ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

Σχεδίαση σε VHDL και υλοποίηση σε FPGA Μονάδας Παραγωγής Μουσικού Σήματος

Πτυχιακή Εργασία

Ασβεστόπουλος Θεόδωρος (Α.Ε.Μ. 1370)

Τσιώτρα Ειρήνη (Α.Ε.Μ. 899)

Επιβλέπων: Δρ. Καλόμοιρος Ιωάννης, Επίκ. Καθηγητής

ΣΕΡΡΕΣ, 2014

Υπεύθυνη Δήλωση:

Βεβαιώνουμε ότι είμαστε συγγραφείς αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης, έχουμε αναφέρει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Εν κατακλείδι, βεβαιώνουμε ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμάς προσωπικά, ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής Τ.Ε. Σερρών.

Ασβεστόπουλος Θεόδωρος

Τσιώτρα Ειρήνη

Περιεχόμενα

Περίληψη

Σ’ αυτή την πτυχιακή εργασία, σχεδιάστηκε και υλοποιήθηκε μια απλή μονάδα επεξεργασίας, που αναπαράγει μουσικό σήμα με τη βοήθεια ενός βομβητή. Η μονάδα σχεδιάστηκε σε γλώσσα περιγραφής υλικού VHDL και υλοποιήθηκε σε Πίνακα Πυλών Προγραμματιζόμενο στο Πεδίο (FPGA) της εταιρίας Altera, με χρήση της διάταξης EPF10KTC144 – 4 και του αναπτυξιακού κυκλώματος LP2900.

Η μονάδα μουσικού σήματος μπορεί να παράγει οκτώ βασικούς μουσικούς τόνους, που αντιστοιχούν στην τέταρτη οκτάβα του πενταγράμμου. Η δυνατότητα αυτή μπορεί εύκολα να επεκταθεί για περισσότερες οκτάβες.

Το μουσικό κομμάτι εγγράφεται σε μνήμη ROM και αναπαράγεται με τη βοήθεια απαριθμητή προγράμματος. Οι καταχωρημένες εντολές αποκωδικοποιούνται και εκτελούνται σειριακά. Κάθε εντολή κωδικοποιεί τη συχνότητα του μουσικού τόνου και τη χρονική του διάρκεια.

Στην εργασία παρουσιάζονται αναλυτικά οι παραπάνω βαθμίδες.

Πρόλογος

Σκοπός της πτυχιακής εργασίας είναι η σχεδίαση σε VHDL (γλώσσα περιγραφής υλικού) και υλοποίηση σε FPGA (Διατάξεις Πυλών Προγραμματιζόμενες στο Πεδίο) μουσικού επεξεργαστή για την παραγωγή ενός μουσικού σήματος μέσω της αναπτυξιακής πλακέτας LP – 2900 της εταιρείας Leap Electronic Co.

Στο 1^{ο} κεφάλαιο ακολουθεί μια περιληπτική αναφορά για τη γλώσσα περιγραφής υλικού VHDL και το εργαλείο σχεδίασης Quartus II. Ο κώδικας γράφτηκε σε γλώσσα VHDL και το πρόγραμμα σχεδίασης που χρησιμοποιήθηκε είναι το λογισμικό Quartus II.

Στο 2^{ο} κεφάλαιο γίνεται μια αναφορά για τις λογικές διατάξεις πυλών προγραμματιζόμενες στο πεδίο (FPGA). Περιγράφεται η γενική δομή μιας διάταξης FPGA, τα μέρη από τα οποία αποτελείται (ακροδέκτες εισόδου / εξόδου, λογικές βαθμίδες, γραμμές διασύνδεσης) καθώς και ο προγραμματισμός ενός FPGA.

Στο 3^{ο} κεφάλαιο ακολουθεί μια αναφορά για το αναπτυξιακό κύκλωμα LP – 2900 της εταιρείας Leap Electronic Co. Η αναπτυξιακή πλακέτα LP – 2900 χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής μας, βασίζεται στο ολοκληρωμένο FPGA EPF10KTC144 – 4 και ανήκει στην οικογένεια Altera.

Στο 4^{ο} κεφάλαιο ακολουθεί μια αναλυτική περιγραφή του μουσικού επεξεργαστή. Περιγράφεται ο τρόπος λειτουργίας του και αναλύονται όλα τα επιμέρους κυκλώματα από τα οποία αποτελείται.

Τέλος, στο 5^{ο} κεφάλαιο παρουσιάζεται ο κώδικας του μουσικού επεξεργαστή. Δημιουργήθηκαν δέκα αρχεία εκ τον οποίων τα επτά αρχεία περιέχουν τον κώδικα των κυκλωμάτων, τα δύο αρχεία περιέχουν τις δηλώσεις μεταβλητών των κυκλωμάτων και το ένα αρχείο περιέχει τη δήλωση του πίνακα ROM του κυκλώματος της μνήμης ROM.

Η πτυχιακή εργασία έγινε στο τμήμα Μηχανικών Πληροφορικής Τ.Ε. το ακαδημαϊκό έτος 2013 - 2014 με επιβλέποντα καθηγητή τον κ. Καλόμοιρο Ιωάννη, τον οποίο θα θέλαμε να ευχαριστήσουμε θερμά κυρίως για την εμπιστοσύνη που μας έδειξε και την υπομονή που έκανε κατά τη διάρκεια υλοποίησης της πτυχιακής

εργασίας. Τον ευχαριστούμε επίσης, για την πολύτιμη βοήθεια και την καθοδήγησή του, για την επίλυση διαφόρων θεμάτων.

