είτε ως δύο 9x9 bit πολλαπλασιαστές, είτε ως ένας 18x18 bit πολλαπλασιαστής, με ταχύτητα έως και 250 MHz. Κάθε ακροδέκτης εισόδου/εξόδου μιας διάταξης Cyclone II τροφοδοτείται από ένα IOE με μέγιστο ρυθμό μετάδοσης 805 Mbps για εισόδους και 640 Mbps για εξόδους. Κάθε IOE περιέχει ένα αμφίδρομο I/O buffer και 3 καταχωρητές για είσοδο καταχωρητών, έξοδο και ακροδέκτες εξόδου. Ορισμένοι ακροδέκτες παρέχουν διασύνδεση με εξωτερικές συσκευές μνήμης όπως DDR, DDR2, SDR, SDRAM, QDRII SRAM, στα 167 MHz. Ο αριθμός των δομών μνήμης M4K, των ενσωματωμένων δομών DSP επεξεργασίας, όπως οι πολλαπλασιαστές και των PLLs, ποικίλουν από διάταξη σε διάταξη.

1.4.1. Cyclone II 2C35 FPGA

Το Cyclone II FPGA EP2C35 είναι ένα ολοκληρωμένο που ενσωματώνεται στην αναπτυξιακή πλατφόρμα DE2 της Altera. Περιέχει 33.216 λογικά στοιχεία (LEs), 35 ενσωματωμένους πολλαπλασιαστές, 4 PLLs και 475 ακίδες εισόδου/εξόδου. Διαθέτει 105 M4K RAM blocks και 483.840 συνολικά RAM bits. Έχει υλοποιηθεί σε wafer 300 nm χρησιμοποιώντας την τεχνολογία TSMC’s 90 nm.

1.5. Το αναπτυξιακό κύκλωμα DE2

Η αναπτυξιακή πλακέτα DE2 (*Σχήμα 4*) είναι κατασκευασμένη από την εταιρεία Altera και διαθέτει υποσυστήματα, μνήμες και μια σειρά περιφερειακών για διάφορες εφαρμογές (π.χ. ήχου, εικόνας, μετάδοσης δεδομένων). Προσφέρει ένα μεγάλο αριθμό χαρακτηριστικών τα οποία παρέχουν πολλές δυνατότητες στον χρήστη για το σχεδιασμό απλών μέχρι και και πολύπλοκων κυκλωμάτων. Για ορισμένες λειτουργίες η πλακέτα χρειάζεται σύνδεση με τον υπολογιστή.



Σχήμα 4 - Η Αναπτυξιακή πλακέτα DE2

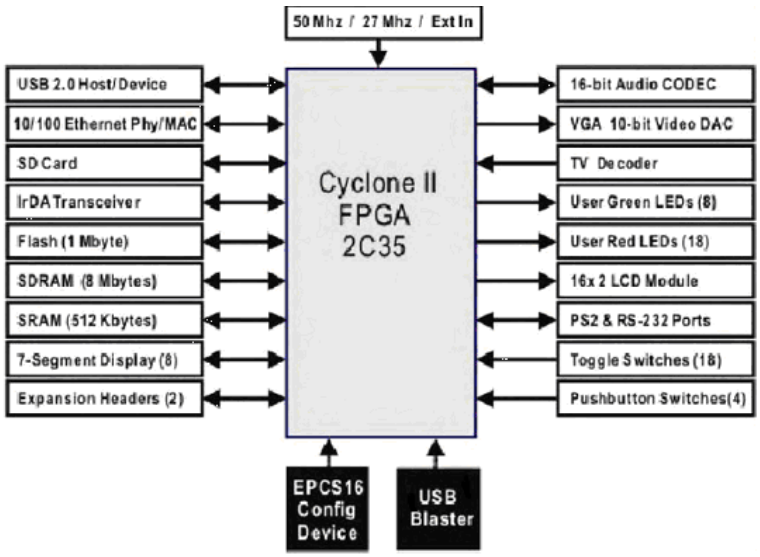
Η αναπτυξιακή πλακέτα περιλαμβάνει τα εξής εξαρτήματα:

- Το Altera Cyclone II 2C35 FPGA chip
- Τη σειριακή συσκευή διαμόρφωσης EPCS16 της Altera
- Το USB Blaster Port (on board) για προγραμματισμό και API έλεγχο από τον χρήστη (υποστηρίζονται JTAG και Active Serial τύποι προγραμματισμού)
- 512 Kbyte SRAM μνήμη
- 8 Mbyte SDRAM μνήμη
- 4 Mbyte Flash μνήμη
- Υποδοχή καρτών (card socket) SD
- Τέσσερις διακόπτες πλήκτρα (push buttons)
- Δεκαοχτώ toggle διακόπτες (switches)
- Δεκαοχτώ κόκκινα LEDs
- Εννέα πράσινα LEDs

- Δύο κρυστάλλους (oscillators) στα 50MHz και 27MHz oscillator για πηγές clock
- Ένα Audio 24-bits CODEC με line-in, line-out και μικρόφωνα
- Ένα VGA video DAC υψηλής ταχύτητας 10-bits
- Μία οθόνη LCD 16 χαρακτήρων και 2 γραμμών
- Ένα TV Decorder (NTSC/PAL) με TV είσοδο
- Ένα 10/100 Ethernet Controller με σύνδεσμο
- Ένα USB Host/Slave Controller με USB τύπου A και τύπου B συνδέσμους
- Ένα RS232 πομποδέκτη 9 ακίδων (pins)
- Διασύνδεση ποντικιού/πληκτρολογίου τύπου PS/2
- Ένα πομπό υπέρυθρων IrDA
- Δύο 40-pin κεφαλές επέκτασης (expansion headers) με προστασία διόδου

Εκτός από αυτά τα χαρακτηριστικά η DE2 πλακέτα υποστηρίζει πίνακα ελέγχου για πρόσβαση στα διαφορετικά εξαρτήματα και λογισμικό (FPGACore Library) για τις τυπικές διασυνδέσεις εισόδου/εξόδου (I/O) το οποίο παρέχει επιπλέον μια ποικιλία από δυνατότητες ώστε να εκμεταλλευτεί ο χρήστης τα χαρακτηριστικά της κάρτας DE2. Η χρήση της αναπτυξιακής πλακέτας απαιτεί εξοικίωση με το πρόγραμμα QuartusII.

Στο *Σχήμα 5* παρατίθεται το δομικό διάγραμμα της DE2 πλακέτας. Για παροχή μεγαλύτερης ευελιξίας προς τον χρήστη, όλες οι συνδέσεις πραγματοποιούνται μέσω του Cyclone II FPGA.



Σχήμα 5 - Το δομικό διάγραμμα της πλακέτας DE2

1.6. Ενσωματωμένοι επεξεργαστές λογισμικού

Ένας ενσωματωμένος επεξεργαστής είναι ένας επεξεργαστής κρυμμένος σε μία συσκευή, μαζί και με άλλα ηλεκτρονικά ή ηλεκτρομηχανικά μέρη, σχεδιασμένος για να κάνει μια ή και περισσότερες λειτουργίες πραγματικού χρόνου. Προγραμματίζεται σε ορισμένες λειτουργίες, διαθέτει δηλαδή τμήμα λογισμικού, αν και οι βασικές του λειτουργίες σχετίζονται με το υλικό μέρος της αρχιτεκτονικής του, στον ρόλο και του ελεγκτή και του επεξεργαστή. Οι ενσωματωμένοι επεξεργαστές είναι καλύτεροι από άποψη απόδοσης σε σχέση με έναν απλό ελεγκτή, αλλά δεν έχουν φτάσει ακόμα στο επίπεδο των γενικής χρήσης επεξεργαστών. Είναι σχεδιασμένοι να είναι μικροί, λειτουργούν με χαμηλή κατανάλωση ενέργειας (χαμηλό κόστος), είναι συνήθως RISC επεξεργαστές και περιλαμβάνουν πολλούς καταχωρητές και μνήμη. Εντούτοις, οι επεξεργαστές που αφιερώνονται στα ενσωματωμένα συστήματα, λειτουργούν σε χαμηλές συχνότητες, κάτι που έχει αρνητική επίπτωση στην απόδοσή τους. Επίσης, όσο αυξάνεται η ικανότητα αποθήκευσης τους τόσο μειώνεται και ταχύτητά τους. Ωστόσο, αυτά τα μειονεκτήματα μπορούν να εξαλειφθούν με κατάλληλη οργάνωση της αρχιτεκτονικής των μικροεπεξεργαστών.

Οι ενσωματωμένοι επεξεργαστές εφαρμόζονται σήμερα εκτενώς σε διάφορες συσκευές ήχου, εικόνας, σε κάμερες, σε ηλεκτρονικά παιχνίδια, σε PDAs, σε υπολογιστές τσέπης, σε περιφερειακά συστήματα γενικής χρήσης όπως modem, κάρτες video, στα τηλέφωνα και στα δίκτυα.

Η ανάπτυξη των επεξεργαστών λογισμικού οφείλεται στην συνεχιζόμενη αύξηση της απόδοσης των μικροεπεξεργαστών, την εύκολη διαμόρφωση συστημάτων, καθώς και την μείωση της ανάπτυξης συστημάτων υλικού (ASICs), καθώς προσφέρουν χαμηλό κόστος και ευελιξία. Η απόδοσή τους μπορεί να αυξηθεί με ταχύτερο ρολόι, με πολλαπλές ALUs, ώστε να εκτελούνται πιο γρήγορα οι εντολές, και με κατάλληλες διοχετεύσεις. Κατασκευάζονται από ομάδες προγραμματιστών, είναι γραμμένοι με τη μορφή κώδικα περιγραφής υλικού (HDL), υλοποιούνται με διαμόρφωση του FPGA, και προγραμματίζονται από τον χρήστη, γράφοντας την κατάλληλη εφαρμογή (application) σε μια γλώσσα προγραμματισμού όπως η C ή assembly. Η εφαρμογή αποθηκεύεται στη μνήμη που προγράμματος που διαθέτουν και η οποία μπορεί να συνδεθεί με τα M4K blocks που υπάρχουν στις σύγχρονες διατάξεις FPGA.

Κάθε μικροεπεξεργαστής τέτοιου είδους έχει ένα σχετικά μικρό σύνολο εντολών, που εκτελούν λειτουργίες όπως αριθμητικές πράξεις και μεταφορά δεδομένων σε καταχωρητές και μνήμη και συναντώνται σε ενσωματωμένα συστήματα.

Η ενσωμάτωση ενός επεξεργαστή σε ένα FPGA παρέχει το πλεονέκτημα της δυνατότητας δημιουργίας ενός μικροϋπολογιστικού «συστήματος» πάνω στο ολοκληρωμένο κύκλωμα, καθώς μπορεί να συνεργαστεί με ελεγκτές περιφερειακών συσκευών, που σχεδιάζονται από τον χρήστη ή διατίθενται με τη μορφή ανοιχτού κώδικα σε γλώσσα περιγραφής υλικού, αυξάνοντας δραματικά τις δυνατότητες του συστήματος. Με τη χρήση των πόρων ενός γενικής χρήσης FPGA μπορεί να υλοποιηθεί εσωτερική μνήμη, δίαυλοι επεξεργαστών και εξωτερικοί περιφερειακοί ελεγκτές. Η χρήση διαύλων, μνημών, ελεγκτών, περιφερειακών και περιφερειακών ελεγκτών, κάνει το σύστημα πιο χρήσιμο, πιο φθηνό και πιο αποδοτικό.

Στα μειονεκτήματα συγκαταλέγονται η πολυπλοκότητα στη δημιουργία του συστήματος καθώς και οι ατέλειες των εργαλείων σχεδίασης, που δεν ανταποκρίνονται πάντα στις απαιτήσεις. Σε περίπτωση που απομένουν στο FPGA αχρησιμοποίητοι πόροι, τότε το κόστος χρήσης του αυξάνεται. Η ανάγκη σύνδεσης εξωτερικής μνήμης στο FPGA και η προσπέλασή της από τον ενσωματωμένο επεξεργαστή με την χρήση ελεγκτών μνήμης μειώνει την αποδοτικότητα.

Η Altera και η Xilinx είναι δύο εταιρίες που ασχολούνται με τους FPGA ενσωματωμένους επεξεργαστές. Σημειώνεται ότι ορισμένες οικογένειες FPGA ενσωματώνουν έτοιμους επεξεργαστές υλικού (hard processors) σε κάποιο τμήμα τους, αντί να τους υλοποιούν συνθέτοντας γλώσσα περιγραφής υλικού.

1.7. Εισαγωγή στα προγράμματα σχεδίασης

Η αυξανόμενη έμφαση τις λεπτομέρειες των εφαρμογών ωθεί συνεχώς την σχεδίαση και την κατασκευή ψηφιακών συστημάτων στην πολυπλοκότητα. Με την βοήθεια του υπολογιστή και με εργαλεία CAD παρέχεται η δυνατότητα ευκολότερης σχεδίασης των επιμέρους στοιχείων του ψηφιακού συστήματος, με αποτέλεσμα η τεχνολογία σχεδίασης συστημάτων με την βοήθεια υπολογιστών να αναπτύσσεται ραγδαία τα τελευταία χρόνια. Αυτό συμβαίνει εξαιτίας τις συνεχόμενης αύξησης των απαιτήσεων των εφαρμογών και την δυνατότητα των υπολογιστών να ανταποκριθούν σε αυτές τις απαιτήσεις.

Τα πλεονεκτήματα, καθώς και οι λόγοι που ώθησαν τις αυτή την κατεύθυνση είναι:

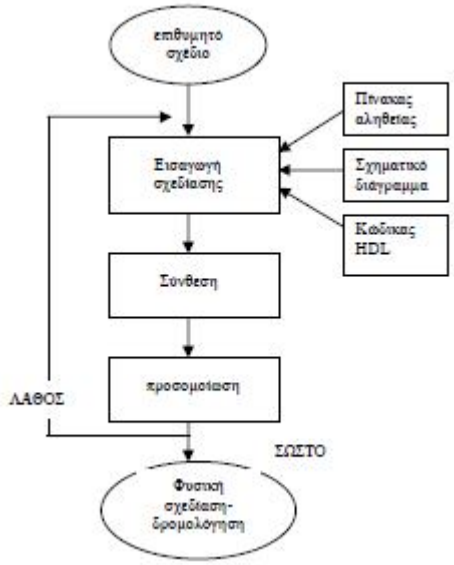
- Η ανάγκη μείωσης του χρόνου σχεδίασης τις VLSI κυκλώματος
- Η ανάγκη υλοποίησης εκ νέου τις κυκλώματος που ήδη υπάρχει
- Χρήση λογισμικού φιλικού τις το χρήστη
- Ανάλυση και εντοπισμός δυσλειτουργιών στην παραγωγική διαδικασία
- Προσιτή τιμή προϊόντων, προτείνοντας το κατάλληλο κατά περίπτωση λογισμικό
- Προσαρμοστικότητα των προϊόντων τις ανάγκες των πελατών

Η τυπική ροή σχεδίασης ξεκινάει με την εισαγωγή σχεδίασης. Η συνάρτηση θα περιγράφεται είτε από ένα πίνακα αληθείας, είτε με την μέθοδο σχηματικών διαγραμμάτων, είτε από μια γλώσσα περιγραφής υλικού.

Η πιο ευρέως διαδεδομένη μορφή σχεδίασης CAD είναι το εργαλείο σχηματικού διαγράμματος. Το εργαλείο χρησιμοποιεί τις δυνατότητες γραφικής απεικόνισης του υπολογιστή και ένα ποντίκι για να επιτρέψει στον χρήστη τη σχεδίασει τις σχηματικού διαγράμματος. Για να επιτρέψει την εισαγωγή πυλών στο διάγραμμα, το εργαλείο παρέχει τις βιβλιοθήκες, δηλαδή συλλογές από γραφικά σύμβολα που παριστάνουν λογικές πύλες διαφόρων τύπων. Οι πύλες εισάγονται από την βιβλιοθήκη και συνδέονται με τέτοιο τρόπο ώστε να δημιουργούν λογικό κύκλωμα. Οποιαδήποτε υποκυκλώματα είχαν δημιουργηθεί στο παρελθόν μπορούν να συμπεριληφθούν ως γραφικά σύμβολα, δημιουργώντας έτσι κυκλώματα που στο εσωτερικό τις περιέχουν άλλα μικρότερα. Αυτό αποτελεί την λεγόμενη ιεραρχική σχεδίαση.

Επιπλέον, παρέχεται η δυνατότητα εισαγωγής αρχείου σε γλώσσα περιγραφής υλικού, η οποία μοιάζει με μια ανώτερη γλώσσα προγραμματισμού, με την διαφορά ότι αυτή περιγράφει κυκλώματα και σημαντικό πλεονέκτημα την μεταφερσιμότητα τις σχεδίασης. Η εισαγωγή σχεδίασης τις λογικού κυκλώματος πραγματοποιείται γράφοντας κώδικα. Στην αγορά υπάρχουν τις γλώσσες περιγραφής υλικού (Hardware Description Languages – HDLs), ωστόσο δύο μορφές αποτελούν πρότυπα του ινστιτούτου IEEE: η VHDL και η Verilog HDL.

Τα βασικά βήματα σχεδιάσης με τη χρήση εργαλείων CAD παρουσιάζονται στο *Σχήμα 6*. Εκτός από την εισαγωγή του κυκλώματος, τα επόμενα βήματα περιλαμβάνουν τη σύνθεση (synthesis), την βελτιστοποίηση και την φυσική σχεδίαση ή δρομολόγηση (place and route).



Σχήμα 6 - Ροή εργασιών σε ένα σύστημα σχεδίασης

Η σύνθεση είναι μια αυτόματη διαδικασία που επιτελεί το σχεδιαστικό λογισμικό και περιλαμβάνει και τις διαδικασίες, τις την μετάφραση. Στόχος τις σύνθεσης είναι ο μετασχηματισμός του σχεδίου του χρήστη ώστε να παράγει ένα βέλτιστο κύκλωμα, κατάλληλο για την τεχνολογία τις διάταξης στην οποία στοχεύει ο χρήστης. Στο σημείο αυτό το σύστημα αντιπροσωπεύεται από ένα σύνολο κατάλληλα προγραμματισμένων καταχωρητών, που υλοποιούν την ψηφιακή σχεδίαση στο επίπεδο τις μνήμης διαμόρφωσης. Αφότου γίνει η αρχική σύνθεση είναι δυνατή η επαλήθευση τις σωστής λειτουργίας του σχεδιασμένου κυκλώματος με την βοήθεια τις προσομοίωσης λειτουργίας.

Η προσομοίωση υλοποιείται με τη βοήθεια κατάλληλων εισόδων, που ενεργούν τις ακροδέκτες εισόδου του κυκλώματος, οπότε τα παραγόμενα σήματα εξόδου μπορούν να συγκριθούν με τα αναμενόμενα. Τα σήματα εισόδου σχεδιάζονται ως κατάλληλες κυματομορφές, με τη βοήθεια ειδικών εργαλείων σχεδίασης σημάτων. Τα σχετικά αρχεία που δημιουργούνται για την ενεργοποίηση των εισόδων, καλούνται “waveform vectors” ή διανύσματα κυματομορφών.

Αν η προσομοίωση δεν είναι ικανοποιητική, θα χρειαστεί επανασχεδιασμός του κυκλώματος. Όποια λάθη εντοπιστούν μπορούν να διορθωθούν επιστρέφοντας στο στάδιο

εισαγωγής σχεδίασης. Όταν τελικά η προσομοίωση είναι μέσα στα επιθυμητά πλαίσια και εφόσον δεν διαπιστώνεται η ύπαρξη σφαλμάτων, τότε προχωρούμε στο επόμενο βήμα, που είναι η δρομολόγηση του κυκλώματος (place and route ή fitting), η οποία σε μεγάλα κυκλώματα είναι πολύωρη διαδικασία. Στο στάδιο αυτό δημιουργούνται οι απαραίτητες συνδέσεις στον πίνακα διασυνδέσεων του κυκλώματος (interconnection matrix), με τις οποίες συνδέονται με βέλτιστο τρόπο μεταξύ τις τα λογικά στοιχεία που επιτελούν τις επιμέρους λειτουργίες του κυκλώματος.

Ορισμένα λογισμικά σχεδίασης επιτρέπουν στο τέλος τον προγραμματισμό του κυκλώματος, ώστε το σχέδιο να αποτυπωθεί στο ολοκληρωμένο κύκλωμα για το οποίο προορίζεται, διαμορφώνοντάς το κατάλληλα.

Μερικά από τα πιο γνωστά εργαλεία σχεδίασης προσφέρουν οι εταιρείες Altera (QuartusII), Xilinx (ISE) και Synopsys (TCAD). Τα λογισμικά αυτά είναι εύχρηστα τόσο για νέους όσο και για έμπειρους χρήστες, γρήγορα και παρέχουν σχεδιαστική λογική που στοχεύει σε FPGAs, χρήση ενσωματωμένων επεξεργαστών και DSP κυκλωμάτων.

Κεφάλαιο 2

Η γλώσσα VHDL

2.1. Εισαγωγή στη VHDL

Η VHDL είναι μία γλώσσα περιγραφής υλικού, που χρησιμοποιείται για την ανάπτυξη ολοκληρωμένων ψηφιακών ηλεκτρονικών κυκλωμάτων και συστημάτων, αν και αρχικά η δημιουργία της αποσκοπούσε απλώς στην περιγραφή και στην προσομοίωση κυκλωμάτων. Ο όρος VHDL προκύπτει από τις λέξεις VHSIC Hardware Description Language, όπου VHSIC σημαίνει Very High Speed Integrated Circuit. Η γλώσσα VHDL ξεκίνησε το 1981 ως έργο (project) του Υπουργείου Άμυνας των ΗΠΑ, με σκοπό τον κοινό τρόπο τεκμηρίωσης (documentation) της δομής και της λειτουργίας όλων των ψηφιακών κυκλωμάτων που διαχειριζόταν το εν λόγω υπουργείο. Το 1987 η VHDL έγινε πρότυπο του IEEE (IEEE STD 1076-1987) και από τότε αναθεωρήθηκε και επεκτάθηκε αρκετές φορές. Αργότερα, το 1993, δημιουργήθηκε μια βελτιωμένη έκδοση της, η IEEE 1164.

Η VHDL είναι πλέον αποδεκτή σαν μια από τις πιο σημαντικές τυποποιημένες γλώσσες για τον καθορισμό, την επιβεβαίωση και την σχεδίαση ηλεκτρονικών κυκλωμάτων. Σήμερα, ένας αριθμός από εταιρείες υψηλής τεχνολογίας χρησιμοποιούν αποκλειστικά την γλώσσα VHDL για την περιγραφή ψηφιακών κυκλωμάτων. Πολλά πανεπιστήμια επίσης προσφέρουν τη δυνατότητα εκμάθησης της γλώσσας VHDL στους φοιτητές και υπάρχουν διάφορες ομάδες για την υποστήριξη νέων χρηστών. Στα επόμενα χρόνια, μικρού και μεσαίου μεγέθους εταιρείες θα αρχίσουν να χρησιμοποιούν επίσης την VHDL. Η VHDL και τα ASIC (Application Specific Integrated Circuits) έχουν ήδη αρχίσει να ανταγωνίζονται τους single-chip ελεγκτές. Η καθιέρωση της ως πρότυπο, διαβεβαιώνει ότι και οι επόμενες εκδόσεις εργαλείων σχεδίασης θα υποστηρίζουν το πρότυπο αυτό. Έτσι, ένα κύκλωμα που περιγράφηκε και αναπτύχθηκε με τα σημερινά εργαλεία σχεδίασης θα είναι μεταφέρσιμο μελλοντικά σε νέα εργαλεία σχεδίασης, με ελάχιστες ή καθόλου αλλαγές.

Δύο σημαντικοί λόγοι για την χρησιμοποίηση της VHDL αντί της χρήσης παραδοσιακών τεχνικών σχηματικής σχεδίασης κυκλωμάτων, είναι οι μικρότεροι χρόνοι ανάπτυξης της ηλεκτρονικής σχεδίασης και η δυνατότητα απλούστερης υποστήριξης.

Όλοι οι σημαντικότεροι κατασκευαστές υποστηρίζουν το πρότυπο της VHDL. Η VHDL είναι πλέον μια τυποποιημένη γλώσσα, με το πλεονέκτημα της εύκολης μεταφοράς του κώδικά της, μεταξύ διαφορετικών εμπορικών πλατφορμών. Αυτό σημαίνει ότι ο κώδικας σε VHDL για τον εξομοιωτή μιας συγκεκριμένης εταιρείας μπορεί να μεταφερθεί σε αυτούς άλλων εταιρειών χωρίς να χρειάζεται να αλλάξει.

Τα τελευταία χρόνια έχουν αναπτυχθεί καλά εργαλεία για την VHDL, και ειδικότερα εξομοιωτές VHDL, για τη χρήση τους με ηλεκτρονικούς υπολογιστές, με αποτέλεσμα οι τιμές να μειωθούν δραματικά επιτρέποντας σε μικρότερες εταιρείες να χρησιμοποιήσουν την VHDL. Υπάρχουν επίσης εργαλεία σύνθεσης, κυρίως για FPGAs και PLDs, αλλά η λειτουργικότητά τους είναι ελαφρά πιο περιορισμένη από αυτή των workstation εργαλείων.

Μια νέα προοπτική, η οποία έχει φέρει αλλαγή στον τομέα των σχεδιαστών ηλεκτρονικών κυκλωμάτων, είναι ο τρόπος με τον οποίο αρκετές χιλιάδες λογικές πύλες και flip-flops μπορούν να προγραμματιστούν σαν ένα ολοκληρωμένο κύκλωμα μέσα σε μόνο μερικά λεπτά με ένα μόνο PC χωρίς την ανάγκη ακριβού εξοπλισμού. Αυτό σημαίνει ότι είναι δυνατόν να γεννηθεί ένα αρχείο αυτόματα από VHDL για τον προγραμματισμό κυκλωμάτων. Αυτή η μέθοδος ονομάζεται γρήγορη προτυποποίηση (rapid prototyping).

Με τη συγγραφή κώδικα σε γλώσσα VHDL, μπορούν να υπερκεραστούν βασικές μονάδες (όπως πύλες) για τον σχεδιασμό κυκλωμάτων. Δουλεύοντας με την VHDL δεν γίνεται απλώς συγγραφή κώδικα, αλλά παρέχονται και δυνατότητες υλοποίησης ιεραρχικών δομών και σχεδιασμού, με τη χρήση βιβλιοθήκης συστατικών (components). Η VHDL είναι επίσης καλή για το γράψιμο κώδικα για τυποποιημένα κυκλώματα, όπως το ολοκληρωμένο της Motorola 680020. Υπάρχουν επίσης εταιρείες που πουλάνε standard components για την εξομοίωση, επιτρέποντας την επιβεβαίωση μοντέλων ολόκληρων circuit boards με standard components. Εάν ένα ολόκληρο κύκλωμα εξομοιωθεί, οι χρόνοι πρόσβασης και οι άκυρες διευθύνσεις (invalid addresses), για παράδειγμα, μπορούν να ανιχνευτούν με ένα πολύ δυνατό τρόπο.

Ο σχεδιασμός με τη γλώσσα VHDL παρέχει τη δυνατότητα τόσο της συγγραφής κώδικα, αλλά και της επιβεβαίωσης της λειτουργικότητάς του με τη χρήση της εξομοίωσης. Η σύνθεση μπορεί να συγκριθεί με έναν συμβολομεταφραστή, ο οποίος μεταφράζει τον κώδικα σε κώδικα μηχανής. Στο υλικό (hardware) ο κώδικας μεταφράζεται σε ένα σχηματικό με πύλες και flip-flops.

Σήμερα η VHDL χρησιμοποιείται όχι μόνο για τεκμηρίωση, αλλά επιπρόσθετα, για προσομοίωση (simulation) και λογική σύνθεση (logic synthesis). Ως πρότυπο του IEEE, προσφέρει δυνατότητες μοντελοποίησης της συμπεριφοράς ψηφιακών κυκλωμάτων. Έτσι η περιγραφή σε VHDL μπορεί να χρησιμοποιηθεί ως είσοδο σε προγράμματα λογισμικού τόσο για την προσομοίωση της λειτουργίας ψηφιακών κυκλωμάτων, όσο και για την σύνθεσή τους.

2.2. Στοιχεία της γλώσσας VHDL

Η VHDL υποστηρίζει το περιβάλλον ανάπτυξης (development environment) για την ψηφιακή εξέλιξη, καθώς και διάφορες μεθόδους εξέλιξης, όπως top-down, bottom-up, ή συνδυασμό και των δύο. Δεδομένου ότι αρκετά από τα σημερινά ηλεκτρονικά προϊόντα έχουν χρόνο ζωής πάνω από 10 χρόνια, απαιτείται να επανασχεδιάζονται αρκετές φορές έτσι ώστε να εκμεταλλευθούν την νέα τεχνολογία, καθώς τα ηλεκτρονικά κυκλώματα έχουν τροποποιηθεί και νέες συναρτήσεις έχουν προστεθεί.. Ο απλούστερος τρόπος για να γίνει αυτό είναι να χρησιμοποιηθεί τεχνολογικά ανεξάρτητη σχεδίαση σε VHDL, που σημαίνει ότι είναι δυνατόν να αλλάξει η τεχνολογία χρησιμοποιώντας αυτόματα εργαλεία. Η VHDL υποστηρίζει την μετατρεψιμότητα (modifiability) γιατί ως γλώσσα είναι εύκολο να διαβαστεί, είναι ιεραρχική και όπως είναι δομημένη.

Η γλώσσα υποστηρίζει ιεραρχίες (block διαγράμματα), επαναχρησιμοποιούμενα συστατικά, διαχείριση λαθών και επιβεβαίωση. Οι ιεραρχίες περιγράφονται χρησιμοποιώντας δομική VHDL, διαδικασίες (procedures) και συναρτήσεις (functions). Η δομική VHDL μπορεί να συγκριθεί με ένα block διάγραμμα. Πολλά συστήματα υποστηρίζουν γραφική είσοδο που μπορεί να μεταφραστεί αυτόματα σε δομική VHDL. Η γλώσσα όπως υποστηρίζει ταυτόχρονες (concurrent) και ακολουθιακές (sequential) δομές, καθώς και τα πάντα από προδιαγραφές μέχρι περιγραφή πυλών.

Στην περίπτωση όπως παραδοσιακής σχηματικής σχεδίασης, ο σχεδιαστής πρέπει να βρίσκεται σε συνεχή επαφή με συγκεκριμένους τεχνολογικούς παράγοντες όπως ο συγχρονισμός, η ικανότητα οδήγησης, η επιλογή του συστατικού και το fan-out. Ένα από τα μεγαλύτερα πλεονεκτήματα στην σχεδίαση με την VHDL είναι ότι ο σχεδιαστής μπορεί να επικεντρωθεί στην συνάρτηση, όπως η υλοποίηση των απαιτούμενων προδιαγραφών, και δεν χρειάζεται να αφιερώσει χρόνο και ενέργεια σε συγκεκριμένους τεχνολογικούς παράγοντες που δεν επηρεάζουν τη συνάρτηση.

Η VHDL είναι τυποποιημένη και αυτό κάνει δυνατή την μετακίνηση του κώδικα από διαφορετικά συστήματα ανάπτυξης για μοντελοποίηση. Για την σχεδίαση είναι δυσκολότερη η μετακίνηση του κώδικα γιατί δεν υπάρχει κανένα πρότυπο. Η VHDL δεν έχει ακόμα τυποποιηθεί για τα αναλογικά ηλεκτρονικά. Παρόλα αυτά η εργασία για την τυποποίηση είναι σε εξέλιξη στην VHDL με μία αναλογική επέκταση AHDL, που επιτρέπει και την περιγραφή αναλογικών κυκλωμάτων. Το νέο πρότυπο θα βασίζεται εξολοκλήρου στο VHDL πρότυπο και θα έχει έναν αριθμό προσθηκών για την περιγραφή αναλογικών συναρτήσεων.

2.3. Στάδια ανάπτυξης

Η φάση της ανάπτυξης για ένα προϊόν είναι από τα πρώτα στάδια για τον κύκλο ζωής του, καθώς κατά την διάρκειά της το προϊόν καθορίζεται, σχεδιάζεται και επιβεβαιώνεται. Η ροή της εξέλιξης από τις προδιαγραφές μέχρι το πρωτότυπο μπορεί να χωριστεί σε τέσσερα στάδια:

Ανάλυση

Το στάδιο της ανάλυσης αποτελείται από το γράψιμο των προδιαγραφών, οι οποίες μπορούν να γραφούν σε VHDL ή σε μια συνηθισμένη γλώσσα. Ο σκοπός των προδιαγραφών είναι η κατανόηση του αντικειμενικού στόχου του σχεδίου. Οι προδιαγραφές μπορούν να περιγραφούν σε VHDL και να επιβεβαιωθούν με έναν εξομοιωτή σε VHDL.

Σχεδίαση

Στο στάδιο της σχεδίασης μετασχηματίζονται οι προδιαγραφές σε μία αρχιτεκτονική και σε κώδικα σε VHDL. Το στάδιο αρχίζει με τον καθορισμό της αρχιτεκτονικής (block διάγραμμα) και τη συγγραφή του κώδικα σε VHDL για τα διάφορα συστατικά (blocks) ή την αντιγραφή έτοιμων συστατικών από μια βιβλιοθήκη. Η λειτουργία της σχεδίασης επιβεβαιώνεται με έναν εξομοιωτή. Όταν το αποτέλεσμα συμφωνεί με τις προδιαγραφές, ο σχεδιαστής μπορεί να προχωρήσει στο επόμενο στάδιο, που περιλαμβάνει την σχεδίαση της αρχιτεκτονικής και των συστατικών.