Θα θέλαμε επίσης να απευθύνουμε τις ευχαριστίες μας στους γονείς μας, οι οποίοι στήριξαν τις σπουδές μας με διαφόρους τρόπους, φροντίζοντας για την καλύτερη δυνατή μόρφωσή μας.

ΚΕΦΑΛΑΙΟ 1^{ο} – ΓΛΩΣΣΑ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ VHDL ΚΑΙ ΕΡΓΑΛΕΙΑ ΣΧΕΔΙΑΣΗΣ

1.1 Εισαγωγή στη γλώσσα VHDL

Η VHDL είναι μία γλώσσα περιγραφής υλικού, που χρησιμοποιείται για την ανάπτυξη ολοκληρωμένων ψηφιακών κυκλωμάτων και συστημάτων. Η VHDL αποτελεί συντόμευση των λέξεων VHSIC Hardware Description Language, όπου τα αρχικά VHSIC είναι συντόμευση των λέξεων Very High Speed Integrated Circuit (Ολοκληρωμένα Κυκλώματα Υψηλής Ταχύτητας).

Η αρχική έκδοση της VHDL έγινε γνωστή το 1987 με το πρότυπο IEEE 11076, ενώ αργότερα το 1993 εμφανίστηκε μια βελτιωμένη έκδοσή της με το πρότυπο IEEE 1164. Το πρότυπο στη συνέχεια αναβαθμίστηκε το 2000 και το 2002.

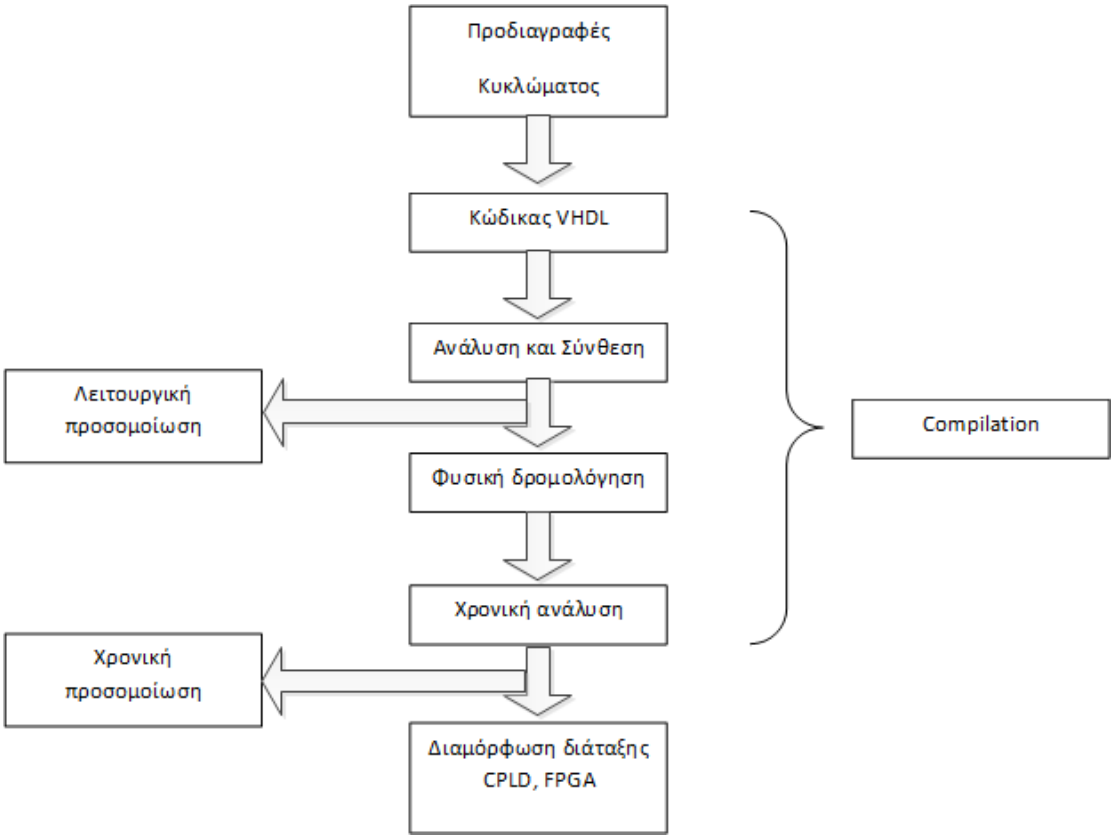
Η γλώσσα VHDL αρχικά είχε ένα διττό σκοπό. Πρώτον, να αποτελεί μια γλώσσα κειμένου που να περιγράφει τη δομή περίπλοκων ψηφιακών κυκλωμάτων. Δεύτερον, παρείχε δυνατότητες μοντελοποίησης ως προς τη συμπεριφορά των ψηφιακών κυκλωμάτων και μπορούσε να χρησιμοποιηθεί ως είσοδος σε προγράμματα λογισμικού που προσομοίωναν τη συμπεριφορά των ψηφιακών κυκλωμάτων.