Απεικόνιση στην τεχνολογία (technology mapping)

Το στάδιο αυτό είναι σε μεγάλο βαθμό αυτοματοποιημένο και αφορά παραμέτρους, όπως η τιμή, η απόδοση, η τροφοδοσία και τα λοιπά που καθορίζουν ποια τεχνολογία θα επιλεγεί. Οι χρονικοί περιορισμοί περιγράφονται σε ένα πρότυπο που μπορεί να διαβαστεί από το εργαλείο της σύνθεσης. Εάν το εργαλείο της σύνθεσης δεν μπορεί να συναντήσει τους χρονικούς περιορισμούς, το στάδιο της σχεδίασης πρέπει να επαναληφθεί ολόκληρο ή κατά ένα μέρος. Μία εγκεκριμένη σύνθεση παράγει ένα τεχνολογικά εξαρτώμενο σχηματικό διάγραμμα, το οποίο είναι ένα αρχείο εισόδου σε άλλα εργαλεία. Ο χρόνος που χρειάζεται για να προγραμματιστεί ένα ολοκληρωμένο κύκλωμα FPGA, είναι περίπου 5 λεπτά.. Εάν, για παράδειγμα, περιλαμβάνεται ένα gate array, θα πάρει κάποιες εβδομάδες πριν ένα τελειωμένο κύκλωμα να τροφοδοτηθεί. Εάν χρησιμοποιείται ένα gate array, διανύσματα ελέγχου πρέπει να περιληφθούν στην σχεδίαση. Η εργασία κατά την διάρκεια αυτής της φάσης είναι βασικά η παραγωγή μιας κατασκευαστικής βάσης.

Επικύρωση

Στο στάδιο αυτό κατασκευάζεται ένα πρωτότυπο και συγκρίνεται με τις προδιαγραφές. Εάν το αποτέλεσμα είναι το ίδιο με τις προδιαγραφές, το κύκλωμα είναι έτοιμο.

2.4. Λογική σύνθεση

Η λογική σύνθεση γίνεται με τη βοήθεια λογισμικού, το οποίο ονομάζεται λογισμικό σύνθεσης και αποτελεί αναπόσπαστο κομμάτι των σημερινών προγραμμάτων ψηφιακής σχεδίασης. Έτσι ο σχεδιαστής αποφεύγει την εργασία της μετάφρασης, της ελαχιστοποίησης, και της συνάντησης των χρονικών περιορισμών από τον κώδικα σε VHDL. Η σύνθεση καθορίζεται από τις ακόλουθες διαφορετικές τάξεις:

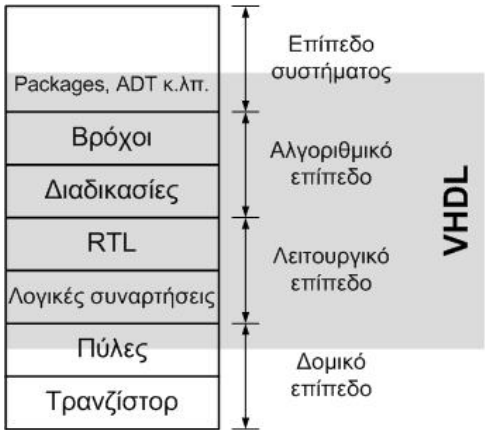
- *Logic synthesis* - Μεταφράζει και ελαχιστοποιεί τις boolean συναρτήσεις μόνο μέσα σε πύλες
- *RTL (Register Transfer Level) synthesis* - Είναι ίδια σαν την logic synthesis αλλά επιπλέον μεταφράζει ακολουθιακές δηλώσεις της γλώσσας σε πύλες και flip-flops (state machines)
- *Behavioural synthesis* - Μπορεί να ξαναχρησιμοποιήσει ένα hardware component για περισσότερη από μία παράλληλη ακολουθιακή δήλωση γλώσσας

Ο σχεδιαστής αρχικά περιγράφει ένα ψηφιακό κύκλωμα με την βοήθεια της γλώσσας VHDL, που αντιστοιχεί σε ένα ή περισσότερα αρχεία απλού κειμένου (plain text), τα οποία έχουν συνήθως την κατάληξη .vhd, αντί για την συνήθη .txt. Επειδή ο πηγαίος κώδικας των περισσότερων γλωσσών προγραμματισμού που αφορούν λογισμικό γράφεται επίσης σε αρχεία απλού κειμένου (με αντίστοιχες καταλήξεις), η περιγραφή ενός ψηφιακού κυκλώματος σε VHDL συχνά αναφέρεται και ως "κώδικας VHDL" (VHDL source code). Όταν ο σχεδιαστής προχωρήσει στην σύνθεση της περιγραφής του, το λογισμικό σύνθεσης θα ελέγξει την περιγραφή για συντακτικά λάθη από τον VHDL compiler (HDL Compilation). Στην περίπτωση συντακτικών λαθών το λογισμικό σύνθεσης τερματίζει ανεπιτυχώς. Τα συντακτικά λάθη υποδεικνύονται και δίνονται συμβουλές για την διόρθωσή τους. Εφόσον δεν υπάρχουν συντακτικά λάθη, η περιγραφή σε VHDL υποβάλλεται στην διαδικασία σύνθεσης (HDL Synthesis) και το αποτέλεσμα είναι ένα αρχείο το οποίο καλείται netlist, το οποίο περιλαμβάνει βελτιστοποιημένες (optimized) λογικές εκφράσεις, οι οποίες αναπαριστούν σε επίπεδο υλικού την δομή και την λειτουργία του ψηφιακού κυκλώματος που περιγράφηκε σε VHDL.

Η VHDL δεν είναι γλώσσα προγραμματισμού λογισμικού, αλλά γλώσσα περιγραφής υλικού. Αυτό σημαίνει ότι υπάρχουν περιγραφές σε VHDL, οι οποίες παρόλο που είναι συντακτικά και λογικά σωστές, εντούτοις δεν αντιστοιχούν σε κάποιο συνδυασμό υπαρκτών ψηφιακών μονάδων, με αποτέλεσμα το λογισμικό σύνθεσης να αδυνατεί να συνθέσει ένα ψηφιακό κύκλωμα, το οποίο να λειτουργεί σύμφωνα με την περιγραφή σε VHDL.

2.5. Επίπεδα αφαίρεσης

Τα επίπεδα αφαίρεσης αποτελούν έννοιες που σχετίζονται με την απόκρυψη των λεπτομερειών. Ένα ψηφιακό κύκλωμα μπορεί να περιγραφεί σε διάφορα επίπεδα αφαίρεσης και η VHDL είναι μια γλώσσα πλούσια σε αφαιρέσεις (abstractions), καθώς μπορεί να περιγράψει ολόκληρα συστήματα μέχρι το επίπεδο των αυτόνομων λογικών πυλών (*Σχήμα 7*). Τα επίπεδα αφαίρεσης που υποστηρίζονται πλήρως από την γλώσσα είναι το λειτουργικό και το αλγοριθμικό, ενώ σε μικρότερο βαθμό υποστηρίζονται το δομικό και το επίπεδο συστήματος.



Σχήμα 7 - Επίπεδα αφαίρεσης VHDL

Στο δομικό επίπεδο η περιγραφή ενός ψηφιακού κυκλώματος αποτελείται μόνο από πύλες (καθόλου πρακτική όταν σχεδιάζονται μεγάλα και πολύπλοκα ψηφιακά κυκλώματα). Το λειτουργικό επίπεδο περιλαμβάνει περιγραφές απλών λογικών συναρτήσεων και RTL περιγραφές που αφορούν καταχωρητές που συνδέονται μέσω τμημάτων συνδυαστικής λογικής. Το αλγοριθμικό επίπεδο περιλαμβάνει αφηρημένες δομές, όπως βρόχους, διαδικασίες και αλγοριθμικά στοιχεία, όπως εξισώσεις με αθροίσματα και πολλαπλασιασμούς. Το επίπεδο συστήματος αφορά συγκεκριμένες δομές της VHDL, οι οποίες επιτρέπουν περιγραφές αυτού του είδους.

Η μάθηση της VHDL για μικρές σχεδιάσεις είναι εύκολη, αλλά χρειάζεται σημαντικά μεγαλύτερη γνώση για να χρησιμοποιηθεί η γλώσσα για πολύπλοκες σχεδιάσεις.

2.6. Εξομοίωση

Η εξομοίωση μοντέλων αποτελεί έναν αποτελεσματικό τρόπο για την επιβεβαίωση της σχεδίασης. Το μοντέλο στον υπολογιστή είναι μόνο ένα μοντέλο διακριτού χρόνου, παρόλα αυτά η πραγματικότητα είναι διακριτή. Το υπολογιστικό μοντέλο είναι λιγότερο κοντά στη πραγματικότητα στο υψηλό επίπεδο αφαίρεσης και πιο κοντά στην πραγματικότητα στο χαμηλότερο επίπεδο.

Ο υπολογιστής διαθέτει διάφορες μεθόδους τελειοποίησης που είναι καλύτερες από ένα φυσικό μοντέλο. Θέτοντας την χειρότερη περίπτωση (worst case) για την επεξεργασία των παραμέτρων και το εύρος των θερμοκρασιών παρέχεται καλύτερη επιβεβαίωση από το να κατασκευαστεί ένα πρωτότυπο. Με ένα πρωτότυπο, μόνο αυτό το ίδιο επιβεβαιώνεται.

Σε ένα στατικό αναλυτή χρόνου ή σε ένα εξομοιωτή είναι δυνατόν να επιβεβαιωθούν - εξομοιωθούν διαφορετικές χρονικές περιπτώσεις:

- Χειρότερη περίπτωση (Worst case): Χαμηλότερη τάση (π.χ. 4.5 V), υψηλότερη θερμοκρασία (π.χ. 125^0C) και βραδύτερα χαρακτηριστικά επεξεργασίας
- Κανονική περίπτωση (Typical case): Κανονική τάση (π.χ. 5.0 V), κανονική θερμοκρασία (π.χ. 25^0C) και κανονικά χαρακτηριστικά επεξεργασίας
- Καλύτερη περίπτωση (Best case): Υψηλότερη τάση (π.χ. 5.5 V), χαμηλότερη θερμοκρασία (π.χ. -55^0C) και γρηγορότερα χαρακτηριστικά επεξεργασίας

Το πρόβλημα των εξομοιώσεων, η απαίτηση δηλαδή όλο και περισσότερου χρόνου όσο η ακρίβεια των μοντέλων αυξάνει, καθιστά αδύνατη την εξομοίωση μεγάλων σχεδιαστικών κυκλωμάτων στο δομικό επίπεδο. Ο χρόνος της εξομοίωσης αυξάνεται επίσης με την περιγραφή διαφόρων ελέγχων, όπως έλεγχοι set-up, έλεγχοι hold και έλεγχοι spikes. Ο χρονισμός αναλύεται πιο αποτελεσματικά και πιο γρήγορα όταν χρησιμοποιείται ένας αναλυτής χρόνου στο λογικό επίπεδο.

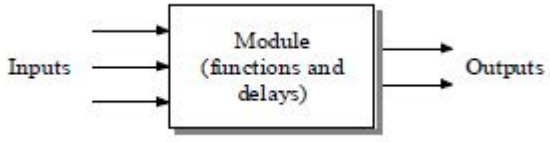
Υπάρχουν πλέον εξομοιωτές μικτού επιπέδου, που μπορούν να χειριστούν μοντέλα σε διαφορετικά επίπεδα αφαίρεσης και σε διαφορετικές γλώσσες περιγραφής υλικού. Το πλεονέκτημα αυτών των εξομοιωτών είναι ότι η σχεδίαση μπορεί να μοντελοποιηθεί σε ένα χαμηλό επίπεδο (gate delay, load delay και καθυστέρηση διασύνδεσης) και τα γειτονικά κυκλώματα σε ένα υψηλό επίπεδο (VHDL behavioral επίπεδο, testbenches) για να μειωθεί ο χρόνος της εξομοίωσης.

2.7. Δομικά στοιχεία σχεδίασης

Με την VHDL περιγράφεται ο τρόπος διασύνδεσης ενός ψηφιακού κυκλώματος με το εξωτερικό περιβάλλον, καθώς και η εσωτερική υλοποίηση του. Η VHDL περιγραφή της σχεδίασης ενός κυκλώματος αποτελείται από *οντότητες (entities)* και *αρχιτεκτονικές (architectures)*. Σε κάθε οντότητα αντιστοιχεί τουλάχιστον μία αρχιτεκτονική. Με την γλώσσα VHDL μπορούμε να περιγράψουμε από απλά κυκλώματα, όπως μια πύλη (π.χ. nand), έως πολύπλοκα ολοκληρωμένα ψηφιακά κυκλώματα. Στα ολοκληρωμένα ψηφιακά κυκλώματα υπάρχει η δυνατότητα χρήσης όχι μίας, αλλά πολλών οντοτήτων και πολλών αρχιτεκτονικών, καθώς επίσης είναι δυνατό να περιέχεται μία οντότητα μέσα σε μια άλλη, δηλαδή να δημιουργήσουμε με ιεραρχικό τρόπο ένα κύκλωμα, χρησιμοποιώντας υποκυκλώματα. Στην περίπτωση αυτή, όπου η μια οντότητα περιέχεται σε άλλη ή άλλες ως υποκύκλωμα, η οντότητα λέγεται *συστατικό (component)*.

Οι εντολές στη γλώσσα VHDL μπορούν να γραφούν είτε με κεφαλαία, είτε με μικρά, είτε πάλι ξεκινώντας το πρώτο γράμμα με κεφαλαίο και συνεχίζοντας με μικρά, έτσι ώστε οι λέξεις π.χ. flip και FLIPt να αναγνωρίζονται σαν μια ίδια λέξη. Αυτό δεν έχει καμία σημασία για το τι καταλαβαίνει ο μεταφραστής (compiler). Η υπογράμμιση όμως των χαρακτήρων στους αναγνωριστές είναι σημαντική, έτσι ώστε οι λέξεις This_Lab και ThisLab να θεωρούνται διαφορετικές. Επίσης δεν έχουν σημασία τα κενά για την ερμηνεία του κώδικα από τον compiler. Αυτό δεν σημαίνει βέβαια ότι δεν πρέπει να αφήνονται κενά για να ξεχωρίζουν μεταξύ τους οι εντολές, αλλά ότι δεν έχει σημασία το πόσους χαρακτήρες καταλαμβάνει ένα κενό. Επίσης, αν χρειάζεται να γραφούν σχόλια στον κώδικα πρέπει να εισαχθούν με διπλή παύλα (--), στην γραμμή που έχει επιλεγεί. Ο compiles βλέποντας αυτούς τους δύο χαρακτήρες αγνοεί ό,τι υπάρχει στη γραμμή μετά από αυτούς.

Το βασικό δομικό στοιχείο που περιγράφεται στην VHDL είναι μία module (*Σχήμα 8*) που χαρακτηρίζεται από τις εξόδους, τις εισόδους, τη συνάρτηση και τις καθυστερήσεις.



Σχήμα 8 - Δομή μιας module

2.7.1. Συστατικά (Components)

Τα συστατικά είναι μια κεντρική ιδέα στην VHDL και χρησιμοποιούνται, μεταξύ άλλων, για την κατασκευή βιβλιοθηκών από συστατικά (components libraries), όπως μικροεπεξεργαστές, ειδικά κυκλώματα και άλλα στάνταρντ κυκλώματα. Είναι μικρότερα κυκλώματα, σε VHDL, που έχουν ήδη περιγραφεί και μπορούν να κληθούν ως αρχεία βιβλιοθήκης, αφού επιτρέπουν την αντιγραφή τους όσες φορές και αν χρειαστεί, δηλαδή είναι επαναχρησιμοποιούμενα. Τέτοια αρχεία μπορεί να είναι έτοιμα αρχεία από κάποιες υπάρχουσες βιβλιοθήκες, ή μπορεί να τα σχεδιάσει ο ίδιος ο χρήστης.

Στην γλώσσα της επιστήμης των υπολογιστών, αυτή η αντιγραφή ονομάζεται δημιουργία στιγμιότυπων (instances) ενός συστατικού, δηλαδή δημιουργία ενός συστατικού σε ένα σχηματικό ή σε ένα text file. Η VHDL είναι μια object-based γλώσσα, δηλαδή δεν έχει κληρονομικότητα (inheritance). Τα generic components και οι instantiations είναι τυπικά χαρακτηριστικά object-based γλωσσών.

Η εσωτερική δομή ενός συστατικού μπορεί να αποκρυβεί από τον σχεδιαστή και αυτό ονομάζεται ως η αρχή του μαύρου κουτιού (black box principle). Σε μερικές περιπτώσεις δεν υπάρχει καμία απολύτως ανάγκη για την γνώση του πως είναι κατασκευασμένο το εσωτερικό ενός συστατικού. Ο σχεδιαστής συνήθως ενδιαφέρεται μόνο για τις εισόδους και τις εξόδους, τον καθορισμό της λειτουργίας και για τους χρόνους πρόσβασης. Ένα συστατικό σε μια βιβλιοθήκη μπορεί να αποτελείται από άλλα συστατικά. Είναι δυνατόν μια βιβλιοθήκη να περιέχει μεγάλα, σύνθετα συστατικά, όπως επεξεργαστές, κυκλώματα επικοινωνιών κ.τ.λ.

Τα κύρια χαρακτηριστικά της σχεδίασης με υποκυκλώματα είναι η δήλωση των συστατικών που περιέχονται στο κύκλωμα, καθώς και η περιγραφή του τρόπου με τον οποίο συνδέονται μεταξύ τους. Με την δήλωση του κάθε συστατικού καθορίζεται το όνομά του καθώς και τα ονόματα των εισόδων και των εξόδων του.

Η γενική μορφή μιας τέτοιας δήλωσης παρουσιάζεται παρακάτω.

COMPONENT *όνομα συστατικού*
[**GENERIC** (*όνομα παραμέτρου* : integer := τιμή{ ;
όνομα παραμέτρου : integer := τιμή }) ;]
PORT (*όνομα ακροδέκτη* : **mode** τύπος ;
όνομα ακροδέκτη : **mode** τύπος) ;
END COMPONENT ;

Στο *Πρόγραμμα 1*, παρουσιάζεται ένα παράδειγμα το οποίο δηλώνει ένα συστατικό μιας μνήμης (μόνο ανάγνωσης) με βάθος διευθυνσιοδότησης και πλάτος δεδομένων, τα οποία εξαρτώνται από σταθερές generic. Η εσωτερική δομή του κυκλώματος έχει αποκρυβεί, παρουσιάζοντας μόνο τα σήματα εξόδου και εισόδου του κυκλώματος.

```
COMPONENT read_only_memory
  GENERIC (data_bits, addr_bits : POSITIVE);
    PORT (en    : IN BIT ;
          addr  : IN BIT_VECTOR (depth-1 DOWNTO 0) ;
          data  : OUT BIT_VECTOR (WIDTH-1 DOWNTO 0 );
END COMPONENT ;
```

Πρόγραμμα 1 - Δήλωση Component

Η δήλωση ενός συστατικού μπορεί να γίνει ή στην περιοχή δήλωσης της αρχιτεκτονικής ή σε μία δήλωση πακέτου (package) Στη συνέχεια, στο κύριο σώμα της αρχιτεκτονικής, πρέπει να δημιουργηθούν στιγμιότυπα για τα συστατικά που έχουν δηλωθεί, τα οποία λειτουργούν σαν υποκυκλώματα της σχεδίασης. Η δημιουργία των στιγμιότυπων γίνεται πάντα μετά τη δήλωση ενός συστατικού και είναι της μορφής:

όνομα στιγμιότυπου: όνομα συνιστώσας **PORT MAP** *(ονόματα σημάτων)*

2.7.2. Οντότητα (Entity)

Η οντότητα μπορεί να θεωρηθεί σαν ένα μαύρο κουτί με εισόδους και εξόδους. Ένας μικροεπεξεργαστής, για παράδειγμα, έχει μία οντότητα που αποτελείται από τον δίαυλο δεδομένων, τον δίαυλο διευθύνσεων, τον δίαυλο ελέγχου και άλλα σήματα.

Η δήλωση της οντότητας περιγράφει την διασύνδεση που μπορεί να έχει ένα λειτουργικό κομμάτι ενός κυκλώματος με το εξωτερικό περιβάλλον, γι' αυτό ακολουθείται από μια περιγραφή των εισόδων και εξόδων του κυκλώματος η οποία περιέχεται μέσα σε μια δήλωση port. Στην δήλωση της οντότητας μπορούν να προστεθούν παράμετροι, ώστε κάθε φορά που χρησιμοποιείται σαν συστατικό ενός άλλου κυκλώματος να καθορίζονται οι τιμές των παραμέτρων. Η δήλωση της οντότητας είναι το πρώτο κομμάτι κώδικα που συναντάμε σε μία VHDL περιγραφή και σε αυτό δηλώνουμε καταρχήν το όνομα της οντότητας, καθώς και τα σήματα εισόδου/εξόδου του κυκλώματος, δηλαδή τους ακροδέκτες του κυκλώματος, οι οποίοι ενώνουν το κύκλωμα με το εξωτερικό περιβάλλον.

Η διεπαφή του κυκλώματος ξεκινάει με τη δεσμευμένη λέξη ENTITY και περιέχει τις εισόδους και τις εξόδους τους κυκλώματος. Μπορούν επίσης να συμπεριληφθούν άλλα εξωτερικά χαρακτηριστικά, όπως ο χρόνος ή εξαρτήσεις θερμοκρασίας.

Συντακτικά, μια οντότητα έχει την εξής μορφή:

ENTITY *όνομα οντότητας* **IS**
PORT (*όνομα ακροδέκτη* : **mode** *τύπος* ;
……
όνομα ακροδέκτη : **mode** *τύπος*) ;
END *όνομα οντότητας* ;

Το όνομα της κάθε οντότητας θα πρέπει να είναι ένα έγκυρο όνομα της γλώσσας VHDL, το οποίο συνήθως επιλέγεται με τέτοιο τρόπο ώστε να αποδίδει νόημα στο κύκλωμα που περιγράφει. Οι έσοδοι και έξοδοι του κυκλώματος καθορίζονται από τη λέξη PORT (θύρα) και το mode (κατάσταση). Στο mode θέτουμε το τι είδους ακροδέκτη έχουμε, δηλαδή αν είναι είσοδος, έξοδος, κτλ. Οι τιμές που μπορούμε να του δώσουμε είναι:

- IN: είσοδος
- OUT: έξοδος που δεν διαβάζεται εσωτερικά
- BUFFER: έξοδος που διαβάζεται και εσωτερικά
- INOUT: είσοδος/έξοδος

Αν ένας ακροδέκτης δεν ορίζεται, δηλαδή δεν έχει δηλωθεί αν είναι είσοδος ή έξοδος, θεωρείται αυθαίρετα ως είσοδος. Στο πεδίο όπου δηλώνουμε τον τύπο του κάθε ακροδέκτη έχουμε τις εξής περιπτώσεις:

- BIT / BIT_VECTOR
- STD_LOGIC / STD_LOGIC_VECTOR
- STD_ULOGIC / STD_ULOGIC_VECTOR
- BOOLEAN / INTEGER / REAL / CHARACTER
- TIME

Στο *Πρόγραμμα 2*, παρουσιάζεται μια δήλωση η οποία καθορίζει μια οντότητα με όνομα adder, που έχει δύο εισόδους (x, y) και μία έξοδο (z), όλες τους τύπου STD_LOGIC. Οι είσοδοι αναγνωρίζονται από την λέξη-κωδικό IN και οι έξοδοι από τον λέξη-κωδικό OUT.

```
ENTITY adder IS
    PORT (x, y  : IN STD_LOGIC ;
          z     : OUT STD_LOGIC) ;
END adder
```

Πρόγραμμα 2 - Δήλωση Entity

2.7.3. Αρχιτεκτονική (Architecture)

Σε αυτό το τμήμα του κώδικα περιγράφεται η εσωτερική δομή του κυκλώματος, δηλαδή οι σχέσεις μεταξύ των εισόδων, έτσι ώστε να πάρουμε κάποιες τιμές στην έξοδο. Η αρχιτεκτονική, μπορεί να είναι ένας δομικός κώδικας VHDL, δηλαδή να αποτελείται και από άλλα συστατικά, όπως π.χ. αριθμητική και λογική μονάδα (ALU), καταχωρητές γενικής χρήσεως και καταχωρητές κατάστασης.

Η δήλωση της αρχιτεκτονικής χρησιμοποιείται για να περιγράψει την σχέση (τη συνάρτηση) μεταξύ εισόδων και εξόδων σε μια οντότητα ή αλλιώς περιγράφει την λειτουργία του υλικού. Σε αντίθεση με την δήλωση της οντότητας η οποία πρέπει να είναι μοναδική είναι δυνατόν να κατασκευαστούν περισσότερα του ενός σώματα αρχιτεκτονικής συνδεδεμένα με την ίδια οντότητα. Έτσι σε μια συνιστώσα ενός κυκλώματος μπορούν να αντιστοιχηθούν περισσότερες της μιας περιγραφές που μπορεί να οφείλονται στα διαφορετικά επίπεδα προσέγγισης και λεπτομέρειας στην περιγραφή, είτε στην κατασκευή της συνιστώσας με βάση διαφορετικές τεχνολογίες σε υλικό.

Υπάρχουν τρεις τύποι σωμάτων αρχιτεκτονικής (architecture bodies):

- *Αρχιτεκτονική Dataflow*: Η περιγραφή της module γίνεται βάση της ροής των σημάτων μέσα από αυτή και με τη χρήση των κατάλληλων λογικών και αριθμητικών τελεστών
- *Αρχιτεκτονική Behavioral*: Η περιγραφή της συνάρτησης ολόκληρης της module γίνεται σαν μια ολότητα, με τη χρήση υψηλού επιπέδου εκφράσεων και γλωσσικών δομών. Η συμπεριφορά της module περιγράφεται με μία *process* της οποίας το σώμα αποτελείται από ένα σετ δηλώσεων.

- *Αρχιτεκτονική Structural*: Η module περιγράφεται σαν η διασύνδεση απλούστερων modules (ιεραρχική περιγραφή) ενώ οι απλούστερες modules περιγράφονται ανεξάρτητα. Η περιγραφή της συνάρτησης των υψηλότερων επιπέδων modules καθορίζεται από τη συμπεριφορά των απλούστερων modules και της διασύνδεσής τους.

Δύο ονόματα περιγράφονται σε μια δήλωση αρχιτεκτονικής: το συστατικό σε VHDL που περιγράφει σε ποια οντότητα ανήκει η αρχιτεκτονική και το όνομα της αρχιτεκτονικής. Στην περιοχή δήλωσης, δηλώνονται σήματα, τύποι, σταθερές, συνιστώσες και χαρακτηριστικά. Η περιοχή αυτή αρχίζει με την δήλωση του ονόματος της αρχιτεκτονικής και τελειώνει στο BEGIN. Στο κύριο σώμα της αρχιτεκτονικής (architecture body) καθορίζεται η λειτουργία του κυκλώματος και αρχίζει μετά το BEGIN και τελειώνει με την λέξη END.

Η γενική μορφή της αρχιτεκτονικής ενός κυκλώματος είναι η εξής:

ARCHITECTURE *όνομα αρχιτεκτονικής* **OF** *όνομα οντότητας* **IS**
    Signal ;
    Constant ;
    Type ;
    Component ;
    Attribute ;
BEGIN
    Εντολές ;
END *όνομα αρχιτεκτονικής* ;

Στο *Πρόγραμμα 3*, παρουσιάζεται ένας απλός ημιαθροιστής δύο bits. Το τμήμα της οντότητας, που τον περιγράφει ως block, φυσικά προηγείται.

```
ENTITY half_adder IS
    PORT (x, y  : IN STD_LOGIC ;
          a, b  : OUT STD_LOGIC) ;
END half_adder

ARCHITECTURE behavior OF half_adder IS
BEGIN
    a <= x OR y ;
    b = x AND y ;
END behavior ;
```

Πρόγραμμα 3 - Δήλωση Architecture

2.7.4. Πακέτα (Packages)

Τα πακέτα λειτουργούν κατά κάποιο τρόπο σαν αποθήκη προγραμμάτων και δηλώσεων, όπως παράδειγμα σταθερές, δηλώσεις τύπων, υποπρογράμματα, οντότητες, αρχιτεκτονικές και άλλα. Μέσα στα πακέτα μπορούν να συμπεριληφθούν και ορίσματα τύπων. Με την δημιουργία πακέτων επιτυγχάνεται η ομαδοποίηση μιας περιγραφής κυκλώματος καθώς και σχετικών δηλώσεων, που εξυπηρετούν ένα κοινό σκοπό. Ένα πακέτο μπορεί να αποτελείται από ένα σύνολο δηλώσεων, που μοντελοποιούν έναν σχεδιασμό, και από ένα σύνολο υποκυκλωμάτων (components).

Οι δηλώσεις του κυρίου μέρους των υποπρογραμμάτων, οντότητες και αρχιτεκτονικές μπορούν να δηλωθούν είτε μέσα στο πακέτο στο σώμα του πακέτου είτε εξωτερικά σε ένα άλλο αρχείο το οποίο και καλείται όταν χρειαστεί. Τα πακέτα αποτελούν ξεχωριστά αρχεία και αναγκαστικά πρέπει να ανήκουν σε μια βιβλιοθήκη. Συνολικά ένα πακέτο μπορεί να περιέχει ορισμούς τύπων (types) και υποτύπων (subtypes).

- Ορισμούς σταθερών (constants)
- Ορισμούς αρχείων
- Ορισμούς συστατικών (αναφέρονται σε οντότητες)
- Ορισμούς ιδιοτήτων (attributes)
- Ορισμούς υποπρογραμμάτων (functions, procedures)

Η γενική μορφή της δήλωσης ενός πακέτου είναι:

Ονόματα βιβλιοθηκών
PACKAGE *όνομα πακέτου* **IS**
[Δηλώσεις σημάτων]
[Δηλώσεις συνιστωσών]
END *όνομα πακέτου* ;

Αφού δημιουργηθεί ένα πακέτο, μπορούμε να κληθεί μέσα από άλλα προγράμματα, με τη μορφή στιγμιοτύπων, όπως γίνεται και με τα συστατικά. Για να χρησιμοποιηθεί κάποιο πακέτο στον κώδικα θα πρέπει καταρχήν να δηλωθεί η βιβλιοθήκη που το περιέχει, καθώς και ποιο πακέτο από αυτή την βιβλιοθήκη θα χρησιμοποιηθεί. Στο *Πρόγραμμα 4*, παρουσιάζονται οι δηλώσεις και το υποκύκλωμα ενός πλήρη αθροιστή.

```
PACKAGE fulladd IS
    COMPONENT fulladd
        PORT (cin, x, y : IN STD_LOGIC ;
              s, cout   : OUT STD_LOGIC ) ;
    END COMPONENT ;
END fulladd ;
```

Πρόγραμμα 4 - Δήλωση Package

2.7.5. Βιβλιοθήκες (Libraries)

Η βιβλιοθήκη στην VHDL μοιάζει με μια αποθήκη στην οποία έχουν καταχωρηθεί δομικές μονάδες οι οποίες έχουν ελεγχθεί και οι οποίες μπορούν να αποτελέσουν δομικά κυκλώματα για μια μελλοντική κατασκευή. Τα κυκλώματα συνεπώς που περιλαμβάνονται σε μια βιβλιοθήκη είναι κυκλώματα που η χρήση τους είναι αρκετά συχνή. Κανονικά θα πρέπει κάθε φορά να προσδιορίζεται στο περιβάλλον που χρησιμοποιεί κάποιος, η βιβλιοθήκη στην οποία ανήκει η κατασκευή. Αν δεν γίνει αυτό τότε το περιβάλλον καταχωρεί την κατασκευή σε μια προκαθορισμένη (default) βιβλιοθήκη η οποία ονομάζεται work library. Το περιεχόμενο τη βιβλιοθήκης μπορεί να είναι entities, architectures και packages.

Για να συμπεριληφθεί κάθε φορά στην νέα κατασκευή κάτι από τα περιεχόμενα μιας βιβλιοθήκης, γράφεται πρώτα η βιβλιοθήκη στην οποία γίνεται η αναφορά με μια δήλωση

LIBRARY *όνομα βιβλιοθήκης*

και στην συνέχεια προσδιορίζεται η δομική μονάδα η οποία καλείται από την βιβλιοθήκη με μια πρόταση use. Έτσι για να γίνουν ορατά στην νέα κατασκευή όλα τα στοιχεία κάποιου πακέτου χρησιμοποιείται η δήλωση:

USE *όνομα βιβλιοθήκης. όνομα πακέτου*.**all**

Με τη δήλωση αυτή καθίστανται διαθέσιμες όλες (all) οι μονάδες του πακέτου που βρίσκεται μέσα στην βιβλιοθήκη. Σε περίπτωση που χρειάζεται να γίνουν διαθέσιμες μόνο κάποιες από τις μονάδες της βιβλιοθήκης, η σύνταξη της δήλωσης γίνεται:

USE *όνομα βιβλιοθήκης. όνομα πακέτου. όνομα μονάδας*

Επίσης για να γίνει ορατό οτιδήποτε υπάρχει μέσα σε μια βιβλιοθήκη τότε:

Use *όνομα βιβλιοθήκης.* **all**

Στο *Πρόγραμμα 5*, παρουσιάζεται ένα παράδειγμα δήλωσης βιβλιοθήκης.

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

Πρόγραμμα 5 - Δήλωση Library

2.8. Τύποι δεδομένων και αντικείμενα

Υπάρχουν τριών ειδών αντικείμενα στη γλώσσα VHDL που επιτρέπουν την παράσταση των πληροφοριών: σήματα (signals), σταθερές (constants) και μεταβλητές (variables). Τα σήματα είναι τα πιο σημαντικά αφού αντιπροσωπεύουν τις τιμές των σημάτων που διέρχονται από τα καλώδια ενός κυκλώματος. Τα ονόματα των αντικειμένων δεδομένων πρέπει να υπακούουν σε κάποιους κανόνες προκειμένου να θεωρούνται έγκυρα από τον μεταφραστή και να μην οδηγούν σε συγχύσεις. Συγκεκριμένα, κάθε όνομα:

- Πρέπει να ξεκινάει με ένα γράμμα
- Μπορεί να περιέχει οποινδήποτε αλφαριθμητικό χαρακτήρα
- Δεν πρέπει να ξεκινάει με τον χαρακτήρα '-'
- Δεν πρέπει να περιλαμβάνει διαδοχικούς χαρακτήρες '_ _'
- Δεν πρέπει να ανήκει στις δεσμευμένες λέξεις της γλώσσας, δηλαδή τις λέξεις εκείνες που έχουν ειδική σημασία στη γλώσσα (π.χ. entity, begin, bit, κ.α.)

2.8.1. Σήματα (Signals)

Τα σήματα είναι αντικείμενα δεδομένων τα οποία χρησιμοποιούνται για να συνδέσουν ταυτοχρονικά στοιχεία με τον ίδιο τρόπο που καλώδια συνδέουν υποκυκλώματα ενός κυκλώματος. Η χρήση τους αφορά στη σύνδεση λειτουργικών τμημάτων μιας κατασκευής καθώς και για την απευθείας εκχώρηση τιμών σε αυτά. Τα σήματα μπορούν να οριστούν είτε καθολικά σε ένα πακέτο είτε τοπικά σε μια αρχιτεκτονική ή σε μια άλλη περιοχή δηλώσεων. Μια δήλωση σήματος δημιουργεί ένα αντικείμενο το οποίο έχει μέσα του οποιοδήποτε χαρακτηριστικό χρειάζεται για να περιγραφεί η έννοια του «καλωδίου» συμπεριλαμβανομένου και του χρόνου. Σε ορισμένες περιπτώσεις δηλώσεων στις οποίες παίρνουν μέρος σήματα μπαίνουν από τις δηλώσεις αυτές και κάποιοι περιορισμοί σε ότι αφορά τον τρόπο που μεταφέρεται πληροφορία με το σήμα. Η μορφή ορισμού σήματος είναι:

SIGNAL *όνομα σήματος* : *τύπος σήματος*

Μερικά παραδείγματα ορισμού σημάτων είναι τα ακόλουθα:

Απλή δήλωση σήματος

```
SIGNAL T : STD_LOGIC ;
```

Τοπική δήλωση σήματος

Το σήμα είναι ορατό μόνο μέσα στο σώμα της αρχιτεκτονικής

```
ARCHITECTURE arch OF design IS
    SIGNAL M, L : STD_LOGIC_VECTOR (8 DOWNTO 0) ;
```

Καθολική δήλωση σήματος

Το σήμα είναι ορατό από όσα κυκλώματα χρησιμοποιούν το land

```
PACKAGE land IS
    SIGNAL M : STD_LOGIC ;
END land ;
```

Το σήμα γίνεται ορατό σε μια μελλοντική κατασκευή με την δήλωση:

```
USE WORK.land.M ;
```

Η σειρά με την οποία αναφέρονται οι δηλώσεις εκχωρήσεως τιμών σε σήματα δεν έχει σημασία, όπως δεν έχει σημασία και η σειρά με την οποία τοποθετούνται τα καλώδια σε ένα κύκλωμα.

```
SIGNAL a, b, c  : BIT ;
SIGNAL d        : REAL ;
SIGNAL x        : STD_LOGIC ;
SIGNAL y        : SIGNED (w DOWNTO 0) ;
y <= y1 + y2 ;
```

Σε κάποιους τύπους σημάτων δεν αρκεί μόνο η δήλωση τους, αλλά πρέπει να οριστούν και συγκεκριμένες βιβλιοθήκες που τους περιέχουν.

2.8.2. Σταθερές (Constants)

Μια σταθερά είναι ένα αντικείμενο με μια στατική τιμή ενός συγκεκριμένου τύπου δεδομένου. Τα αντικείμενα δεδομένων τύπου σταθερές, σε αντίθεση με τα αντικείμενα δεδομένων τύπου σήματος, δεν αντιπροσωπεύουν καλώδια του κυκλώματος. Μια σταθερά χρησιμεύει στην καλύτερη αναγνωσιμότητα του κώδικα. Η τιμή που θα δώσουμε σε μία σταθερά παραμένει ίδια σε όλη την έκταση του κώδικα. Οι σταθερές κάνουν την περιγραφή πιο εύκολα κατανοητή και παράλληλα πιο εύκολη στην διόρθωση.

Η μορφή δήλωσης μιας σταθεράς είναι:

CONSTANT *όνομα σταθεράς* : *τύπος* := *τιμή σταθεράς* ;

Μερικά παραδείγματα δηλώσεων σταθερών είναι τα εξής:

```
CONSTANT pi      : REAL   := 3.1416 ;
CONSTANT message: STRING (1 TO 7) := "project" ;
```

2.8.3. Μεταβλητές (Variables)

Οι μεταβλητές είναι αντικείμενα στην VHDL τα οποία αποθηκεύουν ενδιάμεσες τιμές ανάμεσα σε ακολουθιακές δηλώσεις της VHDL. Επιτρέπονται μόνο σε processes, functions και procedures και είναι πάντα τοπικές σε αυτές. Οι μεταβλητές στην VHDL μοιάζουν υπερβολικά με τις μεταβλητές στις γλώσσες προγραμματισμού. Παίρνουν αμέσως την τιμή που εκχωρείται σε αυτές και διευκολύνουν τους δύσκολους υπολογισμούς ή μια πολύπλοκη ακολουθία από λογικές πράξεις. Η εκχώρηση τιμής σε μια μεταβλητή γίνεται με το σύμβολο (:=).

Μια μεταβλητή μπορεί να δηλωθεί μέσα στο τμήμα δηλώσεων μιας οντότητας ή μιας αρχιτεκτονικής. Η μεταβλητή αυτή μπορεί να προσπελαστεί από οποιαδήποτε process μέσα στην αρχιτεκτονική. Μια μεταβλητή που θα δηλωθεί μέσα σε μια process θα είναι ορατή μόνο μέσα σε αυτή. Όπως και στην περίπτωση των σημάτων, έτσι και στις μεταβλητές μπορεί να δοθούν αρχικές τιμές προαιρετικά. Η τιμή που θα δοθεί στη μεταβλητή θα επηρεάσει την εξομοίωση αλλά και πάλι ο synthesizer θα αγνοήσει την τιμή του.

Η μορφή δήλωσης μιας μεταβλητής είναι:

VARIABLE *όνομα μεταβλητής* : *τύπος μεταβλητής* ;

Μεγάλη προσοχή πρέπει να δοθεί στις μεταβλητές που είναι κοινές σε πολλά componets του κυκλώματος που κατασκευάζεται. Λόγω του ότι η τιμή τους αλλάζει άμεσα μπορεί να υπάρχουν δυσάρεστα αποτελέσματα κατά την εξομοίωση. Τα λάθη αυτά είναι τα πιο δύσκολα στη λύση διότι αν και το πρόγραμμα φαίνεται συντακτικά σωστό ωστόσο η εξομοίωση δίνει λάθος αποτελέσματα. Η χρήση των μεταβλητών θα πρέπει να περιορίζεται όσο το δυνατόν μόνο εκεί που είναι απαραίτητο.

Μερικά παραδείγματα δηλώσεων μεταβλητών είναι τα εξής:

```
VARIABLE found  : BOOLEAN := true ;
VARIABLE P      : INTEGER RANGE 0 TO 15 := 15 ;
VARIABLE zita   : SIGNED (8 DOWNTO 0) ;
x := a - b ;
```

2.8.4. Literals

Τα literals είναι τιμές δεδομένων που εκχωρούνται σε αντικείμενα ή χρησιμοποιούνται μέσα σε εκφράσεις, απεικονίζοντας ορισμένες τιμές χωρίς απαραίτητα να ανήκουν σε κάποιο συγκεκριμένο τύπο. Παράδειγμα το literal '1' μπορεί να αναπαριστά είτε ένα bit είτε έναν χαρακτήρα. Τα literals ωστόσο κατατάσσονται σε ορισμένες κατηγορίες.

- *Character Literals*: κάθε ASCII χαρακτήρας κλεισμένος σε αποστρόφους (π.χ. '1','Z','$' κτλ.)
- *String Literals*: ομάδες χαρακτήρων ASCII κλεισμένες σε διπλές αποστρόφους " " (π.χ. "project", "A32" κτλ.)
- *Bit String Literals*: συγκεκριμένες μορφές από string literals που χρησιμεύουν για την αναπαράσταση δυαδικών οκταδικών και δεκαεξαδικών τιμών δεδομένων (π.χ. ο χαρακτήρας 'B' που χρησιμοποιείται για την απεικόνιση του δυαδικού αριθμού B"1000110" ενώ στο οκταδικό O"446" και στο δεκαεξαδικό H"C300")
- *Numeric Literals*: χωρίζονται σε δύο κατηγορίες τα integer literals και τα real literals. Τα integer literals εισάγονται σαν ακέραιοι έχοντας σαν αρχή τον χαρακτήρα ("-"). Το διάστημα των ακεραίων που υποστηρίζεται εξαρτάται από το εργαλείο σύνθεσης που χρησιμοποιείται. Τα real literals εισάγονται σαν μια πιο επεκταμένη μορφή στην οποία είναι απαραίτητο το δεκαδικό σημείο. Για μεγάλους αριθμούς χρησιμοποιείται η εκθετική μορφή με το σύμβολο 'E' (π.χ. 5.0, -12.9, 1.6E10, 1.2E-20)
- *Physical Literals*: ειδικές περιπτώσεις literals τα οποία αναπαριστούν φυσικές ποσότητες όπως είναι ο χρόνος, η τάση, το ρεύμα, η απόσταση κτλ. Έχουν και αριθμητικό αλλά και μέρος στο οποίο προσδιορίζονται οι μονάδες του μεγέθους (π.χ. 300 ns (nanoseconds), 900 ps (picoseconds), 40 mA (40 milliampers) κτλ.)

2.8.5. Τελεστές και πράξεις (Operators and operations)

Οι τελεστές στην VHDL όπως και στις άλλες γλώσσες προγραμματισμού χρησιμοποιούνται για να απεικονίσουν πράξεις πάνω σε δεδομένα. Με την VHDL μπορούμε να κάνουμε αριθμητικές, λογικές και σχεσιακές πράξεις χρησιμοποιώντας τους αντίστοιχους τελεστές.

Λογικές Πράξεις (Logical Operations)

Τα αποτελέσματα των λογικών πράξεων είναι του ιδίου τύπου και μεγέθους με τους τελεστέους. Πράξεις μεταξύ τύπων διαφορετικού μεγέθους απαγορεύονται. Όλες οι λογικές πράξεις έχουν την ίδια προτεραιότητα, με εξαίρεση τον τελεστή Not, που έχει μεγαλύτερη προτεραιότητα από τους υπόλοιπους. Για τον λόγο αυτό, όταν σε μία δήλωση έχουμε πολλές λογικές πράξεις, είναι καλό να χρησιμοποιούμε παρενθέσεις για να δηλώνουμε την προτεραιότητα της κάθε μιας.

Σχεσιακές Πράξεις (Relational Operations)

Οι σχεσιακές πράξεις κάνουν συγκρίσεις μεταξύ σημάτων, σταθερών ή μεταβλητών και ελέγχουν αν υπάρχει ισότητα ή ανισότητα μεταξύ τους. Όλες οι σχεσιακές πράξεις έχουν την ίδια προτεραιότητα μεταξύ τους και μεγαλύτερη από τις λογικές πράξεις. Το αποτέλεσμα που προκύπτει από μια τέτοια πράξη είναι πάντα τύπου Boolean, δηλαδή True ή False.

Στους *Πίνακες 2-9* που ακολουθούν παρουσιάζονται οι υποστηριζόμενοι από την VHDL τελεστές, η λειτουργία τους καθώς και η κατηγορία στην οποία ανήκουν.

<table>
<tr><th>Τελεστής</th><th>Περιγραφή</th><th>Τύποι τελεστών</th><th>Τύπος αποτελέσματος</th></tr>
<tr><td>and</td><td>And</td><td rowspan="4">Κάθε δυαδικός ή λογικός χαρακτήρας</td><td rowspan="4">Ο ίδιος τύπος</td></tr>
<tr><td>or</td><td>Or</td></tr>
<tr><td>nand</td><td>Not And</td></tr>
<tr><td>nor</td><td>Not Or</td></tr>
<tr><td>xor</td><td>Exclusive Or</td><td></td><td></td></tr>
<tr><td>nxor</td><td>Not Exclusive Or</td><td></td><td></td></tr>
</table>

Πίνακας 2 - Τελεστές Λογικής

Τελεστής	Περιγραφή	Τελεστικοί τύποι	Τύπος αποτελέσματος
=	Ισότητα	Κάθε τύπος	Λογικός
/=	Μη ισότητα	Κάθε τύπος	Λογικός
< <= > >=	Διάταξη	Κάθε διανυσματικός τύπος ή discrete array type	Λογικός

Πίνακας 3 - Τελεστές Σχέσεων

Τελεστής	Περιγραφή	Τύποι τελεστών	Τύπος αποτελέσματος
+	Πρόσθεση	Κάθε αριθμητικός τύπος	Ίδιος τύπος
-	Αφαίρεση	Κάθε αριθμητικός τύπος	Ίδιος τύπος
&	Αλληλουχία	Κάθε αριθμητικός τύπος	Ίδιος τύπος
&	Αλληλουχία	Κάθε διάνυσμα ή element type	Ίδιος διανυσματικός τύπος

Πίνακας 4 - Τελεστές Πρόσθεσης

Τελεστής	Περιγραφή	Τελεστικοί τύποι	Τύπος αποτελέσματος
*	Πολλαπλασιασμός	Δεξιά /Αριστερά: κάθε ακέραιος or floating point type	Ο ίδιος τύπος
*	Πολλαπλασιασμός	Αριστερά: Κάθε φυσικός τύπος Δεξιά: Ακέραιος ή πραγματικός τύπος	Όπως αριστερά
*	Πολλαπλασιασμός	Αριστερά: Κάθε ακέραιος ή πραγματικός τύπος Δεξιά: κάθε φυσικός τύπος	Όπως δεξιά
/	Διαίρεση	Κάθε ακέραιος or floating point type	Ο ίδιος τύπος
/	Διαίρεση	Αριστερά: Κάθε φυσικός τύπος Δεξιά: Κάθε ακέραιος ή πραγματικός τύπος	Όπως αριστερά
/	Διαίρεση	Αριστερά/ Δεξιά: Κάθε φυσικός τύπος	Ακέραιος

Πίνακας 5 - Τελεστές Πολλαπλασιασμού

Τελεστής	Περιγραφή	Τελεστικοί τύποι	Τύπος αποτελέσματος
Mod	Modulus	Κάθε ακέραιος τύπος	Ίδιος τύπος
Rem	Remainder	Κάθε ακέραιος τύπος	Ίδιος τύπος

Πίνακας 6 - Τελεστές Υπολοίπου

Τελεστής	Περιγραφή	Τελεστικοί τύποι	Τύπος αποτελέσματος
+	Identity	Κάθε αριθμητικός τύπος	Ο ίδιος τύπος
-	Άρνηση	Κάθε αριθμητικός τύπος	Ο ίδιος τύπος

Πίνακας 7 - Τελεστές Προσήμου

Τελεστής	Περιγραφή	Τύποι τελεστών	Τύπος αποτελέσματος
Sll	Αριστερή λογική ολίσθηση	Αριστερά: κάθε μονοδιάστατο διάνυσμα που τα στοιχεία του είναι δυαδικά ή λογικά Δεξιά: ακέραιος	Το ίδιο όπως ο αριστερός τελεστικός τύπος
srl	Δεξιά λογική ολίσθηση	(όπως παραπάνω)	(όπως παραπάνω)
sla	Αριστερή αριθμητική ολίσθηση	(όπως παραπάνω)	(όπως παραπάνω)
sra	Δεξιά αριθμητική ολίσθηση	(όπως παραπάνω)	(όπως παραπάνω)
rol	Αριστερή λογική περιστροφή	(όπως παραπάνω)	(όπως παραπάνω)
ror	Δεξιά λογική περιστροφή	(όπως παραπάνω)	(όπως παραπάνω)

Πίνακας 8 - Τελεστές Μετατόπισης

Operator	Description	Operand types	Result Type
**	Εκθετικό	Αριστερά: κάθε ακέραιος τύπος Δεξιά: ακέραιος τύπος	Όπως αριστερά
**	Εκθετικό	Left: any floating point type	Όπως αριστερά
		Δεξιά: ακέραιος	
abs	Απόλυτη τιμή	Κάθε αριθμητικός τύπος	Ο ίδιος αριθμητικός τύπος
not	Λογική άρνηση	Κάθε δυαδικός ή λογικός τύπος	Ο ίδιος τύπος

Πίνακας 9 - Άλλου είδους τελεστές

2.8.6. Ιδιότητες (Attributes)

Οι ιδιότητες στην VHDL αποτελούν ένα πολύ χρήσιμο εργαλείο το οποίο βοηθάει στη λήψη πληροφοριών για ένα αντικείμενο, βάση της ιδιότητας που χρησιμοποιείται, καθώς και στην απόδοση επιπρόσθετων πληροφοριών σε αυτό . Οι πληροφορίες που λαμβάνονται με την χρήση των ιδιοτήτων δεν είναι πληροφορίες γενικά που μεταφέρει το αντικείμενο (π.χ. κάποια τιμή της μεταβλητής ενός αντικειμένου). Οι ιδιότητες είτε μπορούν να οριστούν εκείνη τη στιγμή, είτε μπορούν να ληφθούν έτοιμες από τις βιβλιοθήκες του προγράμματος περιγραφής. Εκφράζονται με πρόθεμα την μεταβλητή ή το σήμα στο οποίο εφαρμόζονται, όπως π.χ. η έκφραση:

```
WAIT UNTIL Sig = '1' AND Sig'EVENT AND Sig'LAST_VALUE = '0' ;
```

στην οποία ιδιότητες είναι οι 'EVENT και 'LAST_VALUE και έχουν πρόθεμα το σήμα Sig .

Πολλές από τις μεταβλητές έχουν παραμέτρους, όπως στην παρακάτω έκφραση, όπου ο ακέραιος 2 χρησιμοποιείται ως παράμετρος στην ιδιότητα VAL:

```
VARIABLE A: state_type := state_type'VAL(2);
```

Οι ιδιότητες χωρίζονται σε διάφορες κατηγορίες ανάλογα με το που αναφέρονται.

2.8.7. Τύποι εντολών

Οι εντολές στην VHDL χωρίζονται σε σε εντολές ακολουθιακής αντιστοίχησης (sequential assignment statements) και σε εντολές σύμφωνης αντιστοίχησης (consistent assignment statements). Η σειρά γραφής των εντολών ακολουθιακής αντιστοίχησης μέσα στον κώδικα έχει σημασία, αφού τα σήματα αλλάζουν τιμή με τη σειρά που εκτελούνται αυτές οι εντολές. Αντιθέτως, με τις εντολές σύμφωνης αντιστοίχησης η εκχώρηση των τιμών στα σήματα γίνεται μετά το πέρας της διαδικασίας (process) μέσα στην οποία εκτελούνται οι εντολές, οπότε η σειρά γραφής τους δεν είναι καθοριστική.

Εντολές Ακολουθιακής Αντιστοίχισης

Η σειρά με την οποία δίνονται οι εντολές ακολουθιακής αντιστοίχισης είναι σημαντική. Για να διαχωρίζονται από τις εντολές σύμφωνης αντιστοίχισης, χρησιμοποιείται η εντολή PROCESS, η οποία αναγράφεται μέσα στο κύριο σώμα της αρχιτεκτονικής και με τη σειρά της περιέχει τις εντολές ακολουθιακής αντιστοίχισης. Αυτές είναι οι IF, CASE και οι εντολές βρόχων (FOR και WHILE).

Εντολή Διαδικασίας (Process)

Η εντολή PROCESS συνοδεύεται πάντα από μια παρένθεση μέσα στην οποία περιέχεται η λεγόμενη "λίστα ευαισθησίας", η οποία περιέχει ένα ή περισσότερα σήματα, με κάθε αλλαγή των οποίων ενεργοποιείται η PROCESS και κατ' επέκταση και οι εντολές που περιέχει. Τότε, οι εντολές που περιλαμβάνονται σ' αυτήν εκτελούνται με τη σειρά. Όταν υπολογιστούν οι τιμές όλων των σημάτων που περιλαμβάνονται στη διαδικασία, τότε οι τελικές αντιστοιχήσεις παράγουν ορατό αποτέλεσμα στα σήματα εξόδου.

Η γενική μορφή της διαδικασίας είναι η εξής:

```
PROCESS (ονόματα σημάτων)
    BEGIN
        IF …
        CASE …
        FOR …
        WHILE …
END PROCESS ;
```

Η δομή διαδικασίας μπορεί να χρησιμοποιηθεί για να περιγράψει τόσο συνδυαστικά όσο και ακολουθιακά κυκλώματα, λόγω της ευκολίας που παρέχει στην περιγραφή του κυκλώματος.

Εντολή IF

Η εντολή IF λειτουργεί ως μια εντολή διακλάδωσης υπό συνθήκη και βρίσκεται πάντα μέσα σε μία εντολή διαδικασίας (PROCESS).

Η γενική μορφή της είναι η εξής:

IF *έκφραση* **THEN**
    Εντολές ;
ELSIF *έκφραση* **THEN**
    Εντολές ;
END IF ;

Στο *Πρόγραμμα 6*, παρουσιάζεται ένα μέρος κώδικα VHDL που περιγράφει έναν πολυπλέκτης 2 προς 1 με χρήση της εντολής IF. Αν το σήμα επιλογής Sel πάρει την τιμή '0' τότε το σήμα εξόδου c θα λάβει την τιμή του σήματος a, διαφορετικά του σήματος b.

```
PROCESS (Sel, a, b)
BEGIN
    IF Sel = '0' THEN
        c <= a ;
    ELSE
        c <=b ;
    END IF;
END PROCESS ;
```

Πρόγραμμα 6 - Εντολή IF

Εντολή CASE

Η εντολή CASE λειτουργεί με την ίδια λογική με αυτή της IF, δηλαδή για τη δημιουργία διακλαδώσεων στον κώδικα.

Η γενική μορφή της είναι η εξής:

CASE *έκφραση* **IS**
    WHEN *σταθερή τιμή* =>
        Εντολές ;
    WHEN *σταθερή τιμή* =>
        Εντολές ;
    WHEN *σταθερή τιμή* =>
END CASE ;

Στο *Πρόγραμμα 7*, περιγράφεται ο ίδιος πολυπλέκτης 2 προς 1 με χρήση της εντολής CASE, η οποία απαιτεί να αναγράφονται όλες οι δυνατές τιμές της έκφρασης – συνθήκης που χρησιμοποιείται. Για το λόγο αυτό γίνεται η χρήση της δεσμευμένης λέξη OTHERS, που υποδηλώνει όλες τις υπόλοιπες περιπτώσεις. Οι εντολές WHEN θα πρέπει να είναι αμοιβαία αποκλειστικές μεταξύ τους.

```
PROCESS (Sel, a, b)
BEGIN
    CASE Sel IS
        WHEN '0' =>
        c <= a ;
        WHEN OTHERS =>
        c <=b ;
    END CASE;
END PROCESS ;
```

Πρόγραμμα 7 - Εντολή CASE

Εντολές Βρόχων

Οι εντολές βρόχων στην VHDL είναι η εντολή FOR και η εντολή WHILE, οι οποίες χρησιμοποιούνται για την επανάληψη μίας ή περισσότερων εντολών.

Η γενική μορφή των εντολών είναι η εξής:

Τίτλος βρόχου:
FOR *όνομα μεταβλητής* **IN** *πεδίο τιμών* **LOOP**
Εντολές ;
END LOOP ;

Τίτλος βρόχου:
WHILE *λογική έκφραση* **LOOP**
Εντολές ;
END LOOP ;

Στα *Προγράμματα 8* και *9*, παρουσιάζεται ένα μέρος κώδικα VHDL που περιγράφει έναν μετρητή που μετράει τον αριθμών των άσσων στο σήμα εισόδου. Κάθε φορά που συναντάει έναν άσσο σε κάποιο από τα τέσσερα bits της εισόδου, αυξάνεται κατά ένα η τιμή της εξόδου. Στους παρακάτω κώδικες παρουσιάζεται το ίδιο κύκλωμα με τη χρήση των εντολών FOR και WHILE.

```
PROCESS (x)
BEGIN
    Expr: FOR i IN 0 to 3 LOOP
        IF x(i) ='1' THEN
            OUT <= OUT + 1 ;
        END IF ;
    END LOOP ;
END PROCESS ;
```

Πρόγραμμα 8 - Εντολή FOR

```
PROCESS (x)
BEGIN
    Expr: WHILE (i >=0) AND (i <4) LOOP
        IF x(i) = '1' THEN
            OUT <= OUT + 1;
        END IF;
    END LOOP ;
END PROCESS ;
```

Πρόγραμμα 9 - Εντολή WHILE

Κεφάλαιο 3

Το λογισμικό Quartus II

Το λογισμικό Quartus II ένα από τα πιο γνωστά προγράμματα σχεδίασης CAD. Αποτελεί πνευματική ιδιοκτησία της εταιρίας ALTERA, η οποία κατασκευάζει ορισμένες από τις πιο διαδεδομένες οικογένειες κυκλωμάτων CPLDs και FPGAs. Είναι κατάλληλο για σχεδιασμό κυκλωμάτων σε υψηλής πυκνότητας FPGAs, για εφαρμογές χαμηλού κόστους.

Το Quartus II χρησιμοποιείται για την ανάπτυξη και τον προγραμματισμό όλων των κυκλωμάτων της εταιρείας ALTERA, και κυρίως των διατάξεων CPLDs και FPGAs.

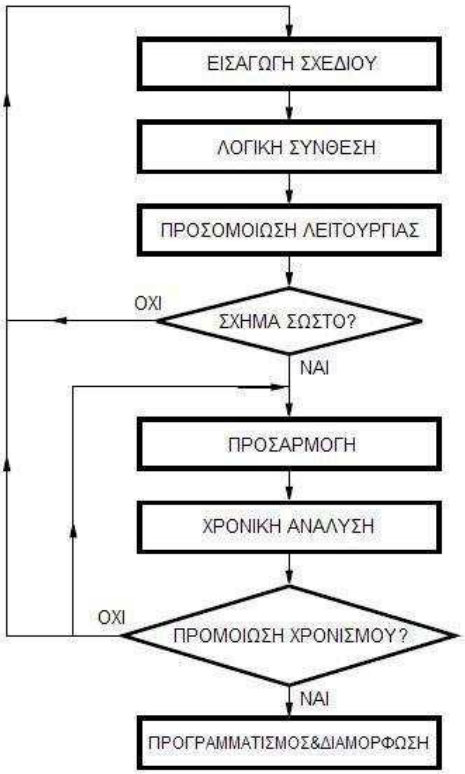
3.1. Ροή Εργασιών

Το Quartus II ενσωματώνει σε ένα ολοκληρωμένο περιβάλλον λογισμικού όλες τις λειτουργίες των συστημάτων σχεδίασης, ενώ παρέχει επιπλέον λειτουργίες όπως την εύρεση της καταλληλότερης διαμόρφωσης, την χρονική ανάλυση ώστε να διαπιστωθεί η ορθή χρονική απόκριση του κυκλώματος στους παλμούς εισόδου καθώς και τον προγραμματισμό (διαμόρφωση) της διάταξης.

Αποτελείται από εργαλεία που εκτελούν τις ακόλουθες εργασίες:

- *Εισαγωγή σχεδίου*: Επιτρέπει στο σχεδιαστή την εισαγωγή του επιθυμητού συστήματος με διάφορες μορφές (πίνακα αληθείας, σχηματικό διάγραμμα ή πρόγραμμα σε γλώσσα HDL)
- *Λογική σύνθεση και βελτιστοποίηση*: Η είσοδος μετατρέπεται στις κατάλληλες λογικές συναρτήσεις, σύμφωνα με τους κανόνες της τεχνολογίας για την οποία προορίζεται το σχέδιο, που τελικά θα διαμορφώσουμε
- *Προσομοίωση λειτουργίας*: Χρησιμοποιείται για την επαλήθευση της ορθής λειτουργίας του κυκλώματος, με βάση τις εισόδους που έχει δώσει ο σχεδιαστής

- *Προσαρμογή (fitting)*: Στη φάση αυτή χρησιμοποιείται η βάση δεδομένων που δημιουργήθηκε κατά την λογική σύνθεση και αντιστοιχίζεται η ψηφιακή λογική στους ελεύθερους πόρους της διάταξης που θέλουμε να κατασκευάσουμε, λαμβάνοντας υπόψη και τις χρονικές απαιτήσεις που τέθηκαν
- *Χρονική ανάλυση*: Οι καλύτεροι και οι χειρότεροι χρόνοι του κυκλώματος παράγονται, με βάση τις προβλεπόμενες καθυστερήσεις κατά μήκος των διαδρομών, εξαιτίας του μήκους των καλωδίων και των ενδιάμεσων βαθμίδων
- *Προσομοίωση χρονισμού*: Προσδιορίζει τις καθυστερήσεις διάδοσης που αναμένεται να προκύψουν στο πραγματικό κύκλωμα
- *Προγραμματισμός και Διαμόρφωση της συσκευής*: Η τελευταία ενέργεια που γίνεται για την ολοκλήρωση της δημιουργίας του κυκλώματος. Το CPLD ή το FPGA πρέπει να διαμορφωθεί κατάλληλα ώστε να έχουμε το κύκλωμα που επιθυμούμε

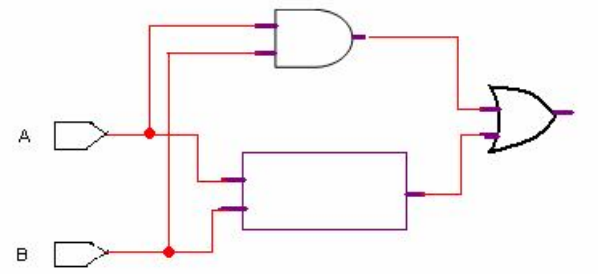


Σχήμα 9 - Ροή Εργασιών στο Quartus II

3.2. Εισαγωγή Σχεδίασης

Κατά τον πρώτο τρόπο σχεδίασης, δηλαδή με την εισαγωγή πινάκων αληθείας, ο χρήστης μέσω ενός επεξεργαστή κυματομορφών εισάγει στο πρόγραμμα τις τιμές των εισόδων και τις επιθυμητές τιμές των εξόδων που θέλει να έχει το κύκλωμά του. Το πρόγραμμα σχεδίασης αναλαμβάνει να δημιουργήσει αυτό το κύκλωμα. Αυτός ο τρόπος εισαγωγής κυκλώματος δεν είναι εύχρηστος γι' αυτό και προτιμάται μόνο για απλά και μικρά κυκλώματα. Για να δώσουμε με αυτό τον τρόπο είσοδο στο Quartus II χρησιμοποιούμε τον*Waveform Editor*.

Ο δεύτερος τρόπος εισαγωγής είναι το σχηματικό διάγραμμα. Σε αυτόν τον τρόπο η σχεδίαση γίνεται με την βοήθεια σχεδιαστικών εργαλείων που παρέχονται από το πρόγραμμα σχεδίασης και με την χρήση ήδη υπαρχουσών βιβλιοθηκών, οι οποίες περιέχουν από απλές πύλες μέχρι και σύνθετα κυκλώματα. Οι πύλες αυτές περιέχονται σε διάφορους τύπους και με διαφορετικό αριθμό εισόδων. Για την αποφυγή δημιουργίας μεγάλων κυκλωμάτων, παρέχεται από το σχεδιαστικό πρόγραμμα η δυνατότητα χρήσης ιεραρχικών βαθμίδων (blocks), που επιτρέπει τη δημιουργία κυκλωμάτων που στο εσωτερικό τους περιέχουν άλλα μικρότερα κυκλώματα. Αυτός ο τρόπος σχεδίασης λέγεται και ιεραρχική σχεδίαση. Για να δώσουμε με αυτό τον τρόπο είσοδο στο Quartus II χρησιμοποιούμε τον *Block Editor*.



Σχήμα 10 - Κύκλωμα με χρήση Ιεραρχικής Σχεδίασης

Ο τρίτος τρόπος είναι η σχεδίαση με την βοήθεια κάποιας γλώσσας περιγραφής κυκλωμάτων. Σ' αυτόν τον τρόπο σχεδίασης περιγράφεται η λειτουργία του κυκλώματος με τη βοήθεια κατάλληλου κώδικα. Για να δώσουμε με αυτό τον τρόπο είσοδο στο Quartus II χρησιμοποιούμε τον *Text Editor*.

3.3. Σύνθεση

Η διαδικασία της σύνθεσης είναι το επόμενο βήμα μετά την ολοκλήρωση της περιγραφής της ψηφιακής λογικής. 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 θέλουμε να διαμορφώσουμε. Για παράδειγμα, έστω ότι έχουμε σαν είσοδο κάποιον πίνακα αληθείας, και σαν στόχο μια διάταξη CPLD. Κατά την διάρκεια της σύνθεσης δημιουργούνται λογικές συναρτήσεις που εκφράζουν κατάλληλα τους πίνακες αληθείας, ώστε τελικά να απεικονιστούν σε μια διάταξη λογικών πινάκων (logic arrays). Οι γεννήτριες συναρτήσεων των CPLDs στηρίζονται στους λογικούς πίνακες AND-OR. 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). Όταν η εισαγωγή γίνεται μέσω μιας γλώσσας περιγραφής υλικού, κατά την διαδικασία της σύνθεσης τίθεται σε λειτουργία ο μεταφραστής, ένα άλλο τμήμα της σύνθεσης, και σαν έξοδο έχουμε την περιγραφή του κυκλώματος σε χαμηλό επίπεδο, κατάλληλο για την τεχνολογία της διάταξης-στόχου.

Η σύνθεση χρησιμοποιείται ως ένα αυτόματο εργαλείο υλοποίησης του αρχικού κυκλώματος που δίνει ο σχεδιαστής, με τρόπο κατάλληλο για την τεχνολογία του τελικού μας στόχου. Ταυτόχρονα, το κύκλωμα που παίρνουμε στην έξοδο της σύνθεσης είναι καλύτερο από το αρχικό, που σημαίνει ότι κατά την σύνθεση έχουμε και βελτιστοποίηση του κυκλώματος. Το βελτιστοποιημένο σύστημα που έχει παραχθεί με την διαδικασία της λογικής σύνθεσης εξαρτάται από το είδος των διατιθέμενων πόρων στο ολοκληρωμένο κύκλωμα-στόχο καθώς και από το πρόγραμμα.

3.4. 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, που σημαίνει ότι 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.