Τα τελευταία χρόνια, η γλώσσα VHDL πέρα από τη χρήση της για καταγραφή και προσομοίωση χρησιμοποιείται ευρέως για την εισαγωγή σχεδίων σε συστήματα σχεδίασης CAD (Computer Aided Design). Επίσης, χρησιμοποιείται για την περιγραφή και υλοποίηση ψηφιακών συστημάτων σε προγραμματιζόμενες λογικές διατάξεις, τύπου CPLDs (Complex Programmable Logic Devices-Σύνθετες προγραμματιζόμενες λογικές διατάξεις) και FPGAs (Field Programmable Gate Arrays-Διατάξεις πυλών προγραμματιζόμενες στο πεδίο).

Ας αναφερθεί, ότι η γλώσσα VHDL δανείζεται πολλά χαρακτηριστικά από τη γλώσσα προγραμματισμού ADA (όσον αφορά τις έννοιες και τη σύνταξη).

1.1.1 Ροή σχεδίασης

Στο παρακάτω σχήμα 1.1. φαίνεται το διάγραμμα ροής σχεδίασης στη VHDL. Αρχικά τίθενται οι προδιαγραφές του κυκλώματος και στη συνέχεια γράφεται ο κώδικας VHDL. Έπειτα, γίνεται η μεταγλώττιση (compilation) όπου αποτελείται από τα εξής μέρη:



Σχήμα 1.1 Διάγραμμα ροής της γλώσσας VHDL (από το λήμμα [1] της βιβλιογραφίας)

- **Ανάλυση**, όπου γίνεται η επεξεργασία του κώδικα για τυχόν συντακτικά λάθη και η διόρθωση τους από το χρήστη.
- **Σύνθεση**, όπου ο μεταγλωττιστής (compiler) σχεδιάζει το κύκλωμα που περιγράφει ο κώδικας. Υπάρχει διαφορετικό αποτέλεσμα σύνθεσης αν διαμορφώσουμε ένα κύκλωμα CPLD ή ένα FPGA καθώς το καθένα έχει τις δικές του βαθμίδες για τη δημιουργία λογικών συναρτήσεων. Το CPLD

χρησιμοποιεί πίνακες πυλών AND και OR (Logic Arrays), ενώ το FPGA πίνακες αναφοράς (Look-up Tables-LUTs).

- Εκτελείται ένα πρώτο στάδιο **λειτουργικής προσομοίωσης** (functional simulation) όπου δεν προστίθενται ακόμη οι χρονικοί περιορισμοί του τελικού κυκλώματος.

- **Φυσική δρομολόγηση**, όπου κάθε λογική δομή η οποία έχει παραχθεί από τη σύνθεση βρίσκει τη φυσική της αντιστοίχηση σε μια λογική βαθμίδα μέσα στη διάταξη.
  - Έπειτα ακολουθεί η **χρονική προσομοίωση** (timing simulation) της σχεδίασης.
- Εφόσον όλα τα προηγούμενα στάδια έχουν υλοποιηθεί επιτυχώς ακολουθεί η **διαμόρφωση διάταξης** όπου δέχεται σήματα εισόδου και παράγει τα αναμενόμενα σήματα εξόδου.

1.1.2 Σχόλια σε προγράμματα VHDL

Κατά τη γραφή κώδικα στη VHDL μπορούν να συμπεριληφθούν σχόλια. Η δήλωση ενός σχόλιου δηλώνεται από δύο χαρακτήρες ¨-¨ και ¨-¨. Ο μεταφραστής της γλώσσας VHDL αγνοεί ότι ακολουθεί τους χαρακτήρες ¨--¨ σε μια γραμμή.

```
-- this is a VHDL comment
```

1.2 Αντικείμενα δεδομένων

Οι πληροφορίες στη VHDL παριστάνονται ως αντικείμενα δεδομένων (data objects). Υπάρχουν τρία είδη αντικειμένων δεδομένων:

- **Σήματα (Signals)**, όπου είναι τα πιο σημαντικά αντικείμενα δεδομένων. Αντιπροσωπεύουν τα λογικά σήματα (δηλαδή τα καλώδια) του κυκλώματος.

- **Μεταβλητές (Variables)**, όπου προσωρινά αποθηκεύουν τιμές που προκύπτουν από την τέλεση αριθμητικών πράξεων.
- **Σταθερές (Constants)**, όπου λαμβάνουν τιμή κατά τη δήλωσή τους η οποία παραμένει στη συνέχεια σταθερή. Δηλαδή, η σταθερά απλώς μεταφέρει μια συγκεκριμένη αριθμητική τιμή, όπου χρειάζεται.