3.5. Χρονική Ανάλυση – Παραγωγή Αρχείων Προγραμματισμού

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 καθυστερήσεις που υπάρχουν κατά μήκος των διαδρομών, οι οποίες οφείλονται στο μήκος των καλωδίων καθώς και στον αριθμό των ενδιάμεσων βαθμίδων. Η χρονική ανάλυση είναι ένα σημαντικό σημείο κατά τη σχεδίαση ενός κυκλώματος, καθώς θα πρέπει οι χρόνοι που παίρνουμε από την εκτέλεσή της να είναι συμβατοί με αυτούς που καθορίζονται από το ρολόι του συστήματος. Σε περίπτωση μη συμβατότητας των χρόνων θα πρέπει οι αποκρίσεις αυτές να διορθωθούν, με κατάλληλες χρονικές εκχωρήσεις (timing assignments) στο λογισμικό και επανάληψη της δρομολόγησης.

Στην τελευταία φάση (assembling) γίνεται και η παραγωγή συμβολικού (.hex) και δυαδικού (.sof) κώδικα, δημιουργούνται τα προγραμματιστικά αρχεία που θα χρησιμοποιηθούν για την τελική διαμόρφωση. Το αρχείο sof είναι το αρχείο που περνάμε και στην αναπτυξιακή πλακέτα ώστε να προγραμματίσουμε το FPGA να είναι το σύστημα που δημιουργήσαμε.

Οι παραπάνω διαδικασίες της σύνθεσης, της προσαρμογής, της χρονικής ανάλυσης 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).

3.6. Προσομοίωση

Κατά την διαδικασία της λειτουργίας της προσομοίωσης ελέγχεται κατά πόσο το κύκλωμα λειτουργεί σωστά, καθώς δεν αρκεί μόνο η λειτουργία της σύνθεσης όπου 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, συνδυάζοντας δύο παραμέτρους. Ο ένας είναι το αρχικό μας 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 ανταποκρίνονται στις αρχικές προδιαγραφές που είχε θέσει για το κύκλωμα. Ο προσομοιωτής εξάγει τις τιμές της εξόδου του κυκλώματος μέσω ενός πίνακα αληθείας ή μέσω ενός διαγράμματος χρονισμού. Στην περίπτωση του διαγράμματος χρονισμού πρέπει να ληφθεί υπόψη ότι ο προσομοιωτής μοντελοποιεί και τον χρόνο που απαιτείται για να μεταβούν τα σήματα των εισόδων στις πύλες, αλλά πάντα με βάση κάποια μοντέλα που μπορεί στην πράξη να μην είναι πάντα ακριβή, με αποτέλεσμα να υπάρχει ένα ποσοστό αμφιβολίας ακόμα και μετά την προσομοίωση για το αν το κύκλωμα που σχεδιάσαμε μπορεί να υλοποιηθεί με ακρίβεια.

3.7. Προγραμματισμός - Διαμόρφωση της Συσκευής

Η διαδικασία του προγραμματισμού είναι η τελευταία ενέργεια που γίνεται για την ολοκλήρωση της δημιουργίας του κυκλώματος. 5ς γνωστό, τα CPLDs ή τα FPGAs πρέπει να διαμορφωθούν για να υλοποιήσουν το κύκλωμα που σχεδιάσαμε. Το αρχείο που χρειαζόμαστε είναι το .sof αρχείο που δημιουργήθηκε κατά την φάση της χρονικής ανάλυσης.

Η εταιρία ALTERA επιτρέπει τον προγραμματισμό των συσκευών της με δύο τρόπους. Ο ένας είναι μέσω του κυκλώματος διεπαφής JTAG και ο άλλος ο «Active Serial (AS) mode». Τα αρχεία διαμόρφωσης μεταφέρονται από τον υπολογιστή του χρήστη στο board όπου βρίσκεται το CPLD ή το FPGA, μέσω ενός καλωδίου το οποίο συνδέεται σε θύρα του υπολογιστή μας (παράλληλη ή USB). Η σύνδεση αυτή γίνεται μέσω του κατάλληλου οδηγού (driver) USB-Blaster ή BYTE-BLASTER.

Στη περίπτωση που χρησιμοποιήσουμε την JTAG διεπαφή, τα δεδομένα μας πηγαίνουν κατευθείαν στο ολοκληρωμένο κύκλωμα. Με αυτόν τον τρόπο το ολοκληρωμένο κύκλωμα (αν είναι FPGA) διατηρεί την διαμόρφωση που του έχουμε δώσει για όσο διαρκεί η τροφοδοσία του. Αν σταματήσει να τροφοδοτείται χάνεται και η διαμόρφωση του. Αυτό δεν ισχύει για τα κυκλώματα CPLDs, τα οποία διαμορφώνονται μόνιμα, με δυνατότητα επανεγγραφής, όπως οι μνήμες flash EEPROM.

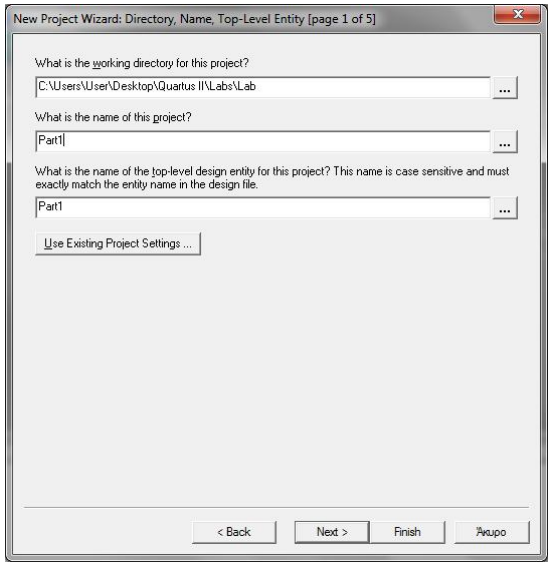
Στην άλλη περίπτωση (AS), μια διάταξη με μνήμες flash, που βρίσκεται πάνω στην ίδια πλακέτα με το FPGA, χρησιμοποιείται για να αποθηκεύει τα αρχεία διαμόρφωσης. Σ' αυτήν την περίπτωση το Quartus II στέλνει τα δεδομένα στη μνήμη Flash και αυτή με τη σειρά της στο chip FPGA. Στην περίπτωση αυτή, τα αρχεία διαμόρφωσης παραμένουν στη μνήμη ακόμη και όταν διακοπεί η τροφοδοσία. Η επιλογή για το ποιόν από τους δύο τρόπους θα χρησιμοποιήσουμε γίνεται συνήθως μέσω ενός διακόπτη RUN/PROG που βρίσκεται πάνω στο board. Η αυτόματη (default) επιλογή είναι για διαμόρφωση με JTAG, ενώ η δεύτερη για Active Serial (AS).

3.8. Σχεδίαση και Προσομοίωση Ψηφιακού Κυκλώματος

Η διαδικασία υλοποίησης ενός ψηφιακού κυκλώματος ξεκινά με την επιλογή του τρόπου σχεδίασης (σχηματικά και με τη γλώσσα περιγραφής υλικού VHDL) για να φτάσει στον προγραμματισμό ενός FPGA, ώστε να εκτελεί την λειτουργία

3.8.1. Ορισμός του σχεδίου (Project)

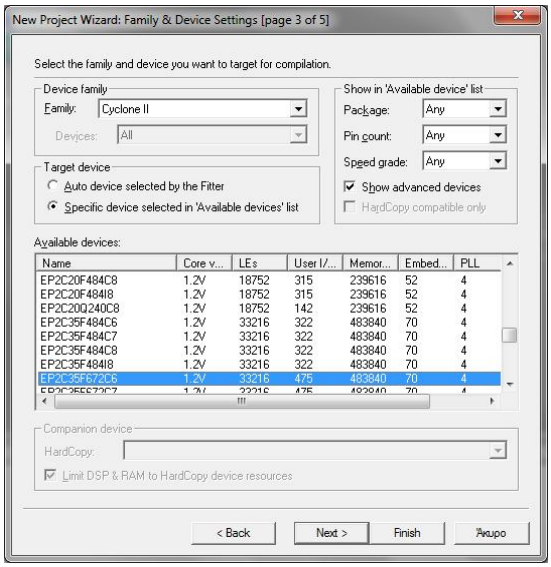
Με την εκκίνηση του σχεδιαστικού περιβάλλοντος του Quartus II, ζητείται η ονομασία του project που θα δημιουργηθεί. Η έννοια του project περιλαμβάνει το σύνολο των αρχείων που δημιουργείται από τον χρήστη (το αρχικό σχέδιο και τις κυματομορφές εισόδου για την προσομοίωση), καθώς και τα αρχεία που δημιουργεί το πρόγραμμα για να εκτελέσει τις διάφορες λειτουργίες του. Με απλά λόγια, ως project θεωρείται το σύνολο των αρχείων που δημιουργούνται για μια εφαρμογή.



Σχήμα 11 - Δημιουργία νέου project

Η δημιουργία ενός project γίνεται από το μενού File | New Project Wizard. Στο παράθυρο του οδηγού που εμφανίζεται (*Σχήμα 11*) δηλώνεται το όνομα του project, καθώς και το όνομα του φακέλου (workind directory) που θα το περιέχει.

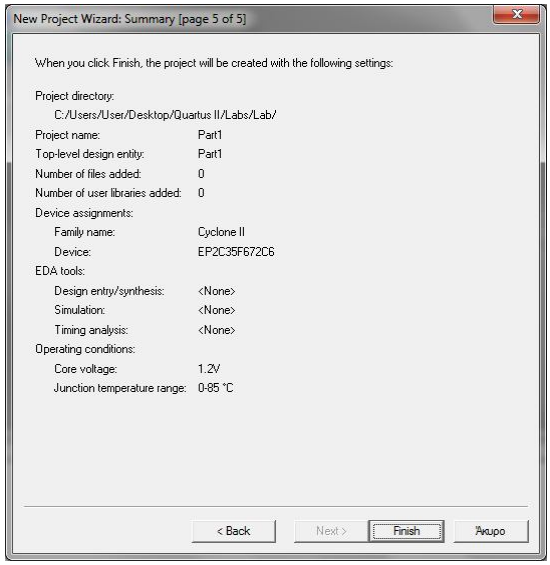
Πατώντας το κουμπί Next στην ίδια καρτέλα, το πρόγραμμα ζητάει την επιλογή από μία λίστα της συγκεκριμένη διάταξης FPGA που θα χρησιμοποιήθεί. Το όνομα του project είναι Part1 και το FPGA που θα χρησιμοποιηθεί ανήκει στην οικογένεια Cyclone II (Device family) και είναι το EP2C35F672C6 (Target device) (*Σχήμα 12*).



Σχήμα 12 - Επιλογή και ρυθμίσεις FPGA

Πατώντας το κουμπί Next οδηγούμαστε στην καρτέλλα επιβεβαίωσης, όπου αναγράφονται τα στοιχεία για το project που θα δημιουργήσουμε.

Η δημιουργία ολοκληρώνεται πατώντας το κουμπί Finish (*Σχήμα 13*).

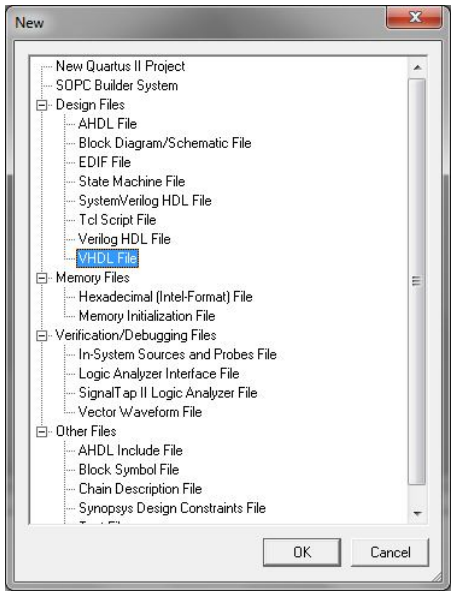


Σχήμα 13 - Ολοκλήρωση δημιουργίας project

3.8.2. Εισαγωγή αρχείου

Αφού έχουμε δώσει όνομα στο project μπορούμε να συνεχίσουμε κάνοντας την εισαγωγή του κυκλώματος με όποιον τρόπο επιθυμούμε.

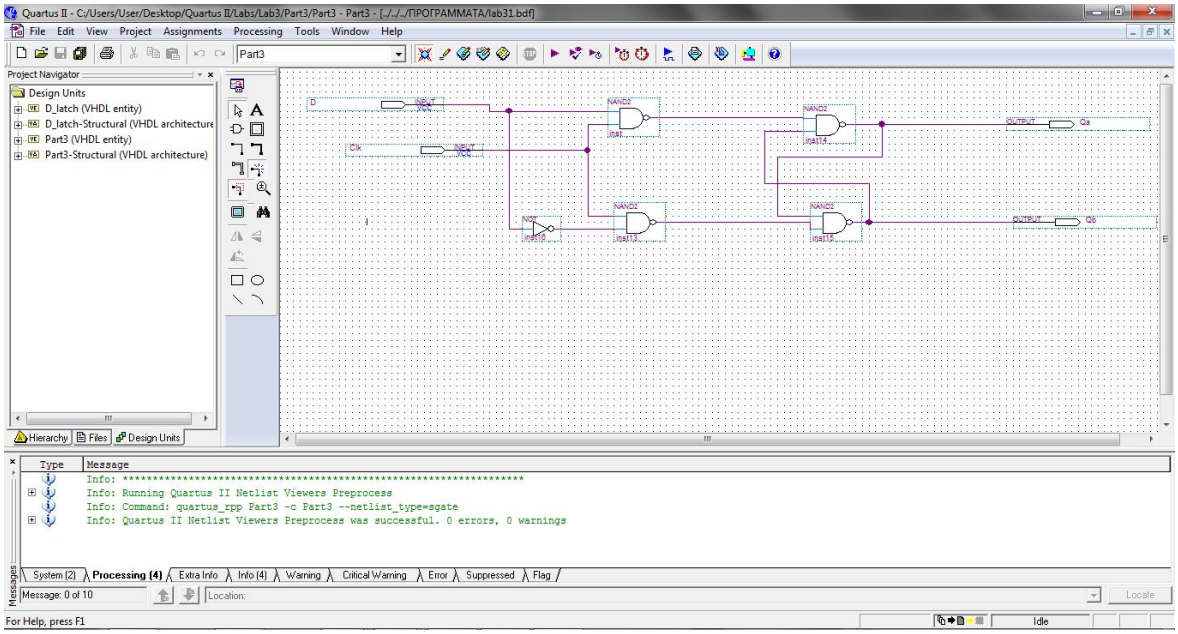
Η επιλογή αυτή γίνεται από το μενού File | New. Υπάρχει η δυνατότητα επιλογής ανάμεσα σε διαφορετικούς τρόπους δημιουργίας του κυκλώματος. Αν θέλουμε να το περιγράψουμε σχηματικά διαλέγουμε την επιλογή Block Diagram/Schematic File, ενώ αν θέλουμε να το περιγράψουμε με κάποια γλώσσα περιγραφής υλικού διαλέγουμε μία από τις Verilog HDL File ή VHDL File.



Σχήμα 14 - Επιλογή εισαγωγής κυκλώματος

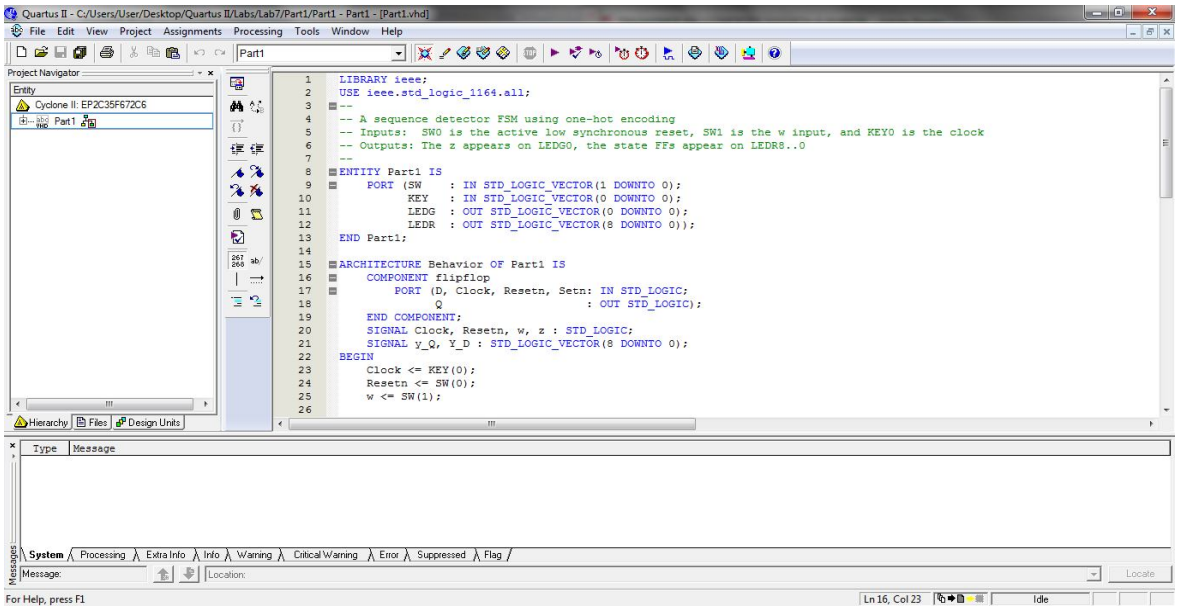
Στην περίπτωση που το κύκλωμα περιγράφεται σχηματικά, το Quartus II παρέχει ένα μεγάλο αριθμό από πύλες, από ακροδέκτες εισόδου/εξόδου καθώς και από άλλα κυκλώματα όπως flip-flop που περιέχονται σε βιβλιοθήκες, τα οποία βρίσκονται στο μενού Edit | Insert symbol. Εναλλακτικά, οι βιβλιοθήκες ανοίγουν πιέζοντας το πλήκτρο Symbol Tool στα αριστερά του επεξεργαστή. Οι βασικές πύλες βρίσκονται στη βιβλιοθήκη Primitives/logic, ενώ οι ακροδέκτες I/O βρίσκονται στη βιβλιοθήκη Primitives/Pins.

Τα ονόματα των ακροδεκτών μπορούν να οριστούν κατάλληλα κάνοντας διπλό κλικ πάνω στους ακροδέκτες. Μετά τη σχεδίαση του κυκλώματος, το αρχείο πρέπει να αποθηκευτεί με το ίδιο όνομα που είχαμε δώσει στο project, δηλαδή Part1.



Σχήμα 15 - Σχηματική περιγραφή κυκλώματος

Στην περίπτωση που το κύκλωμα περιγράφεται σε γλώσσα VHDL, το Quartus II μας μεταφέρει σε ένα παράθυρο εισαγωγής κώδικα VHDL που ανοίγει στο δεξί μέρος της οθόνης με το όνομα vdhl.vhd. Αφού το αρχείο γραφεί σε κώδικα VHDL, θα πρέπει να αποθηκευτεί έχοντας πάντα υπόψη τη χρήση του ίδιου ονόματος για το project και το top-level entity, καθώς και του αρχείου VHDL και του entity μέσα στο αρχείο VHDL.



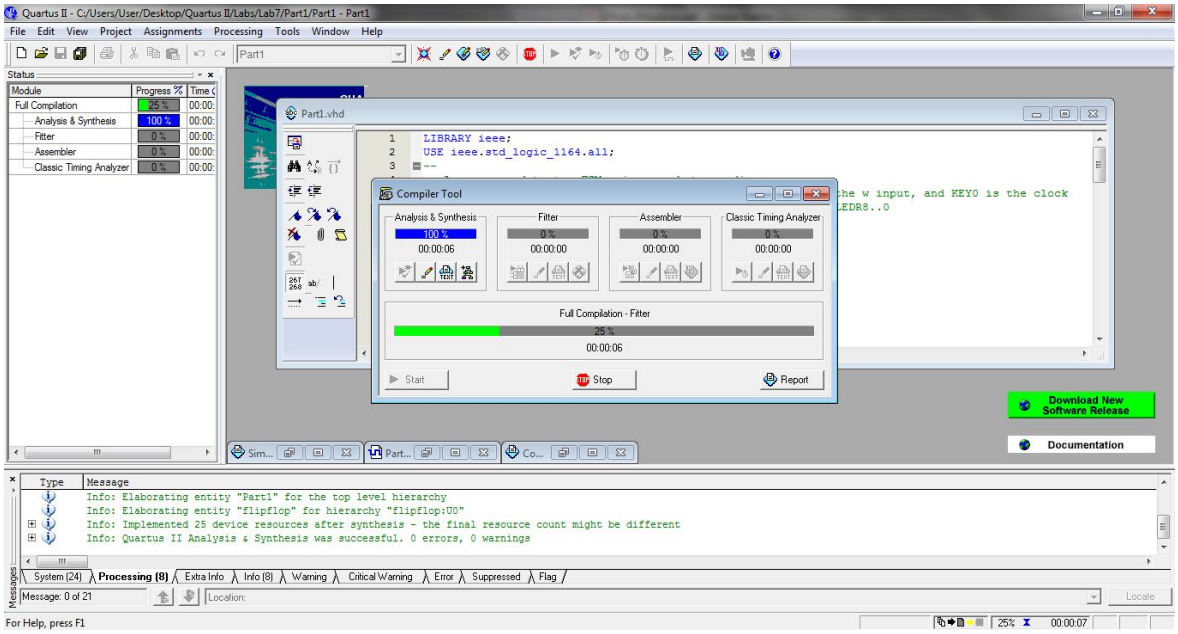
Σχήμα 16 - Συγγραφή κώδικα VHDL

3.8.3. Μετάφραση (Compilation)

Μετά το τέλος της περιγραφής του κυκλώματος και την αποθήκευσή του, ακολουθεί η διαδικασία της μετάφρασης (compilation). Ο μεταφραστής (compiler) του Quartus II αποτελείται από ένα σετ ανεξάρτητων εργαλείων που ελέγχουν και αναλύουν τον κώδικα VHDL ή το σχηματικό διάγραμμα για λάθη και δημιουργούν μία λογική έκφραση για κάθε λογική συνάρτηση του κυκλώματος, απεικονίζουν το σχέδιο σε μία προγραμματιζόμενη διάταξη (FPGA ή CPLD) της Altera και δημιουργούν αρχεία εξόδων για προσομοίωση λειτουργίας (simulation), χρονική ανάλυση (timing analysis), και προγραμματισμό των διατάξεων (device programming). Ο μεταφραστής αποτελείται από τα εργαλεία της Ανάλυσης και Σύνθεσης (Analysis and Synthesis), τον Fitter, τον Assembler και τον Timing Analyzer.

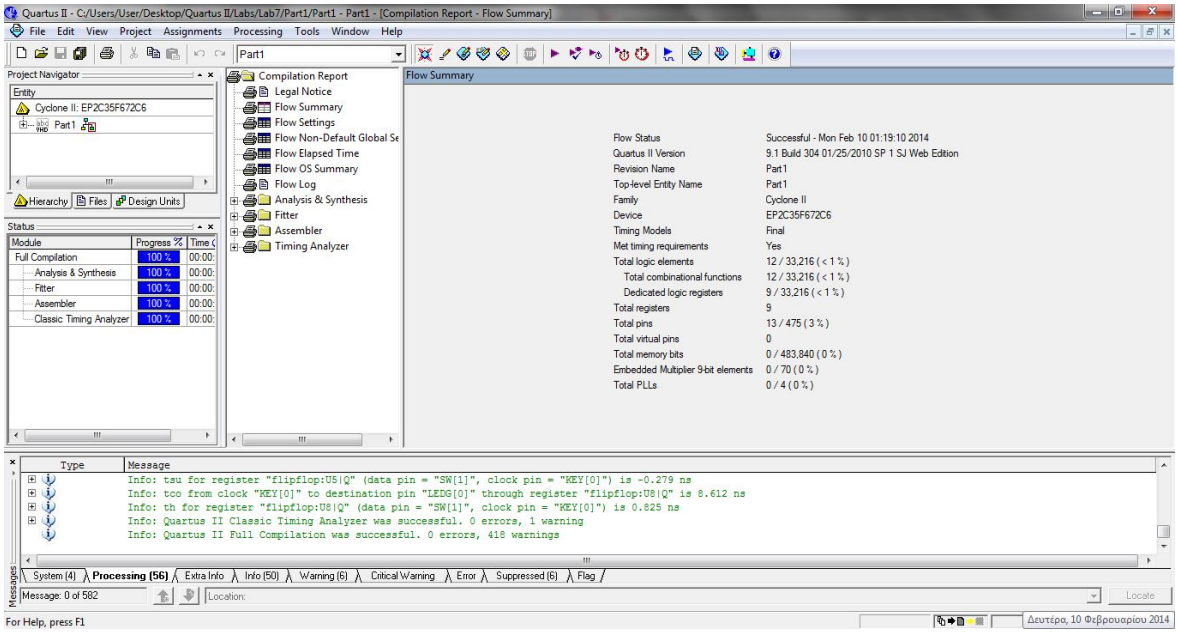
Στην περίπτωση εισαγωγής κώδικα VHDL, είναι απαραίτητη η μη ύπαρξη ορθογραφικών ή συντακτικών λαθών στο πρόγραμμα. Επειδή η πλήρης μετάφραση μπορεί να διαρκεί αρκετό χρόνο το Quartus-II δίνει τη δυνατότητα ανάλυσης του κώδικα και προσδιορισμού τυχόν λαθών εκτελώντας μόνο το πρώτο βήμα. Από το μενού Processing | Start | Start Analysis and Synthesis, η διαδικασία αναλαμβάνει τη μετάφραση του προγράμματός και μόλις ολοκληρωθεί 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. Μετά τις απαραίτητες διορθώσεις, η διαδικασία Start Analysis and Synthesis επαναλαμβάνεται. Πολλές φορές η διόρθωση ενός σφάλματος οδηγεί σε σημαντική ελάττωση του συνολικού αριθμού των σφαλμάτων.

Με την ολοκλήρωση διόρθωσης των σφαλμάτων προχωράμε σε πλήρη μετάφραση του κώδικα. Η διαδικασία ξεκινά από το μενού Processing| Compiler Tool και πατώντας το κουμπί Start στην καρτέλα που εμφανίζεται. Η πρόοδος της μετάφρασης παρατηρείται μέσω των εργαλείων του comiler (*Σχήμα 17*). Η πρόοδος των εργασιών φαίνεται επίσης και στο αριστερό μέρος της οθόνης.



Σχήμα 17 - Compiler Tool

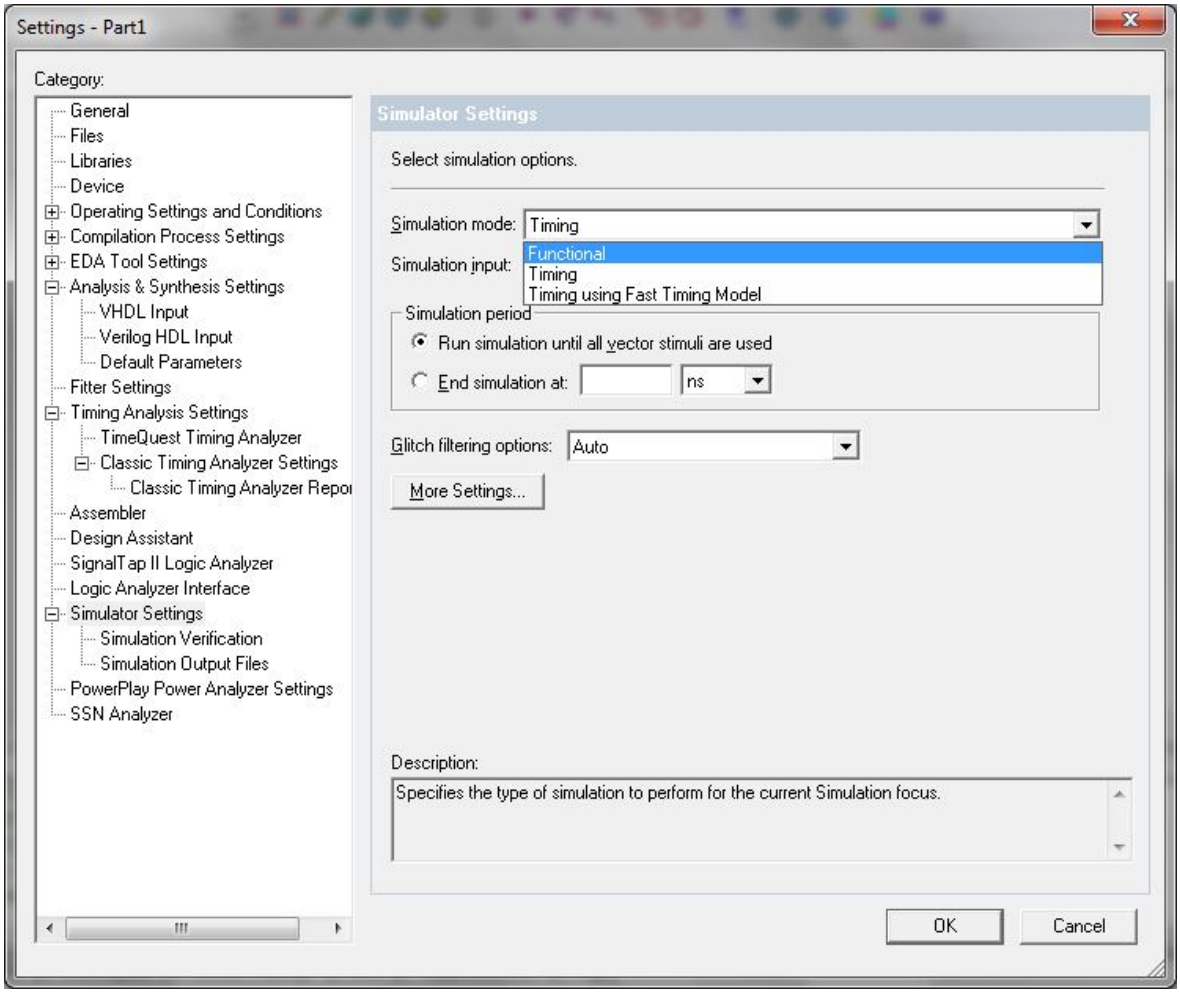
Όταν ο Compiler ολοκληρώσει τη μετάφραση, πατώντας το κουμπί Report, μπορούμε να πάρουμε πληροφοριακά και στατιστικά στοιχεία για τη διαδικασία της μετάφρασης του κυκλώματος. Η αναφορά (*Σχήμα 18*), μας ενημερώνει για τις απαιτήσεις σε λογικά στοιχεία (LEs), bits μνήμης και ακροδέκτες, που έχει η εφαρμογή μας.



Σχήμα 18 - Compilation Report - Flow Summary

3.8.4. Προσομοίωση (Simulation)

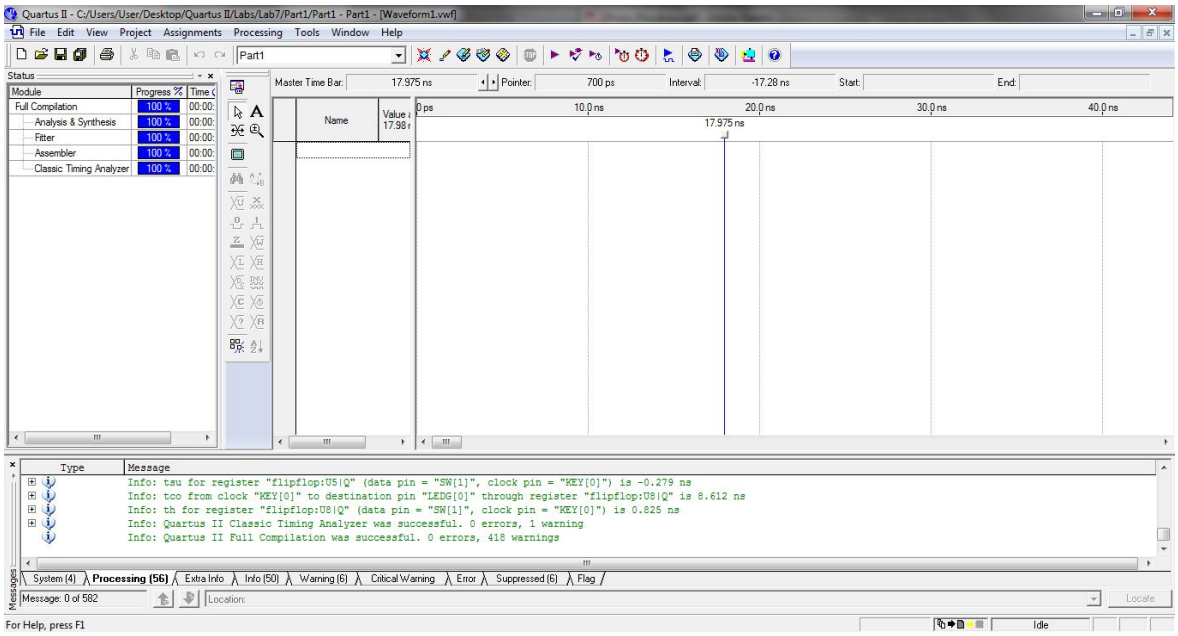
Όταν τελειώσει η διαδικασία της μετάφρασης επιτυχώς, ακολουθεί η διαδικασία της προσομοίωσης. Ο προκαθορισμένος (default) τύπος προσομοίωσης στο Quartus είναι προσομοίωση χρονισμών (timing). Σε περίπτωση που απαιτείται λειτουργική προσομοίωση (functional), καθορίζουμε τον επιθυμητό τύπο προσομοίωσης επιλέγοντας από το μενού Assignments | Settings και στην κατηγορία Simulator Settings αλλάζουμε το Simulation mode από Timing σε Functional. Στην ίδια καρτέλα θέτουμε τη συνολική διάρκεια της προσομοίωσης.



Σχήμα 19 - Ρυθμίσεις Προσομοίωσης

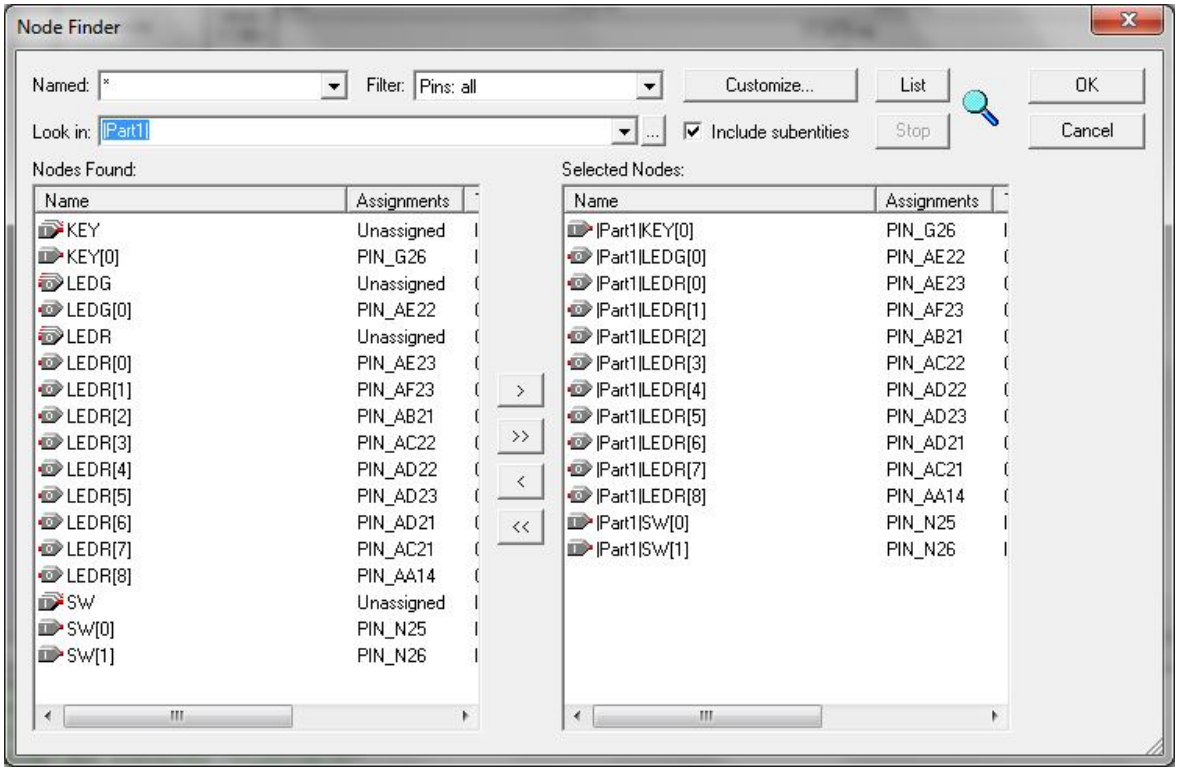
Για να προχωρήσουμε στην προσομοίωση πρέπει πρώτα να δημιουργηθεί το κατάλληλο αρχείο περιγραφής του κυκλώματος (netlist) και ακολούθως ο καθορισμός των παλμών εισόδου. Από το μενού Processing | Generate Functional Simulation Netlist δημιουργείται ένα αρχείο περιγραφής του κυκλώματος, στο οποίο περιγράφεται μόνο η λογική του με χρήση των διαθέσιμων βασικών δομικών στοιχείων και δεν λαμβάνονται υπόψη οι χρονικές καθυστερήσεις των στοιχείων και των διασυνδέσεών τους.

Η διαδικασία της προσομοίωσης γίνεται μέσα από τον Waveform Editor, με τον οποίο εισάγονται οι κατάλληλες κυματομορφές εισόδου (waveform vectors) που είναι απαραίτητες για την προσομοίωση. Ο Waveform Editor καλείται από την εισαγωγική καρτέλα, επιλέγοντας μενού File| New| Other Files| Vector Waveform File. Στην επιφάνεια εργασίας του Quartus εμφανίζεται το παράθυρο που φαίνεται στο *Σχήμα 20*.



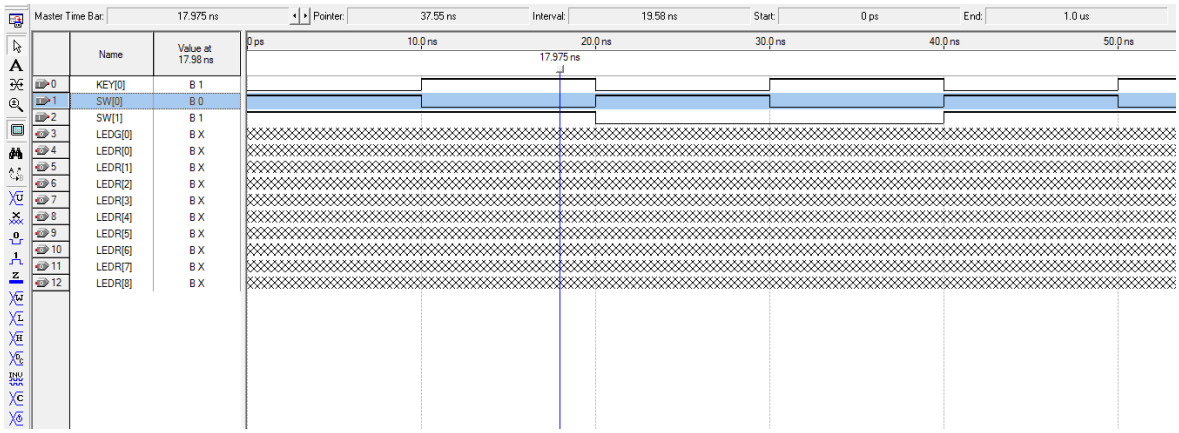
Σχήμα 20 - Waveform Editor

Κάνοντας διπλό κλικ κάτω από το Name, εμφανίζεται το παράθυρο Insert Node or Bus στο οποίο πατάμε το κουμπί Node Finder. Στο Node Finder επιλέγουμε το πλήκτρο List στο άνω-δεξιό τμήμα του ορθογωνίου για την εμφάνιση των ονομάτων των κόμβων του τρέχοντος έργου στο ορθογώνιο με το όνομα Nodes Found. Επιλέγουμε τις εισόδους και εξόδους του κυκλώματος, των οποίων τη λειτουργία επιθυμούμε να προσομοιώσουμε (*Σχήμα 21*).



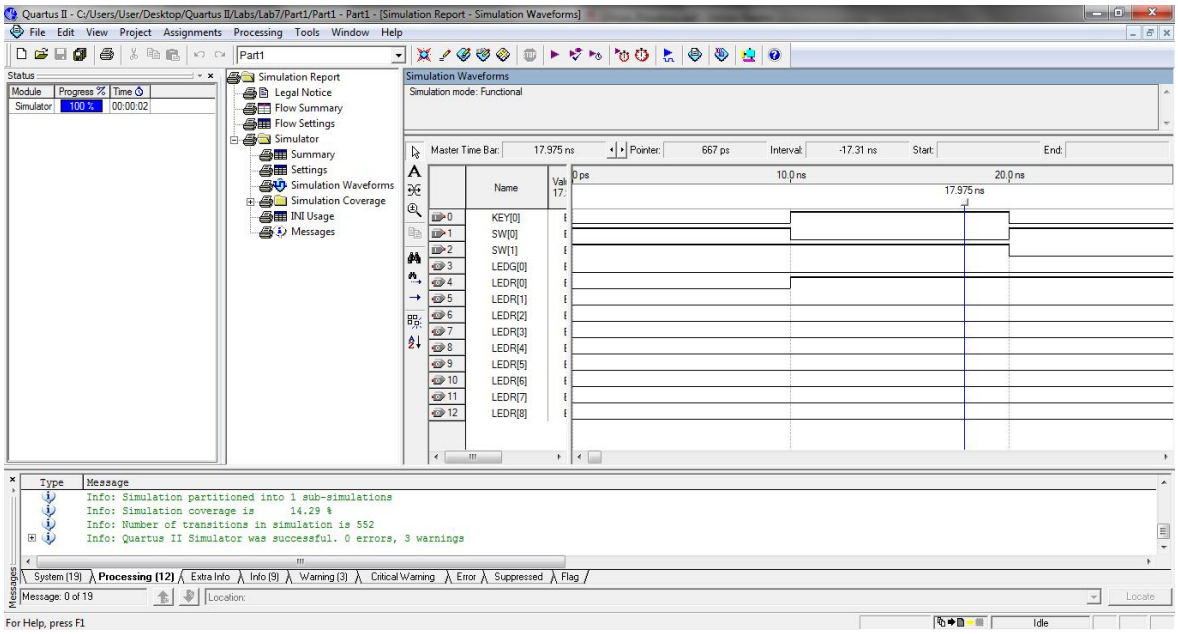
Σχήμα 21 - Επιλογή κόμβων

Οι κόμβοι που επιλέξαμε εμφανίζονται στο παράθυρο του Waveform Editor. Στο αριστερό μέρος της οθόνης εμφανίζεται 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, μπορούμε να δημιουργήσουμε τις κατάλληλες κυματομορφές εισόδου, θέτοντας τις τιμές των εισόδων σε 1 ή 0, για συγκεκριμένα χρονικά διαστήματα. Στη συνέχεια αποθηκεύουμε το αρχείο των διανυσμάτων εισόδου (Waveform vector file) με όνομα Part1.wvf. Το αρχείο κυματομορφών πρέπει να έχει το ίδιο όνομα με το όνομα του entity.



Σχήμα 22 - Καθορισμός τιμών σημάτων εισόδου

Για να εκτελεστεί η προσομοίωση του κυκλώματος επιλέγουμε από το μενού Processing | Start Simulation. 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 (*Σχήμα 23*). Στο σημείο αυτό, θα πρέπει να ελέγξουμε αν οι τιμές που παίρνουμε από την εκτέλεση της προσομοίωσης ανταποκρίνονται στις θεωρητικές τιμές που έχουμε βάση του πίνακα αληθείας του κυκλώματος μας.



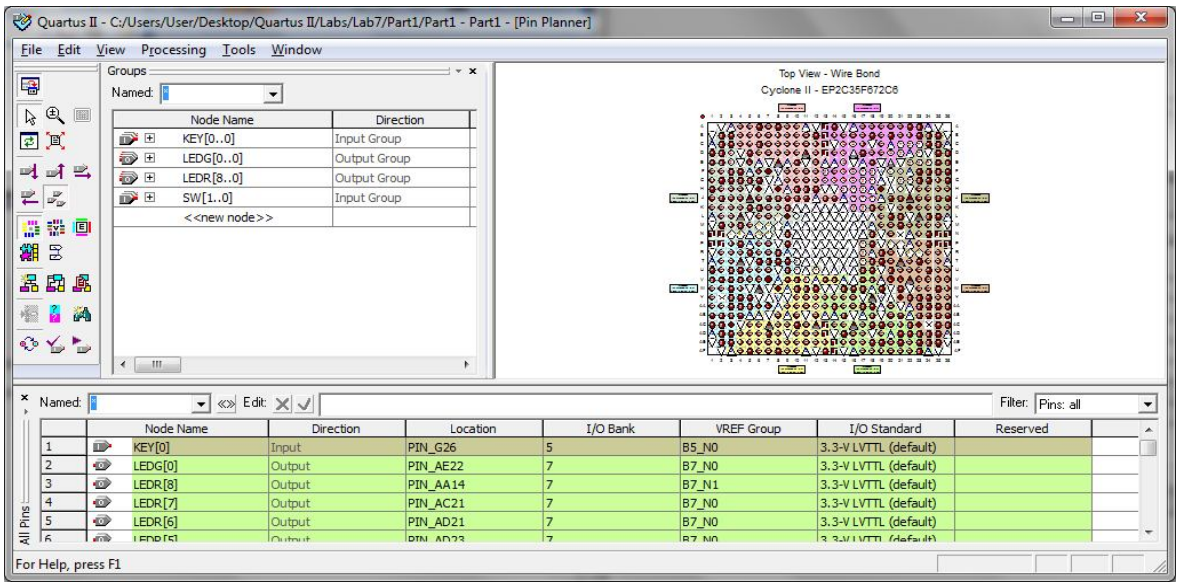
Σχήμα 23 - Προσομοίωση κυκλώματος

3.8.5. Ορισμός ακροδεκτών (Pin assignments)

Το επόμενο βήμα είναι να αντιστοιχήσουμε τις εισόδους και τις εξόδους του κυκλώματός με συγκεκριμένους ακροδέκτες της διάταξης που πρόκειται να προγραμματίσουμε. Έχουμε ήδη ορίσει ότι η διάταξή μας είναι ένα FPGA που ανήκει στην οικογένεια Cyclone II της ALTERA και συγκεκριμένα, η διάταξη EP2C35F672C6 .

Προκειμένου να ορίσουμε σε ποιους ακροδέκτες του τσιπ θα συνδεθούν οι είσοδοι και οι έξοδοι του κυκλώματος που σχεδιάσαμε, επιλέγουμε από το μενού Assignments | Pins ή εισάγουμε το αντίστοιχο αρχείο που περιέχει τους ακροδέκτες από την επιλογή Assignments | Import Assignements. Στο πεδίο Node Name εμφανίζονται τα ονόματα που έχουμε δώσει στους ακροδέκτες μας στο σχηματικό διάγραμμα. Διπλοπατώντας πάνω στο πεδίο Location εμφανίζεται η αρίθμηση όλων των ακροδεκτών της συγκεκριμένης διάταξης, όπου κάνουμε τις επιλογές.

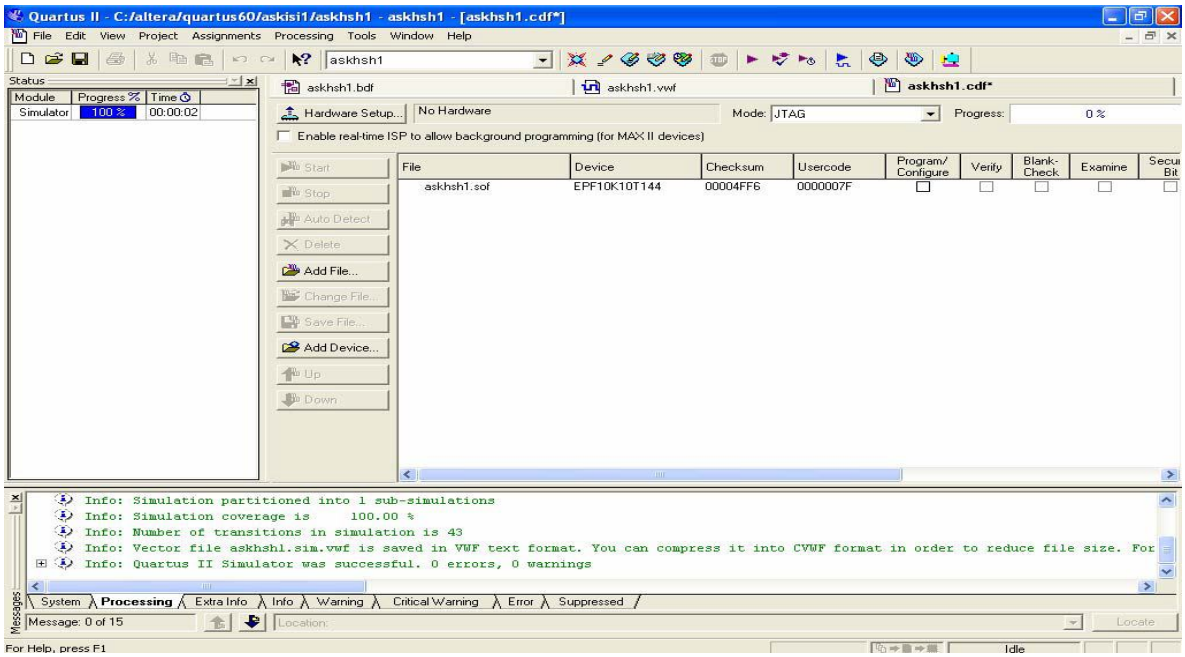
Οι επιλογές αυτές προκύπτουν από τα σχηματικά διαγράμματα του κυκλώματος που πρόκειται να προγραμματίσουμε (*Σχήμα 24*).



Σχήμα 24 - Pin Planner

3.8.6. Προγραμματισμός (διαμόρφωση) κυκλώματος

Με το πέρας της μετάφρασης (compilation) δημιουργείται ένα αρχείο (*.sof), το οποίο περιέχει όποιες πληροφορίες χρειάζονται για την διαμόρφωση του FPGA. Η διαμόρφωση γίνεται με τον Programmer, ο οποίος βρίσκεται στο μενού Tools | Programmer (*Σχήμα 25*). Εδώ γίνονται οι κατάλληλες επιλογές του υλικού (Hardware Setup) για την επιλογή του κατάλληλου κυκλώματος προγραμματισμού. Συνήθεις επιλογές είναι ο οδηγός BYTE-BLASTER για προγραμματισμό μέσω της παράλληλης θύρας ή ο USB-BLASTER για προγραμματισμό μέσω της θύρας USB. Πρέπει απαραίτητα να επιλεγεί το τελικό αρχείο διαμόρφωσης (.sof) που δημιουργήθηκε κατά το τελικό στάδιο της μετάφρασης, καθώς και η επιλογή Program/Configure.



Σχήμα 25 - Programmer

Κεφάλαιο 4

Εργαστηριακές εφαρμογές

4.1. Εισαγωγή

Τα εργαστήρια (labs) που ακολουθούν, αποτελούν εργαστηριακές εφαρμογές που διατίθενται από το Altera University, στα πλαίσια tutorials που αφορούν την εκμάθηση της γλώσσας VHDL.

Το σύνολο των ασκήσεων, αφορούν κυκλώματα η περιγραφή των οποίων γίνεται με τη χρήση της γλώσσας περιγραφής υλικού VHDL, καθώς και του εργαλείου σχεδίασης Quartus II της Altera.

Τα labs, αποτελούν ένα πολύ καλό οδηγό εκμάθησης της γλώσσας VHDL, δεδομένου ότι παρέχουν τη δυνατότητα – μέσω συγκεκριμένων οδηγιών – στην υλοποίηση απλών μέχρι και πιο σύνθετων συνδυαστικών και ακολουθιακών κυκλωμάτων.

Στο σύνολο των κυκλωμάτων που παρατίθενται παρακάτω, γίνεται μια σύντομη περιγραφή των κύριων χαρακτηριστικών τους, αναπαράσταση της λογικής σχεδίασης, του πίνακα αληθείας και του συμβόλου τους (όπου απαιτείται).

4.2. Διακόπτες (Switches) και Φωτεινοί ενδείκτες (Lights)

Το κύκλωμα περιέχει απλές εντολές αντιστοίχησης ενός αριθμού διακοπτών σε έναν αριθμό LEDs. Ως είσοδοι στο κύκλωμα χρησιμοποιούνται 18 εξωτερικοί διακόπτες (toggle switches) της DE2 αναπτυξιακής πλακέτας και ως έξοδοι 18 κόκκινοι φωτεινοί ενδείκτες (LEDs). Η είσοδος έχει όνομα SW, η έξοδος LEDR και είναι των 18 bits.

Για την περιγραφή του κυκλώματος με την γλώσσα VHDL, απαιτείται η γνώση των βασικών κανόνων συγγραφής της. Στην αρχή, ο κώδικας χωρίζεται σε τρία τουλάχιστον τμήματα. Στο πρώτο τμήμα δηλώνονται οι βιβλιοθήκες (library), στο δεύτερο τμήμα δηλώνεται η οντότητα (entity) όπου περιγράφονται τα σήματα εισόδου και εξόδου στο σύστημα και στο τρίτο τμήμα δηλώνεται η αρχιτεκτονική (architecture) όπου περιγράφεται η λογική του κυκλώματος.

Ο κώδικας VHDL είναι ο εξής:

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

-- Simple module that connects the SW switches to the LEDR lights
ENTITY Part1 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(17 DOWNTO 0); -- Toggle switches
          LEDR  : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)); -- Red LEDs
END Part1;

ARCHITECTURE Structure OF Part1 IS
BEGIN
    LEDR <= SW;
END Structure;
```

Πρόγραμμα 10 - Switches and lights

Στο πρώτο τμήμα δηλώνεται μόνο μία βιβλιοθήκη, η ieee.std_logic 1164.all, η οποία περιέχει τον ορισμό τύπου STD_LOGIC_VECTOR. Στο δεύτερο τμήμα υλοποιείται ένα ψηφιακό σύστημα με όνομα Part1, το οποίο έχει ακροδέκτες (σήματα) εισόδου τα SW_{17} ως SW_0 και ως ακροδέκτες εξόδου τα $LEDR_{17}$ ως $LEDR_0$, χωρίς να οριστεί ακόμη η λογική σχέση που θα έχουν τα σήματα εξόδου με αυτά της εισόδου. Η αρχιτεκτονική του ψηφιακού συστήματος ονομάζεται Structure και αναφέρεται στην οντότητα Part1. Στο τμήμα της αρχιτεκτονικής, γίνεται η αντιστοίχηση του κάθε διακόπτη SW σε κάθε LEDR.

Στο *Σχήμα 26*, παρουσιάζεται η μορφή του κυκλώματος. Η έννοια της σύνταξης SW[17…0] και LEDR[17…0] είναι ότι τα σήματα εισόδου SW και εξόδου LEDR αντιστοιχούν σε 18 bits, τα οποία ονομάζονται SW[17], SW[16], ….., SW[0] και LEDR[17], LEDR[16], …., LEDR[0] αντίστοιχα.

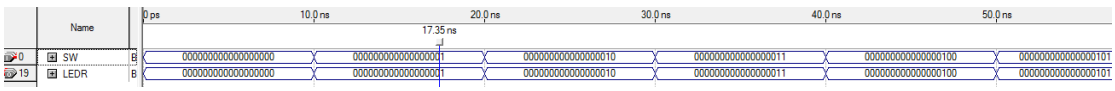
Σχήμα 26 - Κύκλωμα αντιστοίχησης 18 SWs σε 18 LEDRs

Για την υλοποίηση του κυκλώματος θα πρέπει να γίνει η αντιστοίχηση των ακροδεκτών (pin assignment), ώστε οι είσοδοι και οι έξοδοι που ορίστηκαν να αντιστοιχηθούν με ακροδέκτες του FPGA (*Πίνακας 10*).

SW[0]	Input	PIN_N25	LEDR[0]	Output	PIN_AE23
SW[1]	Input	PIN_N26	LEDR[1]	Output	PIN_AF23
SW[2]	Input	PIN_P25	LEDR[2]	Output	PIN_AB21
SW[3]	Input	PIN_AE14	LEDR[3]	Output	PIN_AC22
SW[4]	Input	PIN_AF14	LEDR[4]	Output	PIN_AD22
SW[5]	Input	PIN_AD13	LEDR[5]	Output	PIN_AD23
SW[6]	Input	PIN_AC13	LEDR[6]	Output	PIN_AD21
SW[7]	Input	PIN_C13	LEDR[7]	Output	PIN_AC21
SW[8]	Input	PIN_B13	LEDR[8]	Output	PIN_AA14
SW[9]	Input	PIN_A13	LEDR[9]	Output	PIN_Y13
SW[10]	Input	PIN_N1	LEDR[10]	Output	PIN_AA13
SW[11]	Input	PIN_P1	LEDR[11]	Output	PIN_AC14
SW[12]	Input	PIN_P2	LEDR[12]	Output	PIN_AD15
SW[13]	Input	PIN_T7	LEDR[13]	Output	PIN_AE15
SW[14]	Input	PIN_U3	LEDR[14]	Output	PIN_AF13
SW[15]	Input	PIN_U4	LEDR[15]	Output	PIN_AE13
SW[16]	Input	PIN_V1	LEDR[16]	Output	PIN_AE12
SW[17]	Input	PIN_V2	LEDR[17]	Output	PIN_AD12

Πίνακας 10 - Αντιστοίχιση ακροδεκτών

Η προσομοίωση του κυκλώματος παρουσιάζεται στο *Σχήμα 27*. Στην προσομείωση φαίνεται ότι, όταν θέτουμε κάποιον διακόπτη εισόδου σε λογική κατάσταση 1, τίθεται στην ίδια κατάσταση και η αντίστοιχη έξοδος (π.χ. όταν το SW[2]=’1’ τότε και το LEDR[2]=’1’).

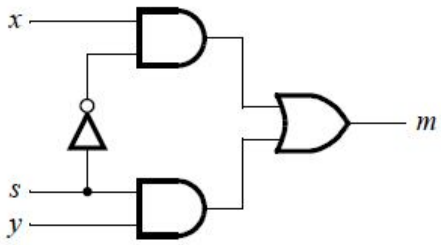


Σχήμα 27 - Προσομοίωση κυκλώματος διακοπτών LEDs

Εφόσον η προσομοίωση του κυκλώματος ανταποκρίνεται στα επιθυμητά αποτελέσματα, διαμορφώνουμε τη διάταξη FPGA χρησιμοποιώντας τα αρχεία διαμόρφωσης του κυκλώματος (Part1.sof) που έχουν δημιουργηθεί κατά τη διάρκεια της συμβολομετάφρασης (compiling).

4.3. Πολυπλέκτες (Multiplexers)

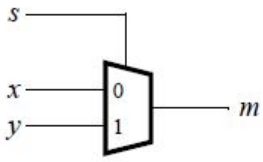
Ο πολυπλέκτης είναι ένα κύκλωμα το οποίο αποτελείται από έναν αριθμό εισόδων, από κάποιες εισόδους επιλογής και από μια έξοδο. Ο αριθμός των διακοπτών επιλογής βρίσκεται από τον τύπο 2^v=x, όπου x ο αριθμός των εισόδων και v ο αριθμός των διακοπτών επιλογής. Η λειτουργία ενός πολυπλέκτη είναι η δημιουργία μιας εξόδου, η οποία είναι ταυτόσημη με μία από τις εισόδους του. Η επιλογή για το ποια είσοδος θα γίνει έξοδος γίνεται από τους διακόπτες επιλογής.



α) Λογικό Κύκλωμα

s	m
0	x
1	y

β) Πίνακας Αλήθειας



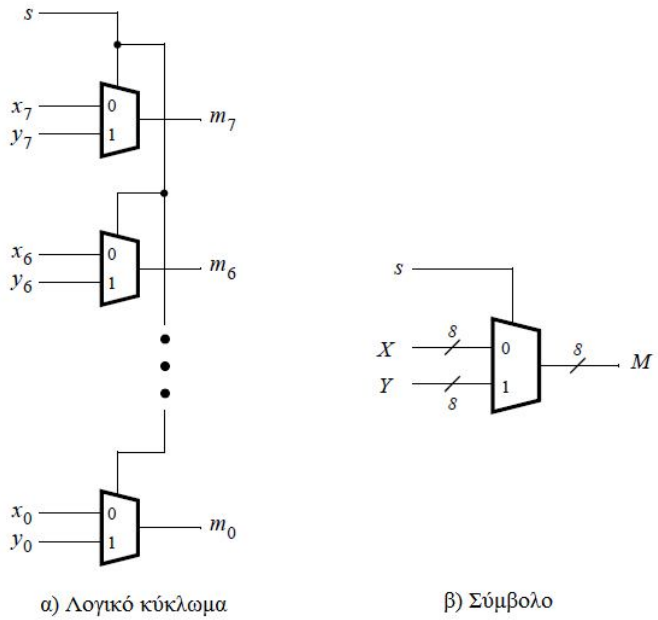
γ) Σύμβολο

Σχήμα 28 - Πολυπλέκτης 2:1

Ο πολυπλέκτης 2 προς 1 αποτελείται από δύο εισόδους (x=2) και έναν διακόπτη επιλογής (v=1) αφού έχουμε 2^1=2. Στο *Σχήμα 28*, φαίνεται το λογικό κύκλωμα, ο πίνακας αληθείας και το σύμβολο ενός πολυπλέκτη 2:1. Ανάλογα με την τιμή του διακόπτη επιλογής (s) λαμβάνεται ως έξοδο (m), η αντίστοιχη είσοδο. Αν η τιμή στην είσοδο επιλογής s = 0, η τιμή της εξόδου m είναι ίση με την τιμή της εισόδου x, ενώ αν s = 1 η έξοδος είναι ίση με το y.

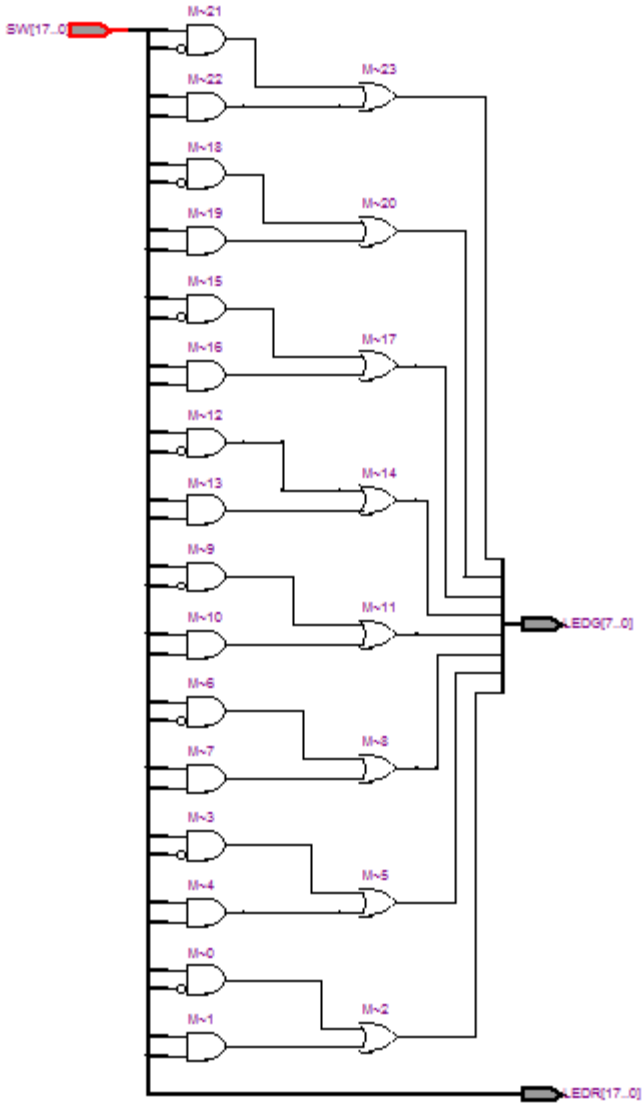
4.3.1. Πολυπλέκτης 2:1 (Multiplexer 2-to-1) 8 bits

Η λογική της σχεδίασης ενός πολυπλέκτη 8 bits δεν διαφέρει από αυτή του απλού πολυπλέκτη:. δύο είσοδοι, μία έξοδο και μια γραμμής επιλογής. Η διαφορά έγκειται ότι τα κανάλια του δεν είναι του 1 bit, αλλά των 8 bits, δηλαδή κάθε είσοδος του πολυπλέκτη καθώς και η έξοδός του έχουν μέγεθος 8 bits. Στο *Σχήμα 29*, φαίνεται το λογικό κύκλωμα και το σύμβολο ενός πολυπλέκτη 8 bits. Το κύκλωμα έχει δύο εισόδους X και Y και μία έξοδο M. Ανάλογα με την τιμή του διακόπτη επιλογής (s) λαμβάνεται ως έξοδο (M), η αντίστοιχη είσοδο. Αν s = 0 τότε M = X, ενώ αν s = 1 τότε M = Y.



Σχήμα 29 - Πολυπλέκτης 8 bits

Ο διακόπτης επιλογής s θα είναι ο διακόπτης SW_{17} (1 bit). Η είσοδος X θα γραφεί ως διάνυσμα SW_{7-0}, η άλλη είσοδος Y ως διάνυσμα SW_{15-8} και η έξοδος M ως διάνυσμα $LEDG_{7-0}$. Οι διακόπτες SW θα συνδεθούν στους κόκκινους φωτεινούς ενδείκτες LEDR ενώ η έξοδος M θα συνδεθεί στους πράσινους ενδείκτες $LEDG_{7-0}$.



Σχήμα 30 - Πολυπλέκτης 2:1 8 bits

Ο κώδικας VHDL χωρίζεται σε τρία τμήματα. Η βιβλιοθήκη που χρησιμοποιείται είναι η ieee.std_logic_1164.all, καθώς περιέχεται σε αυτήν ο τύπος STD_LOGIC_VECTOR που χρησιμοποιείται για τα σήματα 8 bits. Η οντότητα έχει όνομα Part2 με εισόδους SW και εξόδο τα LEDR και LEDG.

Στο σώμα της αρχιτεκτονικής δίνουμε την ονομασία structural. Μέσα στο σώμα της αρχιτεκτονικής ορίζουμε τέσσερα σήματα (Sel, X, Y, M), όπου Sel είναι ο διακόπτης επιλογής s. Τα σήματα αυτά που στην ουσία είναι εσωτερικά καλώδια του κυκλώματος είναι ίδιου τύπου και ίδιου μεγέθους με τις εισόδους αλλά και με την έξοδο. Ο τρόπος περιγραφής της σχέσης μεταξύ εισόδων και εξόδου λέγεται δομικός, και αυτό γιατί η περιγραφή αυτή βασίζεται στα δομικά στοιχεία που απαρτίζουν το κύκλωμα. Τα στοιχεία αυτά δεν είναι άλλα από τις λογικές πύλες AND, OR και NOT.

Ο πολυπλέκτης μπορεί να περιγραφεί από την ακόλουθη δήλωση VHDL:

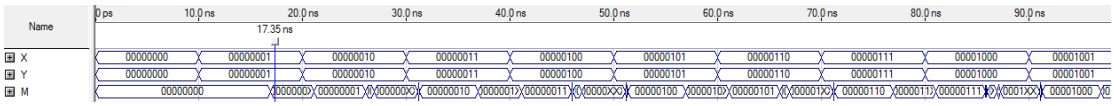
m <= (NOT (s) AND x) OR (s AND y);

Ο κώδικας VHDL περιλαμβάνει οκτώ (8) δηλώσεις εκχώρησης όπως η προηγούμενη για να περιγράψουν το κύκλωμα.