1.2.1 Ονόματα αντικειμένων δεδομένων

Οι κανόνες για να ορίσουμε τα ονόματα των αντικειμένων δεδομένων είναι απλοί. Μπορούν να έχουν οποιονδήποτε αλφαριθμητικό χαρακτήρα πεζό ή κεφαλαίο καθώς και τον χαρακτήρα ¨-¨. Υπάρχουν τέσσερις περιορισμοί:

- Ένα όνομα δεν είναι δυνατό να είναι κάποια λέξη – κλειδί της γλώσσας VHDL (π.χ. ENTITY, ARCHITECTURE κ.τ.λ.)
- Πρέπει να ξεκινά από ένα γράμμα (π.χ. counter1)
- Δεν είναι δυνατό να αρχίζει ή να τελειώνει με το χαρακτήρα ¨_¨ (π.χ. _counter ή counter_)
- Δεν μπορεί να έχει δύο διαδοχικούς χαρακτήρες ¨_¨ (π.χ. count__er)

Ας αναφέρουμε ότι στη VHDL δεν υπάρχει διαχωρισμός μεταξύ πεζών και κεφαλαίων γραμμάτων, δηλαδή το ‘k’ είναι ίδιο με το ‘K’.

1.2.2 Δήλωση αντικειμένων δεδομένων

- Δήλωση σταθεράς:

  CONSTANT όνομα σταθεράς : τύπος := τιμή σταθεράς;

- Δήλωση σημάτων:

  SIGNAL όνομα σήματος : τύπος σήματος;

- Δήλωση μεταβλητών:

  VARIABLE όνομα μεταβλητής : τύπος μεταβλητής;

1.2.3 Τύποι δεδομένων

Στην παρούσα παράγραφο περιγράφουμε τους σημαντικότερους τύπους σημάτων.

- **BIT**

  Τα σήματα αυτού του τύπου μπορούν να έχουν τιμές ΄0΄ ή ΄1΄.

  Signal x1 : bit;

- **BIT_VECTOR**

  Τα σήματα αυτού του τύπου μπορούν να λάβουν μια σειρά τιμών από ΄0΄ ή ΄1΄. Δηλαδή, είναι μια γραμμική σειρά αντικειμένων τύπου BIT.

  Signal c : bit_vector (6 downto 0);

- **INTEGER**

  Ένα σήμα αυτού του τύπου μεταφέρει ακέραιες τιμές. Αντιπροσωπεύει ένα δυαδικό αριθμό. Έχει μήκος 32 bits και μπορεί να αντιπροσωπεύει οποιοδήποτε αριθμό από $-(2^{31} - 1)$ έως $-(2^{31} - 1)$. Επίσης, μπορούν να δηλωθούν ακέραιοι αριθμοί με λιγότερα bits με τη βοήθεια της λέξης – κλειδί RANGE.

  Signal x : integer range -127 to 127;

  Με τον τρόπο αυτό ο αριθμός x ορίζεται ότι είναι ένας προσημασμένος αριθμός μήκους οκτώ bits.

- **NATURAL**

  Είναι υποτύπος του integer και περιλαμβάνει μη αρνητικούς ακεραίους (από 0 μέχρι και το άνω όριο των ακεραίων).

  Signal f : natural range 0 to 16;

- **POSITIVE**

  Είναι υποτύπος του integer και περιλαμβάνει μόνο θετικούς ακεραίους.

1.2.4 Τύποι STD_LOGIC και STD_LOGIC_VECTOR

Αυτοί οι τύποι προστέθηκαν στο πρότυπο IEEE 1164 της γλώσσας VHDL. Για τη χρησιμοποίηση του τύπου Std_logic πρέπει να περιλάβουμε δύο εντολές στην περιοχή δηλώσεων βιβλιοθήκης. Αυτές είναι οι ακόλουθες:

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

Ένα σήμα τύπου Std_logic μπορεί και παίρνει τις τιμές ΄0΄και ΄1΄ όπως ένα σήμα τύπου BIT καθώς και άλλες επτά τιμές:

- **΄0΄** Λογικό μηδέν
- **΄1΄** Λογικό ένα
- **΄Z΄** Υψηλή εμπέδηση
- **΄-΄** Αδιάφορη λογική κατάσταση
- **΄U΄** Μη μηδενισμένη τιμή
- **΄X΄** Άγνωστη τιμή
- **΄L΄** Ασθενής τιμή
- **΄H΄** Ασθενής υψηλή τιμή
- **΄W΄** Ασθενής άγνωστη τιμή

Ο τύπος Std_logic_vector αντιπροσωπεύει μια σειρά αντικειμένων τύπου Std_logic. Δηλαδή, τα σήματα τύπου Std_logic_vector αποτελούν έναν πίνακα από τις παραπάνω εντολές και όχι μόνο μια από αυτές (π.χ. 01100110101).

```
Signal X1, Cin : std_logic;   --1 bit
Signal C : std_logic_vector (4 downto 0);   --5 bit
```

Για τη χρήση των τύπων std_logic και std_logic_vector πρέπει να δηλωθεί επιπλέον ένα από τα πακέτα std_logic signed ή std_logic_unsigned.

- **Std_logic_signed**

  Ο μεταφραστής δημιουργεί ένα κύκλωμα το όποιο δέχεται προσημασμένους αριθμούς.

- **Std_logic_unsigned**

  Ο μεταφραστής δημιουργεί ένα κύκλωμα το όποιο δέχεται μη προσημασμένους αριθμούς.

1.2.5 Διάφοροι τύποι

Στη γλώσσα VHDL υπάρχουν και κάποιοι άλλοι τύποι δεδομένων που μπορεί να χρησιμοποιήσει ο χρήστης.