```
-- Implements eight-bit wide 2-to-1 multiplexers.
-- Inputs:  SW17 represent the s input and selects either X or Y to drive the output LEDs
--          SW7-0 represent the 8-bit input X, SW15-8 represent input Y
-- Outputs: LEDR17-0 show the states of the switches
--          LEDG7-0 shows the outputs of the multiplexers

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Simple module that connects the SW switches to the LEDR lights
ENTITY Part2 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(17 DOWNTO 0); -- Toggle switches
          LEDR  : OUT STD_LOGIC_VECTOR(17 DOWNTO 0); -- Red LEDs
          LEDG  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- Green LEDs
END Part2;

ARCHITECTURE Structure OF Part2 IS
    SIGNAL Sel      : 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);
    Sel <= SW(17);
    M(0) <= (NOT(Sel) AND X(0)) OR (Sel AND Y(0));
    M(1) <= (NOT(Sel) AND X(1)) OR (Sel AND Y(1));
    M(2) <= (NOT(Sel) AND X(2)) OR (Sel AND Y(2));
    M(3) <= (NOT(Sel) AND X(3)) OR (Sel AND Y(3));
    M(4) <= (NOT(Sel) AND X(4)) OR (Sel AND Y(4));
    M(5) <= (NOT(Sel) AND X(5)) OR (Sel AND Y(5));
    M(6) <= (NOT(Sel) AND X(6)) OR (Sel AND Y(6));
    M(7) <= (NOT(Sel) AND X(7)) OR (Sel AND Y(7));
    LEDG(7 DOWNTO 0) <= M;
END Structure;
```

Πρόγραμμα 11 - 8-bit Multiplexer 2-to-1

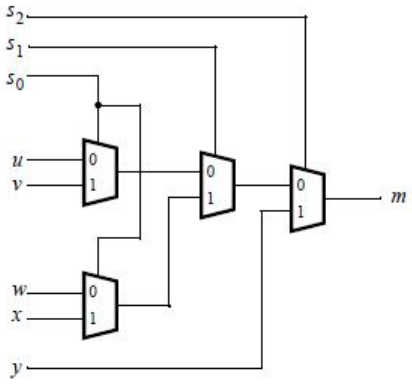
Στο *Σχήμα 31*, παρουσιάζεται η προσομοίωση του πολυπλέκτη:



Σχήμα 31 - Προσομοίωση Πολυπλέκτη 8 bits

4.3.2. Πολυπλέκτης 5:1 (Multiplexer 5-to-1) 3 bits

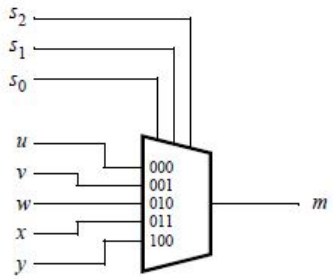
Στο *Σχήμα 32* παρουσιάζεται το λογικό κύκλωμα, ο πίνακας αληθείας και το σύμβολο ενός πολυπλέκτη 5:1. Το κύκλωμα δείχνει πώς μπορεί να υλοποιηθεί ο πολυπλέκτης 5:1 χρησιμοποιώντας τέσσερις πολυπλέκτες 2:1. Το κύκλωμα χρησιμοποιεί τις εισόδους επιλογής $s_2s_1s_0$ μεγέθους 3 bits.



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

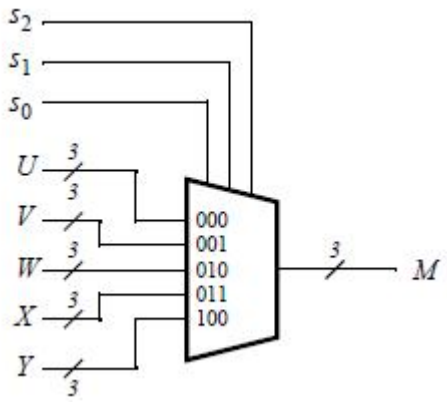
β) Πίνακας αληθείας



γ) Σύμβολο

Σχήμα 32 - Πολυπλέκτης 5:1

Στο *Σχήμα 33* παρουσιάζεται το σύμβολο ενός πολυπλέκτη 5:1 3 bits.



Σχήμα 33 - Πολυπλέκτης 5:1 3 bits

Ο κώδικας VHDL για την υλοποίηση του πολυπλέκτη είναι ο εξής:

```
-- Implements a three-bit wide 5-to-1 multiplexer.
-- Inputs:  SW17-15 connect its select inputs, one group from U to Y
--          SW14-0 represent data in five 3-bit groups, U to Y
-- Outputs: LEDR17-0 show the states of the switches
--          LEDG2-0 displays the selected group

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Simple module that connects the SW switches to the LEDR lights
ENTITY Part3 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(17 DOWNTO 0); -- Toggle switches
          LEDR  : OUT STD_LOGIC_VECTOR(17 DOWNTO 0); -- Red LEDs
          LEDG  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)); -- Green LEDs
END Part3;

ARCHITECTURE Structure OF Part3 IS
    SIGNAL m_0, m_1, m_2 : STD_LOGIC_VECTOR(1 TO 3);
    -- m_0 is used for 3 intermediate multiplexers to produce the
    -- 5-to-1 multiplexer M(0), m_1 for M(1), m_2 for M(2)
    SIGNAL S, U, V, W, X, Y, M : STD_LOGIC_VECTOR(2 DOWNTO 0);
    -- M is the 3-bit 5-to-1 multiplexer
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;

    -- 5-to-1 multiplexer for bit 0
    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)); -- 5-to-1 multiplexer output

    -- 5-to-1 multiplexer for bit 1
    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)); -- 5-to-1 multiplexer output

    -- 5-to-1 multiplexer for bit 2
    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)); -- 5-to-1 multiplexer output

    LEDG(2 DOWNTO 0) <= M;
END Structure;
```

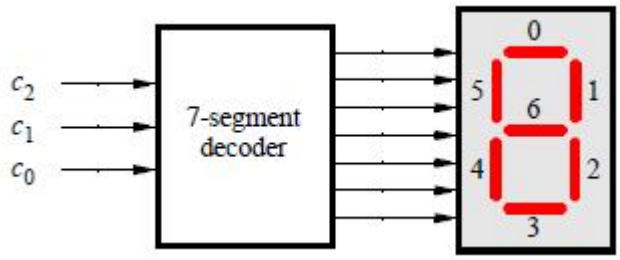
Πρόγραμμα 12 - 3 bit Multiplexer 5-to-1

4.4. Αποκωδικοποιητές (Decoders) – Κωδικοποιητές (Encoders)

Ένας αποκωδικοποιητής είναι ένα κύκλωμα που χρησιμοποιείται για την αποκωδικοποίηση μιας κωδικοποιημένης πληροφορίας, μετατρέποντάς την σε κατάλληλη μορφή. Αντιθέτως, ένας κωδικοποιητής είναι ένα συνδυαστικό κύκλωμα που παράγει μια κωδικοποιημένη πληροφορία.

4.4.1. Αποκωδικοποιητής επτά τομέων (7-segment decoder)

Το *Σχήμα 34* δείχνει μία μονάδα αποκωδικοποιητή 7 τομέων που έχει την είσοδο $c_2c_1c_0$ 3 bit. Ο αποκωδικοποιητής παράγει επτά εξόδους που χρησιμοποιούνται για την απεικόνιση ενός χαρακτήρα σε ενδείκτη επτά τομέων.



Σχήμα 34 - Αποκωδικοποιητής 7 τομέων

Ο *Πίνακας 11* παραθέτει τους χαρακτήρες που θα πρέπει να εμφανίζονται για κάθε τιμή των $c_2c_1c_0$. Ο πίνακας περιλαμβάνει πέντε χαρακτήρες (με τον «κενό» χαρακτήρα για τους κωδικούς 100 - 111).

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

Πίνακας 11 - Κωδικοί χαρακτήρων

Ο κώδικας VHDL εφαρμόζει λογικές συνθήκες που αντιπροσωπεύουν κυκλώματα απαραίτητα για την ενεργοποίηση καθενός από τους 7 τομείς. Χρησιμοποιούνται απλές δηλώσεις εκχώρησης VHDL για τον καθορισμό κάθε λογικής λειτουργίας με τη χρήση εκφράσεων boolean.

Το κύκλωμα αποτελείται από μία είσοδο SW μεγέθος 3 bits, μία έξοδο LEDR 3 bits και μία έξοδο HEX0 μεγέθους 7 bits. Το κάθε bit της εξόδου αντιστοιχεί σε ένα led του ενδείκτη επτά τομέων.

```
-- Implements a circuit that can display five characters on a 7-segment display.
-- Inputs:  SW2-0 selects the letter to display.
--          The characters codes are:
--          SW 2 1 0 Characters
--          -------------------
--             0 0 0    'H'
--             0 0 1    'E'
--             0 1 0    'L'
--             0 1 1    'O'
--             1 0 0    ' ' Blank
--             1 0 1    ' ' Blank
--             1 1 0    ' ' Blank
--             1 1 1    ' ' Blank
--
-- Outputs: LEDR2-0 show the states of the switches
--          HEX0 displays the selected character

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Part4 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- Toggle switches
          LEDR  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0); -- Red LEDs
          HEX0  : OUT STD_LOGIC_VECTOR(0 TO 6)); -- 7-seg display
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);
    --
    --      0
    --     ---
    --    |   |
    --  5|   |1
    --    | 6 |
    --     ---
    --    |   |
    --  4|   |2
    --    |   |
    --     ---
    --      3
    --
    -- The following equations describe display functions in (inverted) cannonical SOP form
    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 13 - 7-segment decoder

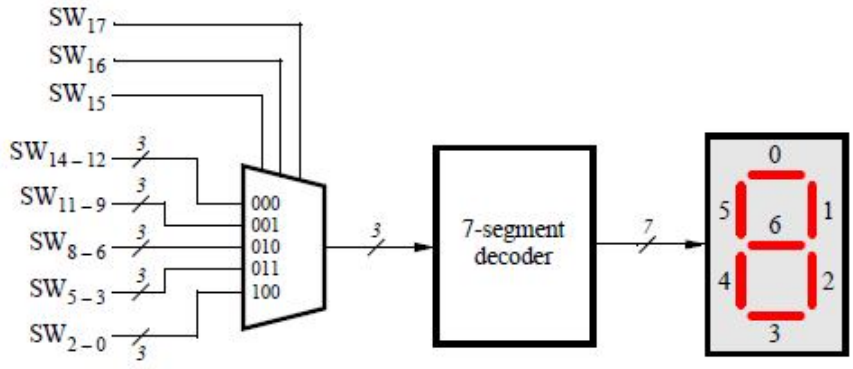
Η αντιστοίχηση των ακροδεκτών (pin assignment) παρουσιάζεται στον *Πίνακα 12*. Η είσοδος γίνεται με τη βοήθεια τεσσάρων διακοπτών της αναπτυξιακής πλακέτας SW_{2-0} και η έξοδος σε απεικόνιση επτά τομέων.

SW[0]	Input	PIN_N25	HEX0[0]	Output	PIN_AF10
SW[1]	Input	PIN_N26	HEX0[1]	Output	PIN_AB12
SW[2]	Input	PIN_P25	HEX0[2]	Output	PIN_AC12
LEDR[0]	Output	PIN_AE23	HEX0[3]	Output	PIN_AD11
LEDR[1]	Output	PIN_AF23	HEX0[4]	Output	PIN_AE11
LEDR[2]	Output	PIN_AB21	HEX0[5]	Output	PIN_V14
			HEX0[6]	Output	PIN_V13

Πίνακας 12 - Αντιστοίχιση ακροδεκτών

4.4.2. Αποκωδικοποιητής 7-segment

Το κύκλωμα που παρουσιάζεται στο *Σχήμα 35*, χρησιμοποιεί έναν πολυπλέκτη 5:1 3 bits για να επιλέξει και να απεικονίσει έναν από πέντε χαρακτήρες ως ένδειξη 7 τομέων.



Σχήμα 35 - Κύκλωμα επιλογής και ένδειξης 1:5 χαρακτήρων

Το κύκλωμα που υλοποιείται εμφανίζει διαφορετικές λέξεις με 5 χαρακτήρες σε πέντε ενδείκτες 7 τομεών. Οι πολυπλέκτες συνδέονται με τους χαρακτήρες με έναν τρόπο που να επιτρέπει την περιστροφή μιας λέξης σε ολόκληρο των ενδείκτη από δεξιά προς τα αριστερά, όπως αλλάζουν οι γραμμές επιλογής του πολυπλέκτη μέσω της ακολουθίας που φαίνεται στον *Πίνακα 13*.

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 13 - Περιστροφή της λέξης HELLO σε πέντε ενδείξεις

Ο κώδικας VHDL είναι ο εξής:

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

-- Implements a circuit that can display different 5-letter words on five 7-segment displays.
-- The character selected for each display is chosen by a multiplexer. These multiplexers are
-- connected to the characters in a way that allows a word to be rotated across the displays
-- from right-to-left as the multiplexer select lines are changed through the sequence 000,
-- 001, 010, 011, 100, 000, etc. Using the four characters H, E, L, O, the displays can
-- scroll any 5-letter word using these letters, such as "HELLO", as follows:
--
-- SW 17 16 15 Displayed characters
--     0  0  0 HELLO
--     0  0  1 ELLOH
--     0  1  0 LLOHE
--     0  1  1 LOHEL
--     1  0  0 OHELL
--
-- Inputs:  SW17-15 provide the multiplexer select lines
--          SW14-0 provide five 3-bit codes used to select characters
-- Outputs: LEDR shows the states of the switches
--          HEX4-HEX0 displays the characters (HEX7-HEX5 are set to "blank")

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"; -- Used to blank a 7-seg display (see module char_7seg)

    -- Instantiate mux_3bit_5to1 (S, U, V, W, X, Y, M)
    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);
```

```
    -- Instantiate char_7seg (C, Display)
    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;

-- Implements a 3-bit wide 5-to-1 multiplexer
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); -- Intermediate multiplexers
BEGIN
    -- 5-to-1 multiplexer for bit 0
    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)); -- 5-to-1 multiplexer output

    -- 5-to-1 multiplexer for bit 1
    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)); -- 5-to-1 multiplexer output

    -- 5-to-1 multiplexer for bit 2
    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)); -- 5-to-1 multiplexer output
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- Converts 3-bit input code on C2-0 into 7-bit code that produces
-- a character on a 7-segment display. The conversion is defined by:
-- C 2 1 0  Character
-- -----------------
--   0 0 0     'H'
--   0 0 1     'E'
--   0 1 0     'L'
--   0 1 1     'O'
--   1 0 0     ' ' Blank
--   1 0 1     ' ' Blank
--   1 1 0     ' ' Blank
--   1 1 1     ' ' Blank
--
-- Codes 100, 101, 110 are not used
--
ENTITY char_7seg IS
    PORT (C       : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Display : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;
--
--    0
--   ---
--  |   |
-- 5|   |1
--  | 6 |
--   ---
--  |   |
-- 4|   |2
--  |   |
--   ---
--    3
--
```

```
ARCHITECTURE Behavior OF char_7seg IS
BEGIN
-- The following equations describe display functions in (inverted) cannonical SOP 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 14 - 7-segment decoder

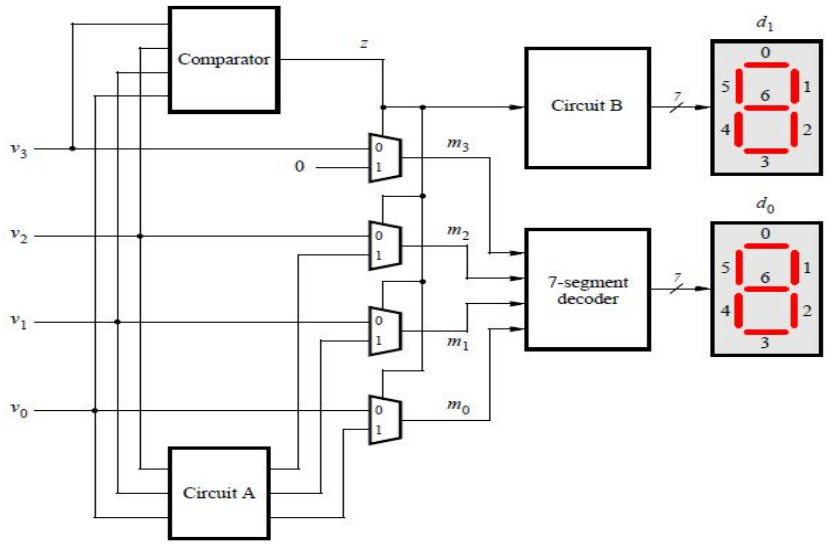
4.4.3. Κωδικοποιητής BCD σε δεκαδικό 7 segment

Ένας κωδικοποιητής κώδικα BCD (binary coded decimal) μετατρέπει έναν αριθμό ο οποίος είναι γραμμένος σε δυαδικό κώδικα BCD στον αντίστοιχο αριθμό του δεκαδικό 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 14 - Τιμές μετατροπής δυαδικού σε δεκαδικό

Ο μερικός σχεδιασμός του κυκλώματος παρουσιάζεται στο *Σχήμα 36*. Το κύκλωμα περιλαμβάνει έναν συγκριτή (comparator), η λειτουργία του οποίου είναι η σύγκριση δύο αριθμών, τους οποίους δέχεται ως είσοδο, ως προς το μέγεθος.



Σχήμα 36 - Μετατροπέας δυαδικού σε δεκαδικό

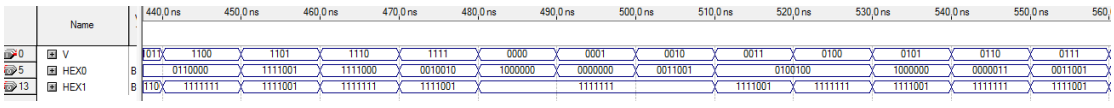
Ο κώδικας VHDL περιλαμβάνει τον συγκριτή, πολυπλέκτες, το κύκλωμα A, το κύκλωμα B, καθώς και τον αποκωδικοποιητή 7 τομέων. Θα διαθέτει μία είσοδο V εύρους 4 bits, μία έξοδο M εύρους 4 bits και μία έξοδο z. Ο κώδικας περιλαμβάνει λογικές εκφράσεις χωρίς καμία δήλωση IF-ELSE, CASE ή παρόμοιες δηλώσεις. Ο συγκριτής ελέγχει αν η τιμή της εισόδου V είναι μεγαλύτερη του 9 και χρησιμοποιεί την έξοδο του συγκριτή για τον έλεγχο του ενδείκτη 7 τομέων. Αν είναι μεγαλύτερη τότε στην έξοδο παίρνουμε την τιμή 1, αλλιώς παίρνουμε το κενό (blank). Οι διακόπτες $SW_{3\text{-}0}$ θα συνδεθούν στην είσοδο V και οι ενδείκτες HEX1 και HEX0 στον 7-seg για να απεικονήσουν τις τιμές των δεκαδικών ψηφίων.

```
-- Bcd-to-decimal converter

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Part2 IS
    PORT (SW        : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- Toggle switches
          HEX1, HEX0: OUT STD_LOGIC_VECTOR(0 TO 6)); -- 7-segs
END Part2;

ARCHITECTURE Structure OF part2 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;

    -- Circuit A
    z <= (V(3) AND V(2)) OR (V(3) AND V(1));

    -- Circuit B
    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));
```

```
    -- Multiplexers
    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
    Circuit_D: bcd7seg PORT MAP (M, HEX0);

    -- Circuit C
    HEX1 <= ('1' & NOT(z) & NOT(z) & "1111"); -- Display a blank or the digit 1
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
    --
    --     0
    --    ---
    --   |   |
    -- 5|   |1
    --   | 6 |
    --    ---
    --   |   |
    -- 4|   |2
    --   |   |
    --    ---
    --     3
    --
    -- B    H
    -- ----------
    -- 0 0000001;
    -- 1 1001111;
    -- 2 0010010;
    -- 3 0000110;
    -- 4 1001100;
    -- 5 0100100;
    -- 6 1100000;
    -- 7 0001111;
    -- 8 0000000;
    -- 9 0001100;
    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 15 - BCD to decimal converter

Στο *Σχήμα 37*, παρουσιάζεται η σωστή λειτουργία του κωδικοποιητή, αφού οι τιμές στις εξόδους ανταποκρίνονται στις τιμές του πίνακα αληθείας.



Σχήμα 37 - Λειτουργική προσωμοίωση κωδικοποιητή BCD

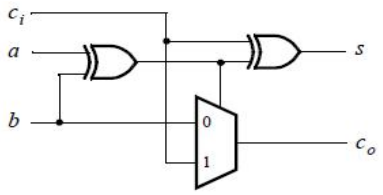
4.5. Αθροιστές (Adders)

Ο αθροιστής είναι ένα κύκλωμα με εισόδους δύο ή περισσότερα σήματα και έξοδο το άθροισμά τους. Βασικά ψηφιακά κυκλώματα αθροιστών, με εφαρμογή στους υπολογιστές, είναι ο ημιαθροιστής (half adder) και ο πλήρης αθροιστής (full adder). Ο ημιαθροιστής είναι ένα βασικό συνδυαστικό κύκλωμα που εκτελεί την πρόσθεση δύο δυαδικών αριθμών. Ο πλήρης αθροιστής είναι το συνδυαστικό κύκλωμα που εκτελεί την πρόσθεση τριών δυαδικών αριθμών και, συγκεκριμένα, δύο σημαντικών και ενός κρατουμένου, το οποίο ενδέχεται να έχει παραχθεί από προηγούμενη άθροιση.

4.5.1. Πλήρης αθροιστής

Ο πλήρης αθροιστής μπορεί να σχεδιαστεί με τη βοήθεια δύο ημιαθροιστών. Η ανάγκη της ύπαρξης ενός κυκλώματος το οποίο θα προσθέτει αριθμούς με πολλά bits, οδηγεί στη δημιουργία ενός κυκλώματος το οποίο εκτός από τους δύο αιρθμούς που προσθέτονται, παρέχει την δυνατότητα για την ταυτόχρονη πρόσθεση και του κρατούμενο c_i, το οποίο προέρχεται από πρόσθεση προηγούμενη τάξης. Αυτό το κύκλωμα λέγεται πλήρης αθροιστής.

Το λογικό διάγραμμα, το σύμβολο και ο πίνακας αληθείας του πλήρη αθροιστή, δίνονται στο *Σχήμα 38*. Όπως φαίνεται, το κύκλωμα έχει τρεις εισόδους (c_i, a, b), όπου c_i είναι το κρατούμενο της προηγούμενης πρόσθεσης, και δύο εξόδους (s, c_o) όπου s είναι το άθροισμα και c_o το κρατούμενο από την εκτέλεση της πρόσθεσης.



α) Λογικό κύκλωμα

β) Σύμβολο

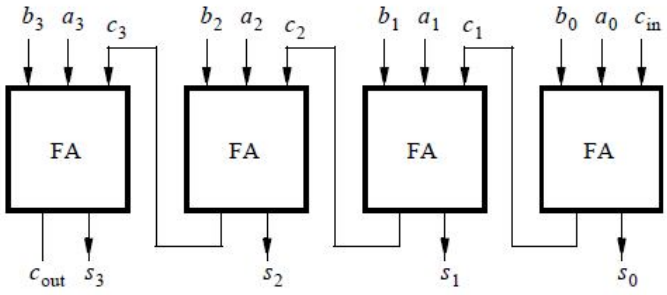
b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

γ) Πίνακας αληθείας

Σχήμα 38 - Πλήρης Αθροιστής

Ο πλήρης αθροιστής στην πιο απλή μορφή του, έχει εύρος εισόδων/εξόδων 1 bit μόνο. Αποτελείται από τρεις εισόδους STD_LOGIC εκ των οποίων η μία (c_i) λειτουργεί για το κρατούμενο εισόδου. Στην έξοδο του αθροιστή παίρνουμε δύο εξόδους εκ των οποίων η μία δίνει το άθροισμα των a, b (έξοδος s) και η άλλη το κρατούμενο της εξόδου (c_o). Η περιγραφή του κυκλώματος γίνεται με βάση τα δομικά στοιχεία που το απαρτίζουν, πύλες XOR, AND και OR.

Ο πλήρης αθροιστής που περιγράφεται έχει εύρος εισόδων/εξόδων των 4 bits. Σαν είδοδο δέχεται δύο αριθμούς των 4 bits και ένα κρατούμενο του 1 bit. Στην έξοδο το άθροισμα των αριθμών είναι και αυτό μεγάθους 4 bits ενώ το κρατούμενό που παράγεται από την πρόσθεση έχει μέγεθος 1 bit. Ο αθροιστής αυτός βασίζεται στην ιδέα ότι το κρατούμενο διαδίδεται στην επόμενη βαθμίδα πρόσθεσης, και καλείται συνήθως ως 4-bit αθροιστής με κυματικό κρατούμενο (Ripple-carry adder). Ο 4-bit αθροιστής ripple-carry αποτελείται από τέσσερις (4) πλήρης αθροιστές ενός bit.



Σχήμα 39 - Four-bit ripple carry adder

Ο πλήρης αθροιστής 4-bits περιγράφεται με τη χρήση στιγμιοτύπων του απλού αθροιστή και όχι με τα δομικά στοιχεία που τον απαρτίζουν. Ο αριθμός των στιγμιότυπων είναι τέσσερα, όσα και τα bits του αθροιστή. Η οντότητα (entiy) του κυκλώματος περιγράφεται σύμφωνα με όσα έχουν ήδη οριστεί. Στο σώμα της αρχιτεκτονικής έχουμε την δήλωση ενός σήματος c το οποίο έχει μέγεθος 3 bits και θα χρησιμοποιηθεί για την μεταφορά του κρατουμένου μεταξύ των αθροιστών. Στη συνέχεια, έχουμε την δήλωση της χρήσης ενός υποκυκλώματος με όνομα fa, το οποίο αποτελείται από τρεις εισόδους (a, b, c_i) και δύο εξόδους (s, c_o).

Αυτός ο τρόπος δήλωσης της χρήσης κάποιου υποκυκλώματος προϋποθέτει την αποθήκευση του υποκυκλώματος στον ίδιο φάκελο (directory) με το κυρίως κύκλωμα. Η δήλωση χρήσης εσωτερικών σημάτων καθώς και υποκυκλωμάτων γίνεται στην περιοχή δήλωσης της αρχιτεκτονικής, πριν από τη λέξη begin, που δηλώνει την έναρξη του κυρίως κυκλώματος. Το όνομα του υποκυκλώματος (component) πρέπει να είναι ακριβώς το ίδιο με το όνομα το οποίο αποθηκεύτηκε.

Στο κύριο σώμα της αρχιτεκτονικής περιγράφεται η χρήση τεσσάρων στιγμιοτύπων με ονόματα bit1, bit2, bit3 και bit4. Μετά το όνομα του στιγμιτύπου γράφεται το όνομα του υποκυκλώματος, προσέχοντας να είναι ίδιο με το όνομα στο components (περιοχή δήλωσης). Στη συνέχεια, τα σήματα του υποκυκλώματος αντιστοιχίζονται με τα σήματα που έχουν οριστεί στο κυρίως κύκλωμα.

```
-- Four-bit ripple-carry adder circuit

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Part3 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Toggle switches
          LEDR  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- Red LEDs
          LEDG  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)); -- Green LEDs
END Part3;

ARCHITECTURE Structure OF Part3 IS
    COMPONENT fa
        PORT (a, b, ci  : IN STD_LOGIC; -- ci: carry-in
              s, co     : OUT STD_LOGIC); -- co: carry-out, s: sum a+b
    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));

    -- Display the inputs
    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;
```

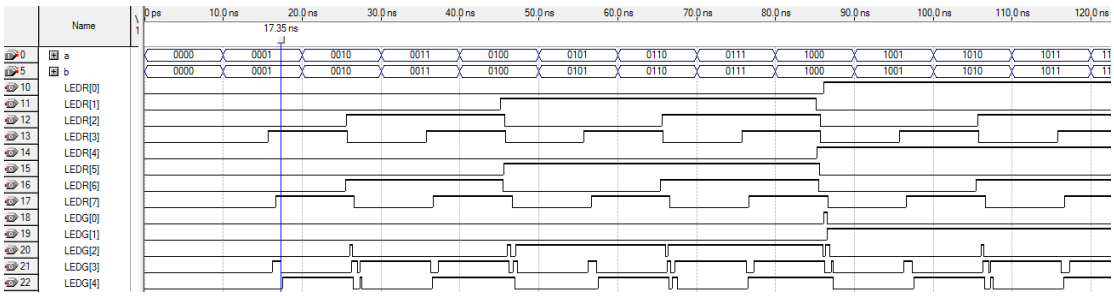
Πρόγραμμα 16 - Four-bit ripple carry adder

Για την υλοποίηση του κυκλώματος θα πρέπει να γίνει η αντιστοίχηση των ακροδεκτών (pin assignment), ώστε οι είσοδοι και οι έξοδοι του κυρίως κυκλώματος (όχι των υποκυκλωμάτων) 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 ανταπτυξιακού.

SW[0]	Input	PIN_N25	LEDR[0]	Output	PIN_AE23
SW[1]	Input	PIN_N26	LEDR[1]	Output	PIN_AF23
SW[2]	Input	PIN_P25	LEDR[2]	Output	PIN_AB21
SW[3]	Input	PIN_AE14	LEDR[3]	Output	PIN_AC22
SW[4]	Input	PIN_AF14	LEDR[4]	Output	PIN_AD22
SW[5]	Input	PIN_AD13	LEDR[5]	Output	PIN_AD23
SW[6]	Input	PIN_AC13	LEDR[6]	Output	PIN_AD21
SW[7]	Input	PIN_C13	LEDR[7]	Output	PIN_AC21
			LEDG[0]	Output	PIN_AE22
			LEDG[1]	Output	PIN_AF22
			LEDG[2]	Output	PIN_W19
			LEDG[3]	Output	PIN_V18
			LEDG[4]	Output	PIN_U18

Πίνακας 15 - Αντιστοίχιση ακροδεκτών

Μετά το τέλος της περιγραφής του κυκλώματος και την αντιστοίχιση των ακροδεκτών, και αφού δεν υπάρχουν λάθη κατά τη μεταγλώττισή του, το κύκλωμα ελέγχεται αν ανταποκρίνεται σωστά στις διάφορες τιμές των εισόδων που δώσαμε.

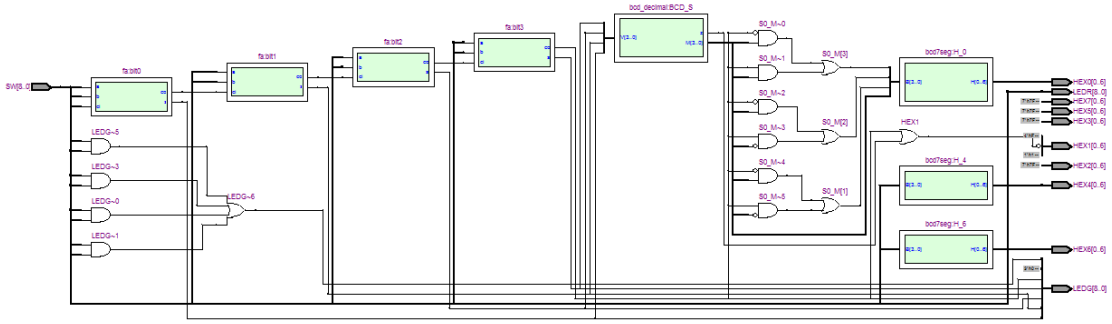


Σχήμα 40 - Προσομοίωση Αθροιστή 4 bits

4.5.2. Αθροιστής ψηφίων BCD

Το κύκλωμα αθροιστή κώδικα BCD προσθέτει δύο δεκαδικά ψηφία σε κώδικα BCD και ένα πιθανό κρατούμενο από προηγούμενη βαθμίδα, το άθροισμα των οποίων στην έξοδο δεν μπορεί να είναι μεγαλύτερο από τον αριθμό 19. Αυτός ο περιορισμός υφίσταται επειδή κάθε ψηφίο εισόδου δεν ξεπερνάει το 9 (1001) και συνεπώς αν τροφοδοτηθεί ένας δυαδικός αθροιστής 4 bits με δύο ψηφία BCD, θα σχηματίσει το άθροισμα δυαδικά και θα παράγει ένα αποτέλεσμα που μπορεί να κυμαίνεται από 0 μέχρι 19 (άθροισμα 9+9+1).

Για τη δημιουργία του αθροιστή χρησιμοποιούνται κάποια υποκυκλώματα, όπως ενός αθροιστή 4 bits (fa), ενός μετατροπέα δυαδικού σε δεκαδικό (bcd_decimal) και ενός μετατροπέα κώδικα από bcd σε ενδείκτη επτά τομέων (bcd7seg).



Σχήμα 41 - RTL Viewer BCD adder

```
-- One-digit BCD adder S1S0 = A + B + Cin
-- Inputs:  SW7-4 = A
--          SW3-0 = B
-- Outputs: A is displayed on HEX6
--          B is displayed on HEX4
--          S1S0 is displayed on HEX1 and HEX0

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Part4 IS
    PORT (SW                    : IN STD_LOGIC_VECTOR(8 DOWNTO 0); -- Toggle switches
          LEDR                  : OUT STD_LOGIC_VECTOR(8 DOWNTO 0); -- Red LEDs
          LEDG                  : OUT STD_LOGIC_VECTOR(8 DOWNTO 0); -- Green LEDs
          HEX7, HEX6, HEX5, HEX4: OUT STD_LOGIC_VECTOR(0 TO 6); -- 7-segs
          HEX3, HEX2, HEX1, HEX0: OUT STD_LOGIC_VECTOR(0 TO 6)); -- 7-segs
END Part4;
```

```
ARCHITECTURE Structure OF Part4 IS
    COMPONENT fa
        PORT (a, b, ci  : IN STD_LOGIC; -- ci: carry-in
              s, co     : OUT STD_LOGIC); -- co: carry-out, s: sum a+b
    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)); -- 7-segs
    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); -- Modified S0 for sums > 15
    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));

    -- Display the inputs
    LEDR <= SW;
    LEDG(4 DOWNTO 0) <= (C(4) & S);

    -- Display the inputs
    H_6: bcd7seg PORT MAP (A, HEX6);
    HEX7 <= ("1111111"); -- Display blank

    H_4: bcd7seg PORT MAP (B, HEX4);
    HEX5 <= "1111111"; -- Display blank

    -- Detect illegal inputs, display on LEDG(8)
    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";

    -- Display the sum bcd_decimal (V, z, M);
    BCD_S: bcd_decimal PORT MAP (S, S1, S0);
    -- S is really a 5-bit # with the carry-out, but bcd_decimal handles only
    -- the lower four bit (sums 00-15). To account for sums 16, 17, 18, 19 S0
    -- has to be modified in the cases that carry-out = 1. Use multiplexers:
    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);
    -- S is really a 5-bit #, but bcd_decimal works for only the lower four bits
    -- (sums 00-15). To account for sums 16, 17, 18, 19 S1 should be a 1 when
    -- the carry-out is a 1
    HEX1 <= ('1' & NOT(S1 OR C(4)) & NOT(S1 OR C(4)) & "1111"); -- Display blank or 1
    HEX2 <= "1111111"; -- Display blank
    HEX3 <= "1111111"; -- Display blank
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)); -- 7-segs
END bcd_decimal;

ARCHITECTURE Structure OF bcd_decimal IS
    SIGNAL B : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    -- Circuit A
    z <= (V(3) AND V(2)) OR (V(3) AND V(1));

    -- Circuit B
    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));

    -- Multiplexers
    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 Structure OF bcd7seg IS
BEGIN
    --
    --    0
    --   ---
    --  |   |
    -- 5|   |1
    --  | 6 |
    --   ---
    --  |   |
    -- 4|   |2
    --  |   |
    --   ---
    --    3
    --
    -- B    H
    -- ----------
    -- 0 0000001;
    -- 1 1001111;
    -- 2 0010010;
    -- 3 0000110;
    -- 4 1001100;
```

```
    -- 5 0100100;
    -- 6 1100000;
    -- 7 0001111;
    -- 8 0000000;
    -- 9 0001100;
    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 17 - One-digit BCD adder

4.6. Ακολουθιακά Κυκλώματα (Sequential Circuits)

Ακολουθιακά κυκλώματα χαρακτηρίζονται τα κυκλώματα στα οποία οι τιμές των εξόδων σε κάθε χρονική στιγμή δεν εξαρτώνται μόνο από τις τιμές που έχουν οι είσοδοί τους εκείνη την χρονική στιγμή, αλλά και από τις τιμές των εξόδων σε προηγουμένες χρονικές στιγμές. Τα ακολουθιακά κυκλώματα χωρίζονται σε δύο κατηγορίες, τα σύγχρονα και τα ασύγχρονα. Σύγχρονο λέγεται ένα ακολουθιακό κύκλωμα το οποίο αλλάζει κατάσταση σε συγκεκριμένη χρονική στιγμή, με τον παλμό ενός ρολογιού, δηλαδή οι τιμές των σημάτων του αλλάζουν σε διακριτές χρονικές στιγμές. Αντίθετα, όταν ένα ακολουθιακό κύκλωμα αλλάζει κατάσταση με βάση την σειρά που αλλάζουν τιμές οι είσοδοι (σε οποιαδήποτε χρονική στιγμή), λέγεται ασύγχρονο ακολουθιακό κύκλωμα.