- **Τύποι που ορίζονται από το χρήστη**

  Ο χρήστης πέρα από τους προηγούμενους τύπους που αναφέραμε μπορεί να δημιουργήσει δικούς του τύπους δεδομένων ώστε να μπορεί να διαχειριστεί ευκολότερα τη σχεδίαση ενός συστήματος. Η δήλωση του συγκεκριμένου τύπου γίνεται στο σώμα της αρχιτεκτονικής ή σε ξεχωριστό πακέτο (package).

  TYPE όνομα_τύπου IS περιγραφή τύπου;

- **Ακέραιοι τύποι ορισμένοι από τον χρήστη**

  Στην περίπτωση που ο χρηστής θέλει να χρησιμοποιήσει ένα σύνολο ακεραίων τιμών.

  TYPE όνομα_τύπου IS RANGE αρχική_τιμή TO τελική_τιμή;

- **Τύποι απαρίθμησης**

  Περιλαμβάνει ένα σύνολο τιμών με τη μορφή συμβόλων.

  TYPE όνομα_τύπου IS (λίστα τιμών);

- **Τύποι πινάκων**

  Δημιουργία ενός πίνακα που μπορεί να είναι μονοδιάστατος ή παραπάνω διαστάσεων.

  TYPE όνομα_τύπου IS ARRAY (περιοχή δεικτών) OF τύπος_στοιχείων;

1.2.6 Μετατροπή τύπων

Η γλώσσα VHDL δεν επιτρέπει την τιμή ενός σήματος κάποιου άλλου τύπου να αντιστοιχηθεί σε ένα σήμα κάποιου άλλου τύπου. Όταν ο χρήστης θέλει να μετατρέψει διαφορετικούς τύπους αντικειμένων θα πρέπει να γίνει μετατροπή του ενός τύπου αντικειμένου στον τύπο του άλλου. Η εντολή που χρησιμοποιούμε είναι η CONV και είναι η ακόλουθη:

σήμα_1 <= CONV_τύπος_σήμα_1 (σήμα_2, αριθμός bits σήμα_1);

Αν για παράδειγμα έχουμε ένα σήμα C τύπου BIT_VECTOR 5 bits και ένα δεύτερο σήμα X τύπου STD_LOGIC_VECTOR 2 bits η μετατροπή του δεύτερου σήματος στο πρώτο θα είναι η ακόλουθη:

C <= CONV_BIT_VECTOR (X, 5);

1.3 Τελεστές

Η γλώσσα VHDL διαθέτει λογικούς τελεστές, τελεστές σύγκρισης και αριθμητικούς τελεστές.

1.3.1 Λογικοί τελεστές

Η VHDL υποστηρίζει τις λογικές πράξεις. Τα αποτελέσματα των πράξεων είναι του ίδιου τύπου και μεγέθους με τους τελεστέους. Ακολουθεί το σχήμα 1.2 όπου φαίνονται οι λογικοί τελεστές.

Τελεστής	Λειτουργία
Not	Αντιστροφή
And	Και
Nand	Όχι και
Or	Ή
Nor	Ούτε
Xor	Αποκλειστικό Ή
Xnor	Αποκλειστικό Ούτε

Σχήμα 1.2 Τελεστές λογικών πράξεων

1.3.2 Σχεσιακοί τελεστές

Οι σχεσιακοί τελεστές ελέγχουν για ισότητα, ανισότητα και γενικά κάνουν συγκρίσεις μεταξύ σημάτων ή μεταβλητών. Όλες οι σχεσιακές πράξεις έχουν την ίδια προτεραιότητα και το αποτέλεσμα που προκύπτει είναι πάντα τύπου Boolean (TRUE / FALSE). Παρακάτω, στο σχήμα 1.3 φαίνονται οι σχεσιακοί τελεστές.

Τελεστής	Λειτουργία
=	Ισότητα
/=	Διάφορο
<	Μικρότερο
<=	Μικρότερο ή ίσο

>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο

Σχήμα 1.3 Τελεστές σχεσιακών πράξεων

1.3.3 Αριθμητικοί τελεστές

Οι αριθμητικές πράξεις θεωρούνται γνωστές και φαίνονται στο σχήμα 1.4 .

Τελεστής	**Πράξη**
+	Πρόσθεση
-	Αφαίρεση
*****	Πολλαπλασιασμός
/	Διαίρεση
****	Ύψωση σε δύναμη
ABS	Απόλυτη τιμή
REM	Υπόλοιπο (Remainder)
MOD	Modulo

Σχήμα 1.4 Αριθμητικοί τελεστές

1.4 Δομή σχεδίασης

Η περιγραφή ενός κυκλώματος που γίνεται σε γλώσσα VHDL ονομάζεται οντότητα. Η γενική δομή μιας οντότητας ή ενός προγράμματος VHDL αποτελείται από δύο μέρη, τη δήλωση οντότητας (entity) και την αρχιτεκτονική (architecture). Παρακάτω, στο σχήμα 1.5 φαίνεται η γενική δομή μιας οντότητας.