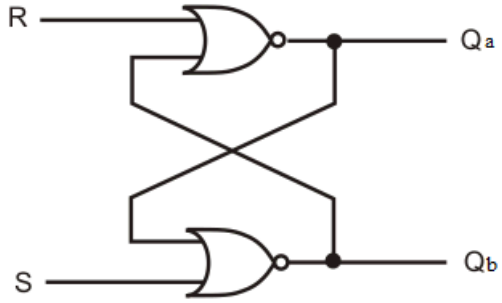
Σε ένα σύχρονο ακολουθιακό κύκλωμα τα στοιχεία μνήμης είναι flip-flops. Το flip-flop χρησιμοποιείται ως κύτταρο μνήμης γιατί είναι ένα κύκλωμα που μπορεί να διατηρηθεί σε μια κατάσταση έως ότου κάποιο κατάλληλο σήμα εισόδου το κάνει να αλλάξει κατάσταση (αποθήκευση 1 bit πληροφορίας).

Σε ένα ασύχρονο ακολουθιακό κύκλωμα τα στοιχεία μνήμης είναι λογικές πύλες που προκαλούν καθυστέρηση διάδοσης στα σήματα που διαδίδονται μέσα απ' αυτές και ονομάζονται μανταλωτές (latches).

Τα κυκλώματα που θα περιγράψουμε σ’ αυτή την ενότητα είναι ένας μανδαλωτής τύπου D (D-latch), ένα flip flop τύπου D και ένας καταχωρητής εύρους 8 bits.

4.6.1. Μανδαλωτής τύπου RS (RS latch)

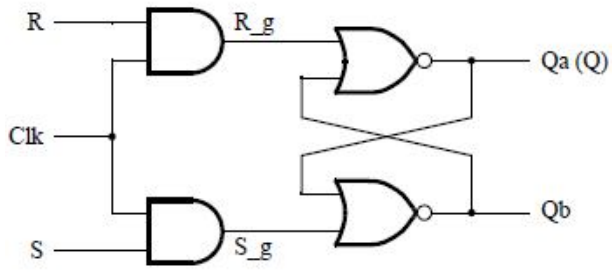
Το κύκλωμα του μανταλωτή (latch) μπορεί να υλοποιηθεί με δυο πύλες NOR, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 42 - Μανταλωτής SR με πύλες NOR

Ο μανταλωτής με πύλες NOR είναι ένα ασύχρονο ακολουθιακό κύκλωμα που έχει δύο πύλες NOR, δύο εισόδους R (Reset) και S (Set), και δύο εξόδους Qa και Qb. Η κατάσταση του μανταλωτή είναι η τιμή της εξόδου Q. Οι χρήσιμες καταστάσεις, τις οποίες μπορεί να βρεθεί ο μανταλωτής είναι η κατάσταση θέσης (set) όπου Qa=1 και Qb=0, και η κατάσταση επαναφοράς (reset) ή μηδενισμού (clear) όπου Qa=0 και Qb=1.

Αν οι είσοδοι του βασικού κυκλώματος μανδαλωτή SR οδηγηθούν από δυο πύλες AND, στις οποίες η μια είσοδος τους συνδέεται σε κοινή πηγή παλμών ρολογιού, προκύπτει ο χρονιζόμενος μανδαλωτής SR (*Σχήμα 43*).



Σχήμα 43 - Χρονιζόμενος μανδαλωτής RS

Ένα τέτοιο κύκλωμα μπορεί να διατηρήσει την εσωτερική του κατάσταση επ' αόριστο. Η λειτουργία του latch τροποποιείται (flip-flop) με την τοποθέτηση πρόσθετης εισόδου ελέγχου (clock) που καθορίζει πότε θα αλλαχθεί η κατάστασή του. Όταν το σήμα ελέγχου Clk ισούται με 0, τότε οι είσοδοι R_g και S_g του μανταλωτή θα είναι μηδέν, ανεξάρτητα από τις τιμές των σημάτων S και R. Επομένως ο μανδαλωτής θα διατηρεί την τρέχουσα κατάσταση του για όσο χρόνο είναι Clk=0. Όταν αλλάξει στην τιμή 1 η είσοδος Clk, τότε

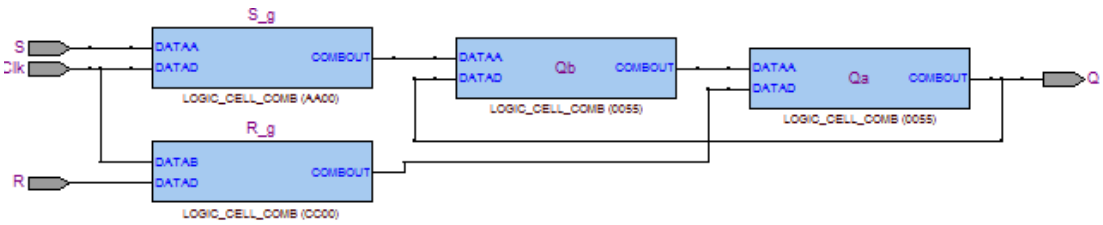
τα σήματα R_g και S_g θα είναι τα ίδια με τα σήματα R και G αντίστοιχα. Επομένως σε αυτόν τον τρόπο λειτουργίας ο μανδαλωτής λειτουργεί όπως περιγράψαμε αρχικά.

Ο κώδικας VHDL που χρησιμοποιεί λογικές εκφράσεις για να περιγράψει αυτό το κύκλωμα είναι ο εξής:

```
-- A gated RS latch
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Part1 IS
    PORT (Clk, R, S : IN STD_LOGIC;
          Q         : OUT STD_LOGIC);
END Part1;

ARCHITECTURE Structural OF Part1 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;
```

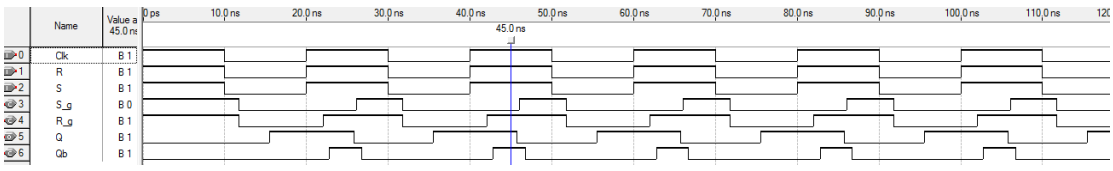
Πρόγραμμα 18 - RS latch

Στο *Σχήμα 44*, φαίνεται η υλοποίηση του RS latch στον Technology Map Viewer.



Σχήμα 44 - RS latch Technology Map viewer

Η προσομοίωση επιβεβαιώνει τη σωστή λειτουργία του RS latch (*Σχήμα 45*)

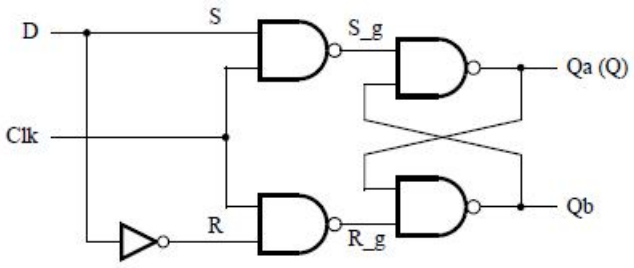


Σχήμα 45 - Προσομοίωση RS latch

4.6.2. Μανδαλωτής τύπου D (D latch)

Το μάνδαλο τύπου D είναι το απλούστερο κύτταρο αποθήκευσης της τιμής ενός bit. Έχει μία είσοδο δεδομένων D, μία είσοδο ενεργοποίησης En και μία έξοδο Q. Η έξοδος ακολουθεί την είσοδο του μανδαλωτή όταν η είσοδος ενεργοποίησης βρίσκεται σε κατάσταση high (Clk=1). Όσο η είσοδος ενεργοποίησης είναι σε κατάσταση low η τιμή της εισόδου δεν επηρεάζει την έξοδο, αλλά αυτή παραμένει στην προηγούμενη κατάσταση.

Στο σχήμα που ακολουθεί φαίνεται το κύκλωμα ενός τέτοιου μανδαλωτή:



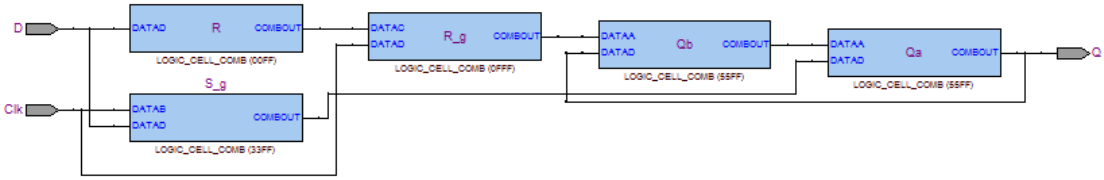
Σχήμα 46 - Μανδαλωτής τύπου D

Ο κώδικας VHDL που χρησιμοποιεί λογικές εκφράσεις για να περιγράψει αυτό το κύκλωμα είναι ο εξής:

```
-- A gated D latch
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 Structural 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 Structural;
```

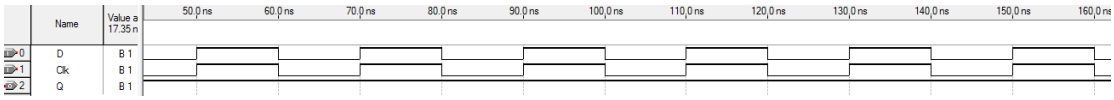
Πρόγραμμα 19 - D latch

Στο *Σχήμα 47*, φαίνεται η υλοποίηση του D latch στον Technology Map Viewer.



Σχήμα 47 - D latch Technology Map viewer

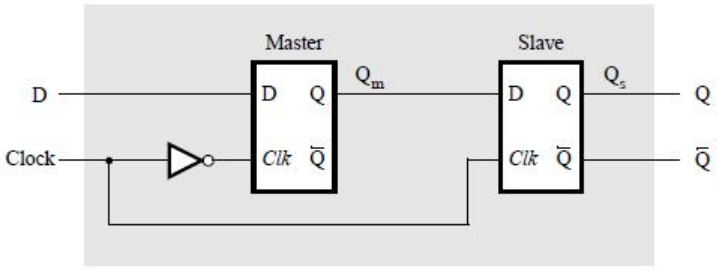
Η προσομοίωση επιβεβαιώνει τη σωστή λειτουργία του D latch (*Σχήμα 48*)



Σχήμα 48 - Προσομοίωση D latch

4.6.3. Flip-Flop τύπου D

Τα flip flop είναι κυκλώματα τα οποία αποτελούνται από μία είσοδο δεδομένων D, μία είσοδο ρολογιού Clk και δύο εξόδους Q και QN. Αποτελούν, όπως και οι μανδαλωτές, τα πιο μικρά στοιχεία αποθήκευσης, χωρητικότητας 1 bit. Υπάρχουν δύο είδη flip-flops. Το πρώτο είδος είναι αυτά που η έξοδος τους επηρεάζεται από την είσοδο κατά το θετικό μέτωπο του παλμού του ρολογιού, δηλαδή την στιγμή της μετάβασης από το 0 (low) στο 1 (high) του παλμού. Το δεύτερο είδος είναι αυτά που η έξοδος τους επηρεάζεται από την είσοδο κατά την αρνητική ακμή του ρολογιού, δηλαδή, την στιγμή της μετάβασης από το 1 (high) στο 0 (low) του παλμού. Τα κυκλώματα αυτά λέγεται ότι εμφανίζουν διέγερση μετώπου (edge-triggering).



Σχήμα 49 - Λογικό κύκλωμα D Flip-Flop master-slave

Ένα τέτοιο κύκλωμα φαίνεται παραπάνω και ονομάζεται D Flip-Flop master-slave.

Αποτελείται από δυο χρονιζόμενους μανδαλωτές D. Ο πρώτος μανδαλωτής που ονομάζεται master αλλάζει κατάσταση όταν Clock = 1. Ο δεύτερος, που ονομάζεται slave αλλάζει κατάσταση όταν Clock = 0. Όταν ο παλμός ρολογιού έχει τιμή 1, ο master παρακολουθεί την τιμή του σήματος εισόδου D αλλά ο slave δεν αλλάζει κατάσταση. Όταν το ωρολογιακό σήμα αλλάξει στην τιμή 0, ο master παύει να παρακολουθεί τις αλλαγές της εισόδου D. Ταυτόχρονα, ο slave αποκρίνεται στην τιμή Qm (της εξόδου του master) και μεταβάλει την έξοδο του ανάλογα. Εφόσον η τιμή Qm δεν αλλάζει καθώς Clock = 0, ο slave μπορεί να υποστεί το πολύ μια αλλαγή κατά την διάρκεια ενός ωρολογιακού παλμού και μάλιστα συμβαίνει κατά το αρνητικό μέτωπο αυτού.

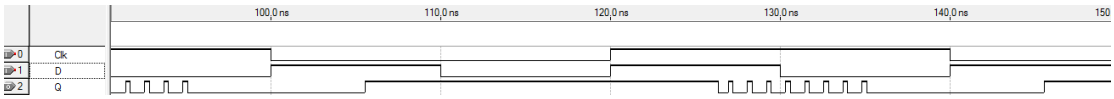
Ο τρόπος περιγραφής του D Flip-Flop γίνεται με τη χρήση πακέτων ή στιγμιοτύπων, όπως και στην περίπτωση του πλήρη αθροιστή.

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

-- SW0 drive the D input of the flip-flop,
-- SW1 is the edge-sensitive Clock input,
-- LEDR0 is Q output
ENTITY Part3 IS
    PORT (SW    : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          LEDR  : OUT STD_LOGIC_VECTOR(0 TO 0));
END Part3;

ARCHITECTURE Structural OF Part3 IS
    COMPONENT D_latch
    PORT (Clk, D: IN STD_LOGIC;
          Q     : OUT STD_LOGIC);
    END COMPONENT;
    SIGNAL Qm, Qs : STD_LOGIC;
BEGIN
    -- D_latch (input Clk, D, output Q)
    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;

-- A gated D latch
ENTITY D_latch IS
    PORT (Clk, D: IN STD_LOGIC;
          Q     : OUT STD_LOGIC);
END D_latch;
```

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

-- A gated D latch
ENTITY D_latch IS
    PORT (Clk, D: IN STD_LOGIC;
          Q     : OUT STD_LOGIC);
END D_latch;

ARCHITECTURE Structural 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 Structural;
```

Πρόγραμμα 20 - Master-slave D flip-flop

Στην προσομοίωση που ακολουθεί (*Σχήμα 50*) φαίνεται η σωστή λειτουργία του D flip-flop. Όταν το ρολόι μεταβαίνει από την κατάσταση low (0) στην κατάσταση high (1), η έξοδος αλλάζει τιμή και παίρνει εκείνη της εισόδου κατά την συγκεκριμένη χρονική στιγμή. Σε όλες τις άλλες περιπτώσεις δεν συμβαίνει καμμία αλλαγή, δηλαδή η έξοδος κρατάει την τιμή που είχε πάρει κατά το τελευταίο θετικό μέτωπο του ωρολογιακού παλμού.



Σχήμα 50 - Προσομοίωση D flip-flop

4.6.4. Καταχωρητής (Register) 16 bits

Οι καταχωρητές θεωρούνται ένα σύνολο από 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, άρα σε έναν καταχωρητή που αποτελείται από n flip flops μπορούμε να αποθηκεύσουμε n bits πληροφορίας. Επομένως, για την δημιουργία ενός καταχωρητή χωρητικότητας 16 bits θα πρέπει να χρησιμοποιήσουμε 16 flip-flops.

Η εισαγωγή των δεδομένων σ' έναν καταχωρητή μπορεί να γίνει είτε σειριακά, είτε παράλληλα. Στην πρώτη περίπτωση σε κάθε ωρολογιακό παλμό έχουμε είσοδο 1 bit, ενώ στην δεύτερη περίπτωση εισάγονται και τα n bits με ένα ωρολογιακό παλμό. Το ίδιο ισχύει και για την εξαγωγή των δεδομένων, η οποία μπορεί να γίνει είτε παράλληλα είτε σειριακά.

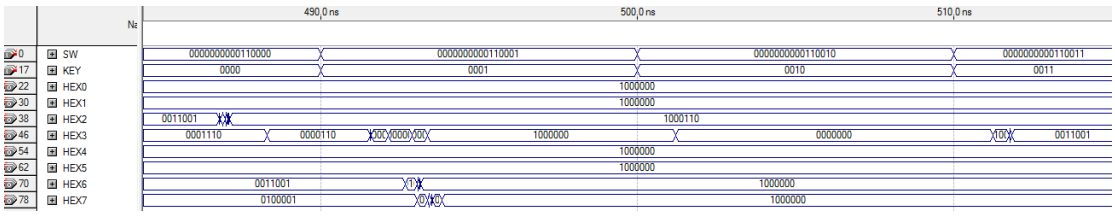
Ο καταχωρητής που περιγράφεται είναι παράλληλης εισόδου και παράλληλης εξόδου. Το όνομα της οντότητας του κυκλώματος είναι regne. Αποτελείται από τρεις εισόδους εκ των οποίων οι δύο (Clock και Resetn) έχουν εύρος 1 bit και η τρίτη, αυτή των δεδομένων (R), έχει εύρος 16 bits. Η έξοδος του κυκλώματος (Q) είναι και αυτή των 16 bits, όπως και η είσοδος. Η είσοδος Resetn λειτουργεί σαν μια ασύγχρονη είσοδος μηδενισμού. Στο μέρος της αρχιτεκτονικής έχουμε μια διαδικασία (process), όπως σε όλα τα σύγχρονα ακολουθιακά κυκλώματα. Στην λίστα ευαισθησίας περιέχονται οι είσοδοι Clock και Resetn. Αν η είσοδος Resetn είναι ίση με το '0' τότε ο καταχωρητής δεν λειτουργεί και σαν έξοδο παίρνουμε την μηδενική τιμή, ανεξαρτήτως της εισόδου. Αν η είσοδος Resetn ισούται με '1' τότε ο καταχωρητής λειτουργεί κανονικά και η τιμή της εξόδου εξαρτάται πλέον από το ρολόι του κυκλώματος, δηλαδή, κατά το θετικό μέτωπο ωρολογιακών παλμών θα είναι στην έξοδο η τιμή που έχει εκείνη την στιγμή η είσοδος (Q<=R;). Η χρήση της εντολής ELSE παραλείπεται, μιας και επιθυμούμε την διατήρηση της τιμής της εξόδου μέχρι το επόμενο θετικό μέτωπο παλμών, οπότε η έξοδος θα ανανεωθεί. Η λογική ανανέωσης της εξόδου σε έναν καταχωρητή είναι ίδια με αυτή του απλού flip flop.

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

-- KEY0 is an active-low asychronous resetn
-- KEY1 is the clock input for reg_A
ENTITY Part5 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 Part5;

ARCHITECTURE Behavior OF Part5 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
    -- regne (R, Clock, Resetn, En, Q)
    A_reg: regne PORT MAP (SW, KEY(1), KEY(0), A);
    B <= SW;
    -- Drive the displays through 7-seg decoders
    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 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
    --
    --     0
    --    ---
    --   |   |
    --  5|   |1
    --   | 6 |
    --    ---
    --   |   |
    --  4|   |2
    --   |   |
    --    ---
    --     3
    --
    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 21 - 16-bits register

Στο *Σχήμα 51*, φαίνεται η προσομοίωση της λειτουργίας του καταχωρητή, η οποία συμφωνεί με τις θεωρητικές απαιτήσεις. Όταν η είσοδος Resetn είναι ίση με ‘0’ στην έξοδο έχουμε πάντα ‘0’, ενώ όταν η είσοδος Resetn παίρνει την τιμή ‘1’ και έχουμε θετικό μέτωπο ωρολογιακών παλμών, η έξοδος παίρνει την τιμή που έχει η είσοδος την συγκεκριμένη χρονική στιγμή.



Σχήμα 51 - Προσομοίωση καταχωρητή 16 bits

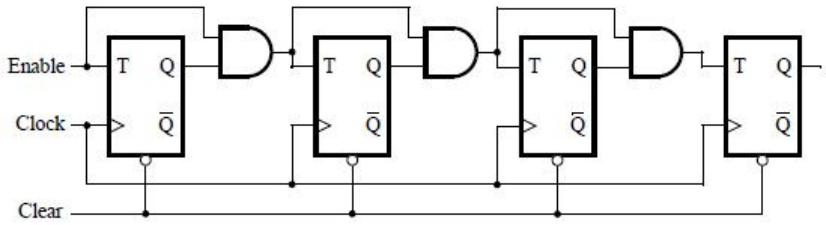
4.7. Απαριθμητές (Counters)

Απαριθμητές ονομάζονται τα κυκλώματα που μπορούν να αυξήσουν ή να μειώσουν την τιμή της εξόδου τους κατά ένα, όταν στην είσοδο ρολογιού δέχονται παλμό clock. Χρησιμοποιούνται για να μετρούν πόσες φορές εμφανίστηκε κάποιο γεγονός ή για να δημιουργούν χρονικές καθυστερήσεις για τον έλεγχο διαφόρων εργασιών σε ένα σύστημα. Η κατασκευή τέτοιων κυκλωμάτων γίνεται συνήθως με την χρήση καταχωρητών, αθροιστών, αφαιρετών ή με flip-flops διαφόρων τύπων.

4.7.1. Απαριθμητής 4 bits (4-bit counter)

Ο απαριθμητής που σχεδιάζεται είναι των τεσσάρων bits και διαθέτει μια είσοδο μηδενισμού και μια είσοδο ενεργοποίησης. Αποτελείται από τρεις εισόδους του 1 bit και μία έξοδο των 4 bits. Αφού η έξοδος έχει εύρος 4 bits σημαίνει ότι μπορεί να πάρει μέχρι και την τιμή ‘1111’, που στο δεκαδικό σύστημα είναι ο αριθμός 15, δηλαδή καταμετρά συνολικά δεκαέξι καταστάσεις.

Οι είσοδοι clear και enable είναι οι είσοδοι μηδενισμού και ενεργοποίησης. Εάν η είσοδος clear βρίσκεται σε λογική κατάσταση μηδέν, η έξοδος του κυκλώματος (count) θα μηδενίζεται. Αν η είσοδος clear είναι ίση με ένα και η είσοδος ενεργοποίησης enable είναι και αυτή ένα, τότε η τιμή της εξόδου του μετρητή θα αυξάνεται κατά ένα σε κάθε θετικό ωρολογιακό μέτωπο, αλλιώς (αν enable=0) η τιμή της εξόδου θα παραμένει αμετάβλητη.



Σχήμα 52 - 4-bit counter

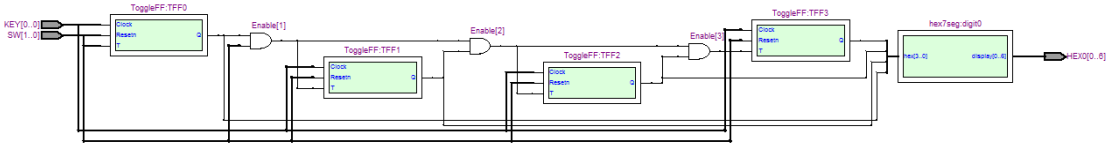
Οι παραπάνω περιπτώσεις των εισόδων περιγράφονται στον κώδικα μέσω μιας διαδικασίας (process). Η λίστα ευαισθησίας περιέχει την είσοδο για το ρολόι και την είσοδο μηδενισμού. Οποιαδήποτε αλλαγή στις τιμές αυτών των δύο εισόδων προκαλεί την αλλαγή της κατάστασης της εξόδου.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--
-- Inputs:  KEY0 is the manual clock
--          SW0 is the active low reset
--          SW1 enable signal for the counter
-- Outputs: HEX0 is the hex segment display
--
ENTITY Part1 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 Part1;

ARCHITECTURE Behavior OF Part1 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
    -- 4-bit Counter based on T-flip flops
    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));

    -- Drive the displays
    digit0: hex7seg PORT MAP (Count(3 DOWNTO 0), HEX0);
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- T Flip-flop
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
    --
    --     0
    --    ---
    --   |   |
    -- 5|   |1
    --   | 6 |
    --    ---
    --   |   |
    -- 4|   |2
    --   |   |
    --    ---
    --     3
    --
    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 22 - 4-bit counter

Στο *Σχήμα 30* φαίνεται το κύκλωμα 4-bit counter σε RTL Viewer.



Σχήμα 53 - 4-bit counter RTL Viewer

Κεφάλαιο 5

Σχεδίαση απλού επεξεργαστή

5.1. Γενική περιγραφή

Στο παρόν κεφάλαιο παρουσιάζεται η σχεδίαση ενός απλού κυκλώματος επεξεργαστή, ο οποίος θα παρέχει τη δυνατότητα εκτέλεσης τεσσάρων εντολών (instructions):

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

Στη λειτουργία της μετακίνησης ο επεξεργαστής θα πρέπει να μετακινεί τα δεδομένα που υπάρχουν σε έναν καταχωρητή Ry σε κάποιον άλλο καταχωρητή Rx. Στη λειτουργία της φόρτωσης, ο επεξεργαστής θα πρέπει να μετακινεί τα δεδομένα που υπάρχουν στον διάδρομο δεδομένων D και να τα βάλει σε κάποιον καταχωρητή. Ο *Πίνακας 16*, παρουσιάζει τις εντολές που θα υποστηρίζει ο επεξεργαστής. Στην αριστερή στήλη (Operation) εμφανίζεται το όνομα της εντολής και ο τελεστής της.

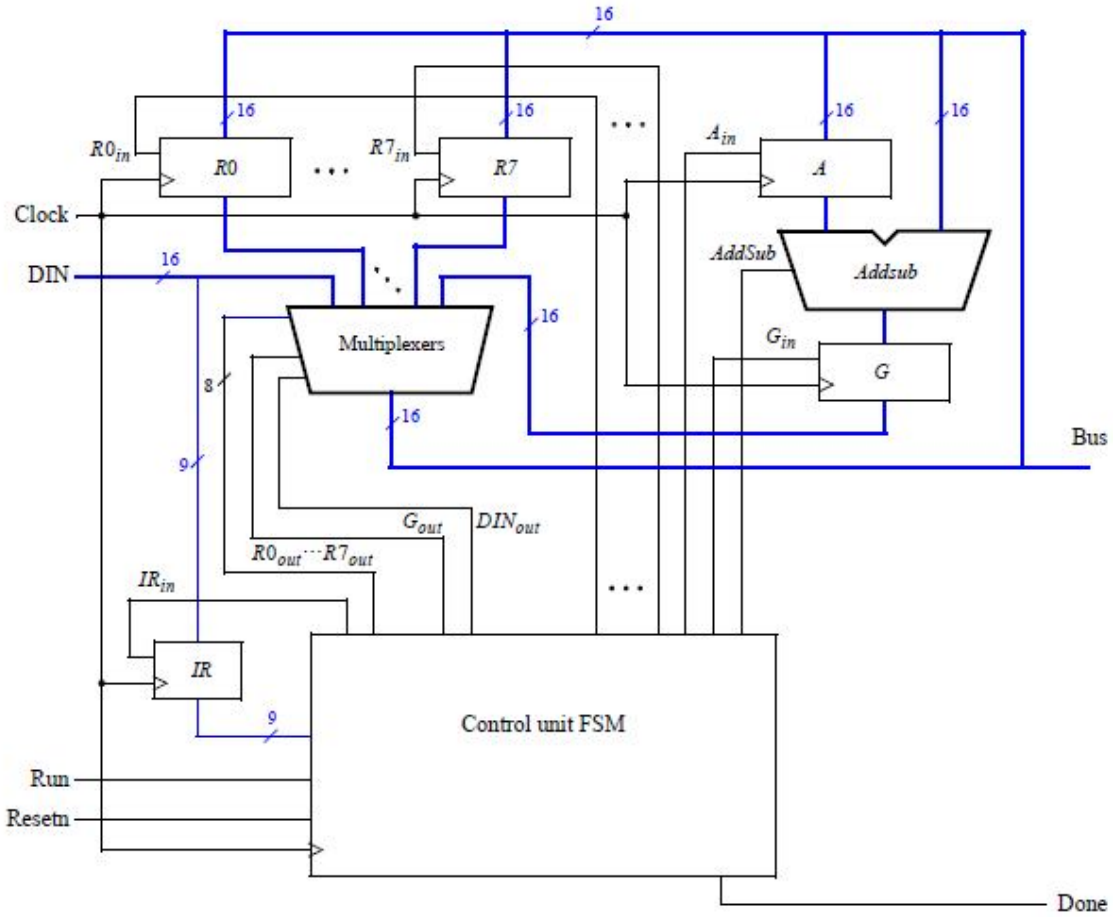
Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Πίνακας 16 - Εντολές εκτέλεσης επεξεργαστή

Η έννοια της σύνταξης Rx ← [Ry] είναι ότι τα περιεχόμενα του καταχωρητή Ry φορτώνονται στον καταχωρητή Rx. Η εντολή **mv** (move) επιτρέπει στα δεδομένα να αντιγραφούν από έναν καταχωρητή σε έναν άλλον. Για την εντολή **mvi** (move immediate) η έκφραση Rx ← D δείχνει ότι το σταθερό D 16-bit φορτώνεται στον καταχωρητή Rx. Οι εντολές add και sub και οι αντίστοιχες δηλώσεις τους Rx ← [Rx] + [Ry] και Rx ← [Rx] + [Ry] επιτρέπουν την εκτέλεση των εντολών της πρόσθεσης και της αφαίρεσης.

Η λειτουργίες της μετακίνησης και της φόρτωσης δεδομένων, λόγω του ότι είναι απλές, θα χρειάζονται μόνον έναν ωρολογιακό παλμό για την εκτέλεσή τους (λειτουργίες ενός βήματος). Οι άλλες δύο λειτουργίες της πρόσθεσης και της αφαίρεσης, απαιτούν τρεις ωρολογιακούς παλμούς για την εκτέλεση τους και αυτό διότι απαιτούν τις δύο προηγούμενες λειτουργίες για την ολοκλήρωσή τους.

Ο επεξεργαστής θα περιλαμβάνει οχτώ καταχωρητές (R0–R7), οι οποίοι θα χρησιμοποιούνται για την αποθήκευση των δεδομένων. Επίσης, θα έχει ένα κανάλι μεταφοράς δεδομένων εύρους 16 bits (δίαυλος επικοινωνίας - bus), ένα κύκλωμα ελέγχου το οποίο θα παράγει τα σήματα ενεργοποίησης για την επιτέλεση των διαφόρων λειτουργιών, καθώς και ένα κύκλωμα αθροιστή/αφαιρέτη (Addsub).



Σχήμα 54 - Digital system

Το *Σχήμα 54*, παρουσιάζει για γενική εικόνα του κυκλώματος. Η περιγραφή του κυκλώματος αρχίζει με τους καταχωρητές που περιέχονται σ' αυτό. Υπάρχει ένας καταχωρητής στη μία από τις δύο εισόδους του αθροιστή/αφαιρέτη, ο οποίος συμβολίζεται με το γράμμα A και θα παίζει τον ρόλο του γενικού καταχωρητή εργασίας ή του

accumulator σε μεγαλύτερα συστήματα. Επίσης, υπάρχει ένας καταχωρητής στην έξοδο της αριθμητικής μονάδας, ο οποίος θα κρατά τα αποτελέσματα και συμβολίζεται με το γράμμα G. Εκτός από αυτούς τους δύο καταχωρητές στο κύκλωμα υπάρχουν άλλοι οχτώ γενικοί καταχωρητές (R0, R1, … , R7). Όλοι οι καταχωρητές έχουν μέγεθος 16 bits, όσο δηλαδή και το μέγεθος των δεδομένων.

Η μεταφορά των δεδομένων από την είσοδο του κυκλώματος (DIN) στους καταχωρητές, αλλά και ανάμεσα στους καταχωρητές, γίνεται μέσω ενός διαύλου επικοινωνίας (bus) των 16 bits. Η δημιουργία του διαύλου γίνεται με τη χρήση πολυπλέκτη.

Το κύκλωμα του αθροιστή/αφαιρέτη αποτελείται από δύο εισόδους δεδομένων, από τις οποίες θα δέχεται τους δύο αριθμούς που θα προστεθούν ή που θα αφαιρεθούν. Η μία είσοδος επικοινωνεί με την δίαυλο μέσω του καταχωρητή A, ενώ η δεύτερη επικοινωνεί απευθείας με την δίαυλο. Η πράξη που θα εκτελέσει το κύκλωμα (πρόσθεση ή αφαίρεση) επιλέγεται από ένα σήμα που ονομάζεται AddSub, το οποίο παράγεται από την μονάδα ελέγχου FSM (Finite State Machine) του επεξεργαστή. Το σήμα AddSub, δηλαδή, λειτουργεί ως ένας διακόπτης επιλογής, που ανάλογα με την τιμή του το κύκλωμα λειτουργεί ως αθροιστής (AddSub=0) ή ως αφαιρέτης (AddSub=1). Το αποτέλεσμα της πράξης θα αποθηκεύεται στον καταχωρητή G.

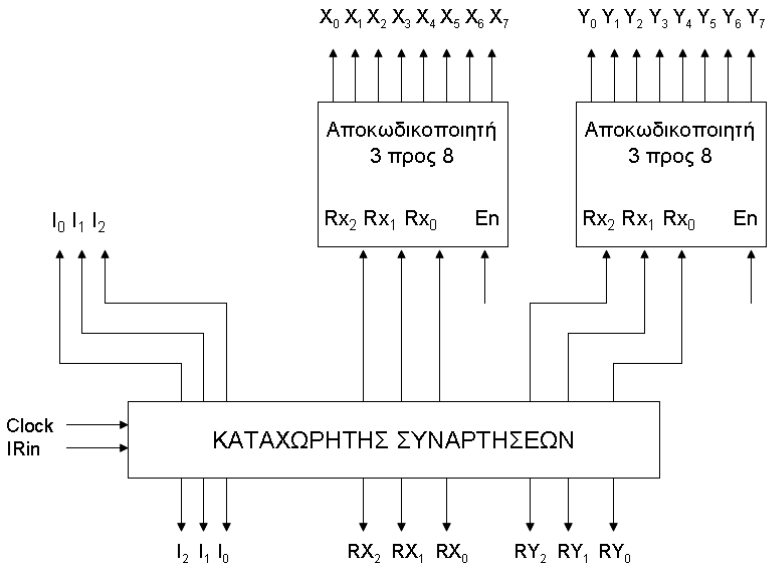
5.2. Εντολές επεξεργαστή

Στο κύκλωμα ελέγχου του επεξεργαστή περιέχονται κυκλώματα τα οποία είναι υπεύθυνα για την σωστή λειτουργία του. Σε ένα από αυτά δηλώνουμε την εντολή (Instruction) που θέλουμε να εκτελέσει ο επεξεργαστής. Η εντολή δίνεται στην είσοδο IRin η οποία έχει μέγεθος 9 bits. Τα bits της εντολής δίνονται με τη βοήθεια εξωτερικών διακοπτών (ush-buttons). Η είσοδος IRin αποθηκεύεται σε έναν καταχωρητή εντολής IR (Instruction Register). Τα τρία πρώτα bits του κωδικού της εντολής χρησιμοποιούνται για τον προσδιορισμό της πράξης. Έτσι, έχουμε ‘000’ για την μετακίνηση, ‘001’ για την φόρτωση, ‘010’ για την πρόσθεση και ‘011’ για την αφαίρεση.

Άρα η μορφή της εντολής είναι:

- “000xxxxxx” για την μετακίνηση
- “001xxxxxx” για την φόρτωση
- “010xxxxxx” για την πρόσθεση
- “011xxxxxx” για την αφαίρεση

Τα τρία πρώτα bits περνούν στα σήματα I_2, I_1, I_0, τα οποία θα χρησιμοποιηθούν μέσα στον κώδικα VHDL σε δομή CASE, για να προσδιορίσουν τον τύπο της εντολής που θα εκτελέσει το κύκλωμα.



Σχήμα 55 - Καταχωρητής εντολών

Τα τρία επόμενα bits (Rx_2, Rx_1, Rx_0) χρησιμοποιούνται για την επιλογή του καταχωρητή όπου θα μεταφερθούν τα δεδομένα και τα τρία τελευταία bits (Ry_2, Ry_1, Ry_0) χρησιμοποιούνται για την επιλογή του δεύτερου καταχωρητή (operands). Στην έξοδο του καταχωρητή συνάρτησης έχουμε δύο αποκωδικοποιητές τρία προς οχτώ (3:8), έναν για κάθε τρίαδα 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.

Τα σήματα ενεργοποίησης των οχτώ καταχωρητών είναι τα Rin (0,1,2,3,4,5,6,7). Οι έξοδοι των καταχωρητών μοιράζονται τη δίαυλο επικοινωνίας με τη βοήθεια ενός πολυπλέκτη, ο οποίος υλοποιείται στη VHDL με μια δομή IF. 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 δύο αποκωδικοποιητές είναι διαρκώς ενεργοποιημένοι. Αυτό επιτυγχάνεται τοποθετώντας την είσοδο enable των αποκωδικοποιητών μόνιμα σε κατάσταση 1. Ο καταχωρητής συνάρτησης αποθηκεύει τις τιμές που έχουν οι είσοδοι όταν η είσοδος IRin είναι ίση με '1'. Τα δεδομένα προς επεξεργασία δίνονται στο σύστημα μέσω εξωτερικών διακοπτών τύπου dip. Μέσω εξωτερικών διακοπτών δίνονται και τα σήματα Clock, W και Resetn.

Εκτός από τον καταχωρητή συναρτήσεων το κύκλωμα ελέγχου συνδέεται και με έναν απαριθμητή. Ο απαριθμητής χρησιμεύει για την δημιουργία των απαιτούμενων σημάτων ελέγχου σε κάθε βήμα. Επειδή, οι λειτουργίες πρόσθεση και αφαίρεση του επεξεργαστή απαιτούν μέχρι και τρία βήματα ή τρεις κύκλους εκτέλεσης, ο απαριθμητής θα είναι των δύο bits και οι τιμές που μπορεί να πάρει η έξοδος T θα είναι: '00', '01', '10', '11'.

5.3. Εντολές επεξεργαστή

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 αντιστοιχεί σε ένα βήμα. Όταν δεν εκτελείται καμία λειτουργία είναι ενεργοποιημένη η έξοδος T_0 (00). Όταν εκτελείται κάποια από τις εντολές μετακίνησης ή φόρτωσης (I='000' ή I='001'), ο απαριθμητής έχει την τιμή '01' και ενεργοποιείται η έξοδος T_1, επειδή οι εντολές αυτές είναι ενός βήματος. Τότε, παράγονται τα κατάλληλα σήματα DINout, για φόρτωση της 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 T_1='01' τα δεδομένα του Rx εξάγονται στη δίαυλο (RXout=1) και μετακινούνται στον καταχωρητή A (Ain=1). Τότε, το σήμα επιλογής Sel του πολυπλέκτη, που τροφοδοτεί τη δίαυλο, παίρνει την κατάλληλη τιμή ώστε ο Rx να εξάγει τα δεδομένα του στο bus.

Το επόμενο βήμα του απαριθμητή T_2=10, αφορά μόνον στις πράξεις της πρόσθεσης και της αφαίρεσης. Στο βήμα αυτό ο Ry εξάγει τα δεδομένα του στη δίαυλο, ώστε να μπορούν να αθροιστούν με τα δεδομένα του A (RYout=1). Στην ίδια περίοδο του απαριθμητή ο G φορτώνεται με το αποτέλεσμα της πρόσθεσης (ή της αφαίρεσης, ανάλογα με την τιμή του σήματος AddSub), οπότε Gin=1.

Στο τελευταίο βήμα του απαριθμητή (T_3=11), ο G εξάγει το αποτέλεσμα της πράξης (add ή sub) στη δίαυλο (Gout=1).

Το κύκλωμα ελέγχου θα δημιουργεί και κάποια σήματα τα οποία θα διασφαλίζουν τη σωστή λειτουργία του επεξεργαστή. Ένα τέτοιο σήμα είναι και το Done, το οποίο θα παίρνει την τιμή '1' κάθε φορά που ολοκληρώνεται μια εντολή του επεξεργαστή. Ένα άλλο σήμα, το οποίο θα το δίνει εξωτερικά ο χρήστης, είναι το W. Κάθε φορά που θα ενεργοποιείται θα επιτρέπει στην είσοδο IR να στέλνει τα δεδομένα της στους αποκωδικοποιητές του κυκλώματος ελέγχου.

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

Στον *Πίνακα 17*, συνοψίζονται τα σήματα ελέγχου που παράγονται για κάθε εντολή του επεξεργαστή, σε κάθε βήμα εκτέλεσης.

<table>
<tr><th></th><th>T_0 = '00'</th><th>T_1 = '01'</th><th>T_2 = '10'</th><th>T_3 = '11'</th></tr>
<tr><td>Μετακίνηση
(mv)</td><td>-</td><td>Rout←[Y]
Rin=[X]
Done=1</td><td>-</td><td>-</td></tr>
<tr><td>Φόρτωση
(mv)</td><td>-</td><td>DINout=1
Rin=[X]
Done=1</td><td>-</td><td>-</td></tr>
<tr><td>Πρόσθεση
(add)</td><td>-</td><td rowspan="2">Rout←[X]
Ain=1</td><td>Rout←[Y]
Gin=1
AddSub=0</td><td rowspan="2">Gout=1
Rin←[X]
Done=1</td></tr>
<tr><td>Αφαίρεση
(sub)</td><td>-</td><td>Rout←[Y]
Gin=1
AddSub=1</td></tr>
</table>

Πίνακας 17 - Σήματα ελέγχου επεξεργαστή σε κάθε εντολή/βήμα

5.4. Υποκυκλώματα επεξεργαστή

Η περιγραφή του κώδικα που υλοποιεί το συγκεκριμένο κύκλωμα του επεξεργαστή, ξεκινά με την περιγραφή των μικρότερων κυκλωμάτων που το απαρτίζουν. Τα κυκλώματα αυτά είναι ο αύξων απαριθμητής (upcount), ο αποκωδικοποιητής (dec3to8) και ο καταχωρητής (regn).

5.4.1. Απαριθμητής

Ο απαριθμητής αποτελείται από δύο εισόδους, εκ των οποίων η μία είναι η είσοδος μηδενισμού του απαριθμητή και η άλλη το clock. Επίσης, έχει μία έξοδο των δύο bits. Κάθε φορά που η είσοδος Clear γίνεται ίση με ‘1’ ο απαριθμητής μηδενίζεται και στην έξοδο Q (Count) παίρνουμε την τιμή ‘00’. Αντίθετα, όταν η είσοδος Clear έχει την τιμή ‘0’ για κάθε θετικό ωρολογιακό παλμό η έξοδος Q (Count) του απαριθμητή αυξάνεται κατά ένα.

```
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 = '1' THEN
                Count <= "00";
            ELSE
                Count <= Count + 1;
            END IF;
        END IF;
    END PROCESS;
    Q <= Count;
END Behavior;
```

Πρόγραμμα 23 - Απαριθμητής upcount

5.4.2. Αποκωδικοποιητής 3:8

Ο αποκωδικοποιητής χρησιμοποιείται για να επιλέγει μία έξοδο με βάση την πληροφορία των γραμμών επιλογής. Το κύκλωμα του αποκωδικοποιητή διαθέτει μία είσοδο ενεργοποίησης (En) η οποία ενεργοποιεί (En=1) ή απενεργοποιεί (En=0) τις εξόδους του αποκωδικοποιητή.

Στον κώδικα (*Πρόγραμμα 24*) η είσοδος En μαζί με την είσοδο W τοποθετούνται σε μια διαδικασία (PROCESS) η οποία χρησιμοποιείται μέσα στο κύριο σώμα της αρχιτεκτονικής. Ανάλογα με τις τιμές των σημάτων (En, W) που περιλαμβάνονται στη διαδικασία, ενεργοποιείται και η αντίστοιχη έξοδος του αποκωδικοποιητή.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dec3to8 IS
    PORT (W  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          En : IN STD_LOGIC;
          Y  : OUT STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;

ARCHITECTURE Behavior OF dec3to8 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 24 - Αποκωδικοποιητής dec3to8

5.4.3. Καταχωρητής

Το κύκλωμα του καταχωρητή που χρησιμοποιείται έχει n bits. Το μέγεθος του καταχωρητή είναι μεταβλητό και προσδιορίζεται από την τιμή που δίνουμε στην παράμετρο n. Στην προκειμένη περίπτωση το μέγεθος του καταχωρητή είναι 16 bits. Ο κώδικας του καταχωρητή (*Πρόγραμμα 25*) καθορίζει ότι αν η είσοδος ενεργοποίησης Rin ισούται με '1', τότε τα flip flop που απαρτίζουν τον καταχωρητή θα φορτώσουν την τιμή της εισόδου R, αλλιώς (αν Rin='0') θα διατηρήσουν την προηγούμενη τιμή που τους δόθηκε.

```
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 25 - Καταχωρητής regn

5.5. Κυρίως πρόγραμμα

Ο κώδικας που περιγράφει το πρόγραμμα του επεξεργαστή χωρίζεται σε τρία τμήματα. Στο πρώτο τμήμα δηλώνονται οι βιβλιοθήκες (library), στο δεύτερο τμήμα δηλώνεται η οντότητα (entity) όπου περιγράφονται τα σήματα εισόδου και εξόδου στο σύστημα και στο τρίτο τμήμα δηλώνεται η αρχιτεκτονική (architecture) όπου περιγράφεται η λογική του επεξεργαστή.

Η δήλωση των βιβλιοθηκών είναι η εξής:

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

Στο σώμα της οντότητας υλοποιείται το ψηφιακό σύστημα με όνομα Proc, στο οποίο δηλώνονται οι είσοδοι και οι έξοδοι του επεξεργαστή.

```
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;
```

Η αρχιτεκτονική του ψηφιακού συστήματος ονομάζεται Behavior και αναφέρεται στην οντότητα Proc. Στην περιοχή δήλωσης του σώματος της αρχιτεκτονικής, γίνονται οι δηλώσεις των συστατικών (COMPONENT), δηλαδή ο ορισμός των υποκυκλωμάτων του αύξων απαριθμητή, του αποκωδικοποιητή και του καταχωρητή, καθώς και οι δηλώσεις των σημάτων που περιέχονται στον επεξεργαστή. Σημειώνεται πως αυτός ο τρόπος χρήσης των υποπρογραμμάτων προϋποθέτει την ύπαρξη όλων των αρχείων στον ίδιο φάκελο (κυρίως πρόγραμμα και υποκυκλώματα).

```
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 T : 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 κύριο σώμα της αρχιτεκτονικής (architecture body) καθορίζεται η λειτουργία του κυκλώματος. Η ύπαρξη ενός στιγμιότυπου (Tstep), αναφέρεται στο υποκύκλωμα του αύξων απαριθμητή. Ο απαριθμητής δέχεται σαν είσοδο το σήμα Clear το οποίο ορίζεται στην προηγούμενη σειρά ως εξής:

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

Όταν το Clear είναι ίσο με ‘1’ δηλαδή όταν κάποια από τις τρεις εισόδους της πύλης or που το δημιουργεί είναι ‘1’, τότε ο απαριθμητής μηδενίζεται.

Το σήμα I είναι κι αυτό εσωτερικό σήμα, το οποίο χρησιμοποιείται για την επιλογή της πράξης. Σ’ αυτό το σημείο του δίνουμε τιμή τα τρία πρώτα bits της συνάρτησης IR:

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

Τα επόμενα δύο στιγμιότυπα με ονόματα decX και decY χρησιμοποιούνται για να αποκωδικοποιήσουν το σήμα count και έχουν σαν έξοδο τις τιμές T_0, T_1, T_2, T_3 και I_0, I_1, I_2 I_3 αντίστοιχα. Στην είσοδο ενεργοποίησης του κάθε αποκωδικοποιητή υπάρχει ένα σήμα (High) στο οποίο έχουμε δώσει την τιμή ‘1’ στην αρχή του κώδικα, έτσι ώστε οι αποκωδικοποιητές να είναι μόνιμα ενεργοποιημένοι.

```
High <= '1';
Clear <= NOT(Resetn) OR Done OR (NOT(Run) AND NOT(T(1)) AND NOT(T(0)));

Tstep: upcount PORT MAP (Clear, Clock, T);
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);
```

Στη συνέχεια δηλώνεται η έναρξη μιας διαδικασίας με όνομα controlsignals, στη λίστα ευαισθησίας της οποίας περιέχονται τα σήματα T, I, Xreg και Yreg. Με αυτήν την διαδικασία ορίζεται η λειτουργία του κυκλώματος ελέγχου με βάση (α) το στάδιο (βήμα T) που βρισκόμαστε και (β) την πράξη που εκτελούμε.

Η περιγραφή του κυκλώματος με αυτόν τον τρόπο οδηγεί στην χρήση δύο εμφωλευμένων εντολών CASE. Στο πρώτο επίπεδο εξετάζονται οι διάφορες τιμές που μπορεί να πάρει το σήμα T και στο δεύτερο επίπεδο εξετάζονται οι τιμές που μπορεί να πάρει το σήμα I για κάθε δυνατή τιμή του T. Έτσι, ξεκινάμε εξετάζοντας πρώτα την περίπτωση T='00'. Σε αυτό το στάδιο (T_0) δεν ενεργοποιείται κανένα σήμα, αφού δεν ενεργοποιείται καμία εντολή.

Στο στάδιο T_1 και στην περίπτωση T='01' έχουμε ενεργοποίηση και των τεσσάρων πράξεων, έτσι ελέγχονται τα σήματα που θα δημιουργηθούν για την κάθε πράξη ξεχωριστά χρησιμοποιώντας το δεύτερο επίπεδο CASE. Για I='000' (μετακίνηση) ενεργοποιούνται τα σήματα Rout, Rin και Done. Τα σήματα που ενεργοποιούνται για I='001' (φόρτωση) είναι τα. DINout, Rin και Done. Η ενεργοποίηση της εξόδου Done δεν συμβαίνει στις περιπτώσεις των πράξεων της πρόσθεσης και της αφαίρεσης, γιατί η μεταφορά και η φόρτωση εκτελούνται σε ένα ωρολογιακό βήμα ενώ η πρόσθεση και η αφαίρεση σε τρία.

Στο επόμενο στάδιο (T_2) όπου T='10', οι μόνες πράξεις που μπορούν να ενεργοποιήσουν κάποιο σήμα είναι η πρόσθεση και η αφαίρεση, για αυτό και οι μόνες πιθανές τιμές που μπορεί να πάρει το σήμα I είναι οι '010' και '011'.

Στο τελευταίο στάδιο (T_3) οι μόνες τιμές που εξετάζουμε για το I είναι αυτές τις πρόσθεσης και της αφαίρεσης, βάζοντας αυτή τη φορά όμως το Done στη τιμή '1', που σημαίνει και το τέλος της πράξης. Η διαδικασία τελειώνει (END PROCESS) αφού ολοκληρώσουμε με όλες τις πιθανές τιμές που μπορεί 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, YYY = Ry
--  For mvi, a second word of data is loaded from DIN
--
```

```
controlsignals: PROCESS (T, I, Xreg, Yreg)
BEGIN
    Done <= '0'; Ain <= '0'; Gin <= '0'; Gout <= '0'; AddSub <= '0';
    IRin <= '0'; DINout <= '0'; Rin <= "00000000"; Rout <= "00000000";
    CASE T IS
        WHEN "00" => -- Store DIN in IR as long as T = 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 εννέα 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 πράξη που θα εκτελέσει ο αθροιστής/αφαιρέτης καθορίζεται από την τιμή που έχει το σήμα AddSub. Με βάση την τιμή αυτή, στο στιγμιότυπο reg_IR καθορίζεται η πράξη που θα εκτελεστεί μεταξύ του περιεχομένου του καταχωρητή A και του αριθμού που υπάρχει στην δίαυλο. Το αποτέλεσμα της πράξης αποθηκεύεται στον καταχωρητή G (reg_G).

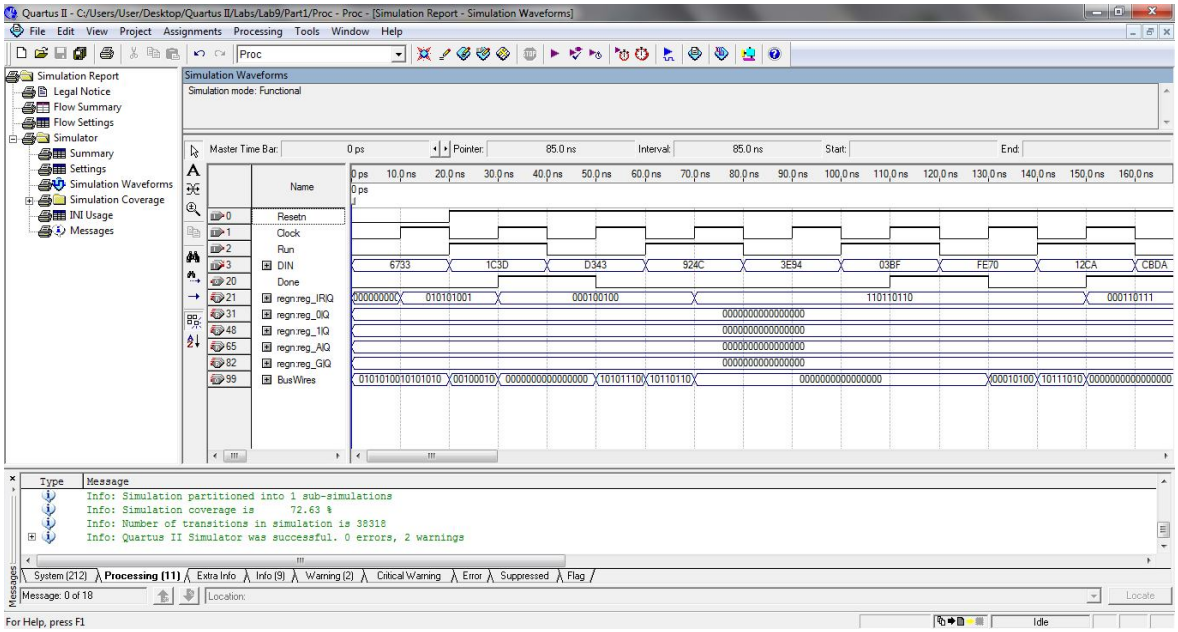
```
    reg_IR: regn GENERIC MAP (n => 9) PORT MAP (DIN(15 DOWNTO 7), IRin, Clock, IR);

    -- 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 τέλος του κώδικα περιγράφεται η δίαυλος επικοινωνίας (bus), αντιστοιχίζοντας τα σήματα επιλογής του πολυπλέκτη που την υλοποιεί. Ανάλογα με την τιμή που έχει το σήμα Sel παίρνουμε στην δίαυλο τα περιεχόμενα του αντίστοιχου καταχωρητή.

```
    -- 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;
```

5.6. Λειτουργική προσομοίωση

Η λειτουργική προσομοίωση του κυκλώματος λαμβάνει χώρα με το πέρας της περιγραφής του κώδικα και την επιτυχή ολοκλήρωση της διαδικασίας της μετάφρασης. Κάθε ωρολογιακός παλμός του διαγράμματος κατά τον οποίο η είσοδος Run έχει τεθεί σε '1' σημαίνει την έναρξη κάποιας λειτουργίας. Η είσοδος Resetn που λειτουργεί ως είσοδος ενεργοποίησης του επεξεργαστή, στην αρχή είναι ίση με '0' και στην συνέχεια την θέτουμε ίση με '1' για να λειτουργήσει το κύκλωμα του επεξεργαστή. Η προσομοίωση επιβεβαιώνει τη σωστή λειτουργία του κυκλώματος (*Σχήμα 56*)



Σχήμα 56 – Λειτουργική προσομοίωση επεξεργαστή

5.7. Αντιστοίχιση ακροδεκτών και υλοποίηση

Αφού διαπιστώθηκε η σωστή λειτουργία του επεξεργαστή μέσα από την προσομοίωση, το μόνο που απομένει είναι η αντιστοίχιση των εισόδων και των εξόδων του κυκλώματος με συγκεκριμένους ακροδέκτες της διάταξης που πρόκειται να προγραμματίσουμε. Η διάταξη έχει ήδη ορισθεί σε ένα FPGA που ανήκει στην οικογένεια Cyclone II της εταιρίας Altera και συγκεκριμένα η διάταξη EP2C35F672C6.

Με την αντιστοίχιση ορίζουμε εννέα διακόπτες (push-buttons) του αναπτυξιακού κυκλώματος DE2 να δίνουν τιμές στη συνάρτηση IR, δηλαδή στο opcode της εντολής. Η εντολή που εκτελείται θα δίνεται με τη βοήθεια αυτών των διακοπτών. Δεκαέξι toggle switches χρησιμοποιούνται ως πηγή δεδομένων εισόδου (DIN), που στη συνέχεια, με τη βοήθεια των εντολών καταχωρούνται στους καταχωρητές του κυκλώματος.

Επιπλέον, ένας διακόπτης 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. Στο τέλος κάθε πρόσθεσης ή αφαίρεσης (όταν Done=1), στους ακροδέκτες της διαύλου θα εμφανίζεται το αποτέλεσμα της πράξης. Ένα τελευταίο Led απεικονίζει το σήμα Done, το οποίο θα ανάβει με το πέρας κάθε εντολής, όταν ολοκληρωθούν τα προβλεπόμενα βήματα του μετρητή.

5.8. Συμπεράσματα

Η ανάπτυξη της αρχιτεκτονικής των διατάξεων προγραμματιζόμενης λογικής (CPLDs και FPGAs), ανέδειξε ότι πρόκειται για κυκλώματα των οποίων το υλικό τους είναι σε θέση να διαμορφωθεί κατάλληλα, ώστε να υλοποιείται η ψηφιακή λογική που σχεδιάζει ο χρήστης. Αυτό οδηγεί στο συμπέρασμα πως αποτελούν κυκλώματα κατάλληλα για προτυποποίηση και ανάπτυξη εφαρμογών.

Η ανάπτυξη μιας σειράς απλών ή λίγο πιο σύνθετων εργαστηριακών εφαρμογών, έχουν ως απώτερο σκοπό να καθοδηγήσουν έναν αρχάριο χρήστη στην ιδέα της ψηφιακής σχεδίασης μέσω διαμορφούμενων κυκλωμάτων FPGAs. Παρουσιάστηκε μια εκτενής αναφορά στην εισαγωγή χρήσης του λογισμικού σχεδίασης, σύνθεσης και προσομοίωσης ψηφιακών κυκλωμάτων Quartus II, πραγματοποιώντας προσομοιώσεις για να επιδείξουμε και στην πράξη τη χρήση του. Χρησιμοποιήθηκε εκτενώς η γλώσσα περιγραφής υλικού VHDL για τη σχεδίαση των εφαρμογών, καταδεικνύνοντας την πλέον ευρεία αποδοχή της στην υλοποίηση ψηφιακής σχεδίασης.

Η καταλληλότητα των εργαλείων που χρησιμοποιήθηκαν και αναλύθηκαν, προκύπτει από το γεγονός της προσομοίωσης και συνεπώς της πρακτικής υλοποίησης, φωτίζοντας πολλές πτυχές της λειτουργίας των κυκλωμάτων που διδάσκονται στη θεωρία. Επίσης, καταδυκνύει τον τρόπο με τον οποίο η θεωρία γίνεται πράξη σε περιβάλλοντα βιομηχανικής σχεδίασης και παραγωγής υλικού (hardware).

Αναμφισβήτητα, η γλώσσα VHDL έχει τη δυνατότητα να οδηγήσει με σχετική ευκολία στη σχεδίαση και υλοποίηση περίπλοκων κυκλωμάτων. Η δυνατότητα σχεδίασης με βάση τη συμπεριφορά και όχι με βάση τη δομή, παρέχει το πλεονέκτημα στον χρήστη να μπορεί να παραμερίσει τα περίπλοκα θέματα εσωτερικής δομής των κυκλωμάτων και να επικεντρωθεί στη συμπεριφορά που ο ίδιος προδιαγράφει. Εξάλλου, ο ιεραρχικός τρόπος σχεδίασης που παρέχει η VHDL είναι κατάλληλος για επαναχρησιμοποίηση υποκυκλωμάτων και ανταλλαγή «υλικού» με τη μορφή κώδικα VHDL ανάμεσα σε χρήστες.

Βιβλιογραφία

Altera, 2006, *"DE2 Development and Education Board User Manual"*, version 1.4

Altera, 2008, *"Introduction to the Altera Quartus II Software"*, version 7.2

Ashenden J. Peter & Lewis Jim, *The designer's Guide to VHDL*, Morgan Kaufmann

Ashenden J. Peter, 2008, *"Digital Design, An Embedded Systems Approach Using VHDL"*, Morgan Kaufmann publishers

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

Carpinelli J. D., 2001, *Computer Systems Organization & Architecture*, Addison-Wesley

Chu P. Pong, *RTL Hardware Design Using VHDL* (e-book).

Gaonkar Ramesh, 2002, *Microprocessor Architecture, Programming, and Applications with the 8085*, Fifth Edition, Prentice Hall

Hamblen O. James & Furman D. Michael, 2008, *Rapid Prototyping Of Digital Systems*, Kluwer Academic Publishers

Mano M. M. & Kime C. R., 2001, *Logic & Computer Design Fundamentals*, Prentice Hall

Navabi Zainalabedin, 1993, *VHDL Analysis and Modeling of Digitas Systems*, International Editions

Navadi Zanalabedin, 2005, *Digital Design and Implementation with Field Programmable Devices*, Kluwer Academic Publishers

Pedroni A. Volnei, 2010, *Circuit Design and Simulation with VHDL*, Second Edition, The MIT Press

Wakerly John, 2002, *Ψηφιακή Σχεδίαση, Αρχές και Πρακτικές*, Εκδόσεις Κλειδάριθμος

Wolf W., 2005, *Computers as Components: Principles of Embedded Computer System Design*, Morgan Kaufman (Elsevier)

Ζυγούρης Ευάγγελος, 2008, *Σχεδίαση Ψηφιακών Κυκλωμάτων και Συστημάτων με χρήση της VHDL*, Τόμος Α & Β, Παν/κές Σημειώσεις, Εργαστήριο Ηλεκτρονικής, Τομέας Ηλεκτρονικής και Υπολογιστών, Τμήμα Φυσικής, Πανεπιστήμιο Πατρών

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

Καπαρός Εμμανουήλ, 2012, *Μελέτη και σχεδίαση μιας υποτυπώδους κεντρικής μονάδας επεξεργασίας στα 32 bit*, Πτυχιακή εργασία, Τμήμα Ηλεκτρονικής, ΤΕΙ Κρήτης

Μουσουλιώτης Παναγιώτης, 2011, *Υλοποίηση μικροεπεξεργαστή σε περιβάλλον FPGA*, Πτυχιακή εργασία, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Ηλεκτρονικής και Υπολογιστών, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Μπεσύρης Δημήτριος, 2002, *Υλοποίηση ενός 8085 εξομοιωτή σε FPGAs με τη χρήση VHDL*, Διπλωματική εργασία (version-III), Εργαστήριο Ηλεκτρονικής, Τομέας Ηλεκτρονικής και Υπολογιστών, Τμήμα Φυσικής, Πανεπιστήμιο Πατρών

Παπαζαχαρίας Αθανάσιος, 2000, *Υλοποίηση ενός 8085 εξομοιωτή σε FPGAs με τη χρήση VHDL*, Διπλωματική εργασία (version-I), Εργαστήριο Ηλεκτρονικής, Τομέας Ηλεκτρονικής και Υπολογιστών, Τμήμα Φυσικής, Πανεπιστήμιο Πατρών

Πασχάλη Μαρία Ελένη, 2008, *Μελέτη ενσωματωμένου επεξεργαστή Nios II και εφαρμογές*, Πτυχιακή εργασία, ΤΕΙ Σερρών, Σχολή Τεχνολογικών Εφαρμογών, Τμήμα Πληροφορικής & Επικοινωνιών, Τομέας Αρχιτεκτονικής Υπολογιστών και Βιομηχανικών Εφαρμογών

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

Προύντζου Χρυσή, 2008, *Μελέτη της αρχιτεκτονικής των ολοκληρωμένων κυκλωμάτων FPGAs και CPLDs και εφαρμογές*, Πτυχιακή εργασία, ΑΤΕΙ Σερρών, Σχολή Τεχνολογικών Εφαρμογών, Τμήμα Πληροφορικής & Επικοινωνιών

Ηλεκτρονικές Πηγές

http://www.altera.com

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

http://www.vhdl-online.de/

http://www.fpga4fun.com/

http://www.fpgajournal.com/

http://www.fpga-guide.com/startpage_frame.html

http://en.wikipedia.org/wiki/Field-programmable_gate_array

http://www.ashenden.com.au/designers-guide/VHDL-quick-start/ppframe.htm

http://www2.uic.edu/~wahmad1/quartus_ii_tutorial.pdf

http://esd.cs.ucr.edu/labs/tutorial/

http://www.d.umn.edu/~gshute/spimsal/talref.html

http://coe.uncc.edu/~amukherj/INTRO2VHDL/alu.eg1.pdf

http://users.eecs.northwestern.edu/~debjit/files/361_Project.pdf

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