Σχήμα 1.5 Δομή κώδικα της γλώσσας VHDL

1.4.1 Δήλωση οντότητας (ENTITY)

Στο τμήμα του κώδικα όπου γίνεται η δήλωση της οντότητας ορίζονται ποια θα είναι τα σήματα εισόδου και εξόδου του κυκλώματος. Το όνομα της οντότητας μπορεί να είναι οποιοδήποτε έγκυρο όνομα της γλώσσας VHDL και τα σήματα εισόδου ή εξόδου καθορίζονται με τη λέξη PORT (Θύρα). Μία θύρα μπορεί να λάβει τις ακόλουθες καταστάσεις:

- IN, αποτελεί είσοδο σε μια οντότητα
- OUT, αποτελεί έξοδο σε μια οντότητα
- INOUT, αποτελεί είσοδο και έξοδο σε μια οντότητα
- BUFFER, αποτελεί έξοδο σε μια οντότητα που διαβάζεται και εσωτερικά

  Στην περίπτωση που δεν καθορίζεται ποια είναι η κατάσταση της θύρας, θεωρείται ότι είναι είσοδος σε μια οντότητα (IN).

  Παρακάτω, ακολουθεί η δήλωση της οντότητας:

entity *όνομα οντότητας* **is**

Port *(όνομα σήματος* : **mode** *τύπος ;*

…

όνομα σήματος : **mode** *τύπος) ;*

end *όνομα οντότητας ;*

1.4.2 Αρχιτεκτονική (ARCHITECTURE)

Η αρχιτεκτονική περιλαμβάνει όλες τις λεπτομέρειες λειτουργίας του κυκλώματος. Αποτελείται από την περιοχή δήλωσης (declarative region) και το σώμα της αρχιτεκτονικής (architecture body). Η περιοχή της δήλωσης αρχίζει πριν από τη λέξη-κλειδί BEGIN, όπου μπορούν να δηλωθούν σήματα, σταθερές, διάφοροι τύποι από το χρήστη, συνιστώσες (COMPONENTS) και τα χαρακτηριστικά τους (ATTRIBUTES). Η λειτουργία του κυκλώματος ορίζεται στο σώμα της αρχιτεκτονικής έπειτα από τη λέξη BEGIN. Εκτελούνται εντολές που καθορίζουν τη λειτουργία του κυκλώματος. Ο ορισμός της αρχιτεκτονικής είναι ο εξής:

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

1.4.3 Παράδειγμα ενός πολυπλέκτη 2:1 (8 bits)

Στο παράδειγμα περιγράφουμε έναν πολυπλέκτη όπου στην οντότητα (ENTITY) καθορίζονται δύο είσοδοι a και b των 8 bits, μια είσοδος sel του 1 bit, όπου είναι ο

διακόπτης επιλογής του πολυπλέκτη και τέλος μια έξοδος f των 8 bits. Στο κομμάτι της αρχιτεκτονικής (ARCHITECTURE) περιγράφεται ποια θα είναι η λειτουργία του κυκλώματος χρησιμοποιώντας στο συγκεκριμένο παράδειγμα την εντολή WITH. Δηλαδή, η έξοδος f παίρνει την τιμή a όταν ο διακόπτης επιλογής sel = '0' και την τιμή b στην άλλη περίπτωση (όταν sel = '1') .

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------ENTITY-----------------------------------------
ENTITY mux IS
PORT (a,b : IN std_logic_vector (7 downto 0);
sel : IN std_logic;
f    : OUT std_logic_vector (7 DOWNTO 0));
END mux;
---------------------------ARCHITECTURE-----------------------------------
ARCHITECTURE behaviour OF mux IS
BEGIN
WITH sel SELECT
f<= a WHEN '0',
b WHEN OTHERS;
END behaviour;
```

1.5 Υποκυκλώματα και πακέτα

1.5.1 Υποκυκλώματα

Η VHDL είναι μια γλώσσα που επιτρέπει την ιεραρχική σχεδίαση κυκλωμάτων. Έτσι, ένα αρχείο προγράμματος μπορεί να χρησιμοποιηθεί ως υποκύκλωμα σε ένα άλλο αρχείο προγράμματος. Τα υποκυκλώματα που συμπεριλαμβάνονται σε μια ανώτερη οντότητα (top – level entity) ονομάζονται συνιστώσες (components). Οι συνιστώσες είναι μικρότερα κυκλώματα που ήδη έχουν περιγραφεί και μπορούν να κληθούν ως αρχεία βιβλιοθήκης. Τα κύρια χαρακτηριστικά της σχεδίασης με

υποκυκλώματα είναι η δήλωση των συνιστωσών που περιέχονται στο κύκλωμα και η περιγραφή του τρόπου με τον οποίο τα στοιχεία συνδέονται μεταξύ τους.

Με τη δήλωση της κάθε συνιστώσας καθορίζουμε το όνομά της καθώς και τις εισόδους και εξόδους της. Η δήλωση της μπορεί να υπάρχει είτε στην περιοχή δηλώσεων της αρχιτεκτονικής είτε σε μια δήλωση πακέτου (PACKAGE). Η γενική μορφή μιας τέτοιας δήλωσης είναι η εξής:

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

Εφόσον γίνει η δήλωση της συνιστώσας πρέπει στο κύριο σώμα της αρχιτεκτονικής να δημιουργήσουμε στιγμιότυπα για τα υποκυκλώματα που δηλώσαμε και να περιγράψουμε τις συνδέσεις τους. Η γενική μορφή δημιουργίας στιγμιότυπου συνιστώσας είναι η ακόλουθη:

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

Στο παρακάτω παράδειγμα παρουσιάζεται η χρήση ενός υποκυκλώματος. Δημιουργήσαμε ένα πρόγραμμα ενός πλήρη αθροιστή του ενός Bit. Όπως γνωρίζουμε ο πλήρης αθροιστής δημιουργείται με τη χρήση ημιαθροιστών. Αναλυτικά, ο πλήρης αθροιστής ενός bit αποτελείται από δύο ημιαθροιστές και μια πύλη OR. Παρακάτω παρουσιάζεται ο κώδικας ενός ημιαθροιστή:

```
Entity halfadder is
Port(a, b : in std_logic;
S, cout : out std_logic);
End halfadder;
Architecture behavior of halfadder is
Begin
S<= a xor b;
cout= a and b;
end behavior;
```

Στον κώδικα του πλήρη αθροιστή χρησιμοποιήθηκε ως υποκύκλωμα ο κώδικας του ημιαθροιστή.

```
Entity full_adder is
Port(x, y, cin : in std_logic;
sum, c : out std_logic));
End full_adder;
Architecture behavior of full_adder is
Signal n1, n2, d : std_logic;
Component halfadder
Port (a, b : in std_logic;
S, cout: out std_logic));
end component;
begin
stage_0: halfadder port map(cin, n1, sum, d);
stage_1: halfadder port map (x, y, n1, n2);
cout<= n2 or d;
end behavior;
```

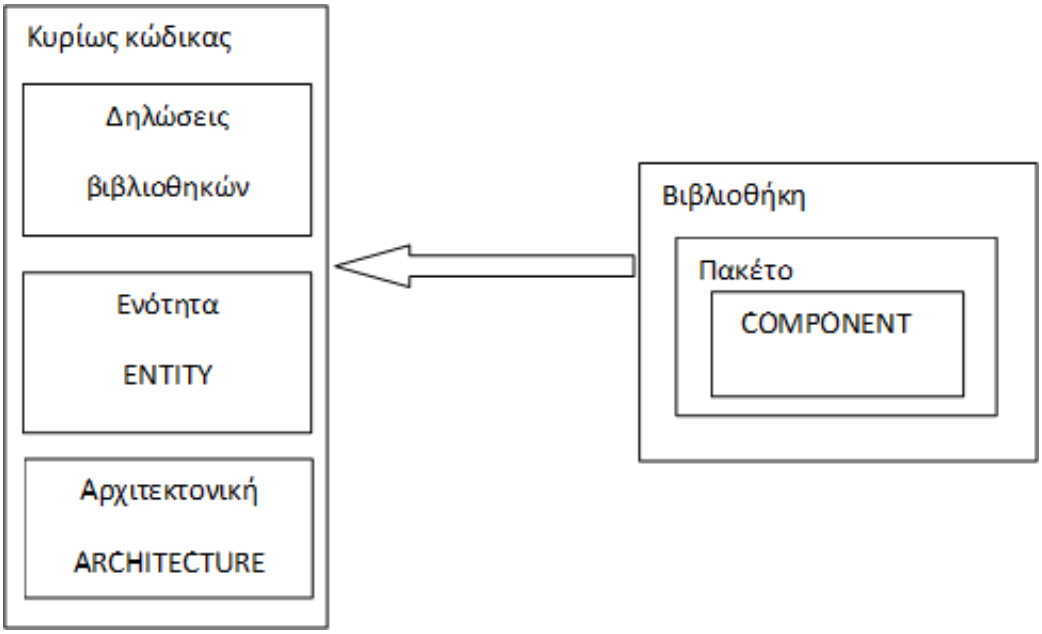
Στην περιοχή της αρχιτεκτονικής δηλώνεται το κύκλωμα του ημιαθροιστή (Component halfadder). Είναι απαραίτητο να δηλωθούν οι ακροδέκτες του υποκυκλώματος αν είναι είσοδοι ή έξοδοι καθώς και το μέγεθος τους. Τα ονόματα των ακροδεκτών του halfadder πρέπει να είναι ίδια με αυτά που είχαν δηλωθεί στο αρχικό κύκλωμα. Έπειτα, στο σώμα της αρχιτεκτονικής δηλώνονται τα σήματα n1, n2 και d όπου χρησιμοποιούνται για την εσωτερική σύνδεση των υποκυκλωμάτων.

Στο κύριο σώμα της αρχιτεκτονικής χρησιμοποιούνται δύο στιγμιότυπα, stage_0 και stage_1. Με αυτόν τον τρόπο καλούνται τα υποκυκλώματα και γίνεται η αντιστοίχηση των ακροδεκτών τους με τους ακροδέκτες του ανώτερου ιεραρχικού κυκλώματος. Η σειρά των ακροδεκτών στα στιγμιότυπα θα πρέπει να είναι ίδια με αυτήν που δηλώθηκαν στο υποκύκλωμα (Component halfadder).

1.5.2 Πακέτα (Packages)

Ένα πακέτο (Package) της γλώσσας VHDL λειτουργεί όπως μια αποθήκη. Δηλαδή, λειτουργεί ως αποθήκη προγραμμάτων και δηλώσεων. Μπορούν να

συμπεριληφθούν και ορίσματα τύπων. Το πακέτο μπορεί να έχει δύο κύρια μέρη, το πακέτο (PACKAGE) και το σώμα του πακέτου (PACKAGE BODY). Στο πρώτο μέρος περιέχονται όλες οι δηλώσεις, ενώ το δεύτερο μέρος χρησιμοποιείται μόνο όταν το στο πρώτο μέρος δηλώνονται ένα ή περισσότερα υποπρογράμματα (συναρτήσεις), όπου στην περίπτωση αυτή πρέπει να περιέχονται οι περιγραφές των υποπρογραμμάτων. Το πακέτο μπορεί να αποτελείται από ένα σύνολο δηλώσεων και από ένα σύνολο υποκυκλωμάτων (components). Στο παρακάτω σχεδιάγραμμα φαίνεται η δομή ενός πακέτου (σχήμα 1.6) .



Σχήμα 1.6 Δομή πακέτου

Η γενική μορφή δήλωσης ενός πακέτου είναι η ακόλουθη:

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

PACKAGE BODY όνομα πακέτου **IS**

Κώδικας υποπρογράμματος (function ή procedure)

END PACKAGE BODY [όνομα πακέτου]

Μετά τη δημιουργία ενός πακέτου ο χρήστης μπορεί να το καλεί μέσα από άλλα προγράμματα. Θα πρέπει όμως να δηλωθεί η βιβλιοθήκη που το περιέχει καθώς και ποιο πακέτο από αυτήν τη βιβλιοθήκη θα χρησιμοποιήσει ο χρήστης. Η δήλωση γίνεται ως εξής:

```
library όνομα βιβλιοθήκης ;
use όνομα βιβλιοθήκης, όνομα πακέτου.all ;
```

Στο παρακάτω παράδειγμα ορίζεται το πακέτο fulladd_package ενός πλήρη αθροιστή.

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

PACKAGE fulladd_package IS
        COMPONENT fulladd
                PORT ( Cin, x, y: IN STD_LOGIC ;
                        s, Cout : OUT STD_LOGIC ) ;
        END COMPONENT ;
END fulladd_package ;
```

Έπειτα, παρουσιάζεται πως ένας πλήρης αθροιστής τεσσάρων bits μπορεί να γραφεί έτσι, ώστε να χρησιμοποιεί το πακέτο αυτό.

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

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

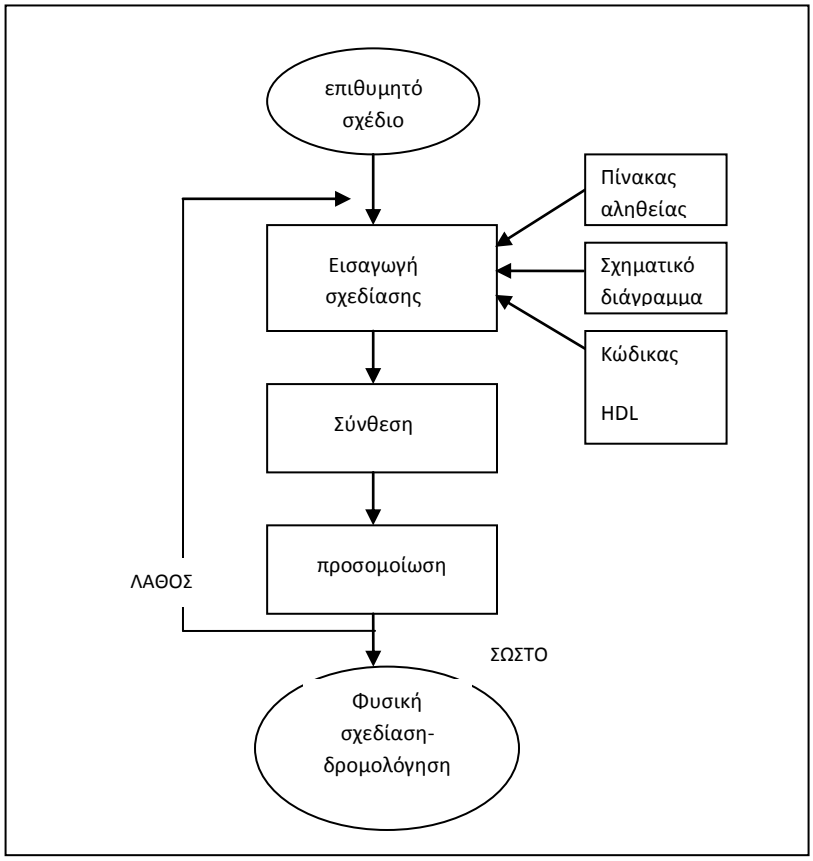
```
ARCHITECTURE Structure OF adder IS
        SIGNAL C : STD_LOGIC_VECTOR(1 TO 3) ;
BEGIN
        stage0: fulladd PORT MAP ( Cin, X(0), Y(0), S(0), C(1) ) ;
        stage1: fulladd PORT MAP ( C(1), X(1), Y(1), S(1), C(2) ) ;
        stage2: fulladd PORT MAP ( C(2), X(2), Y(2), S(2), C(3) ) ;
        stage3: fulladd PORT MAP ( C(3), X(3), Y(3), S(3), Cout ) ;
END Structure ;
```

1.6 Εργαλεία σχεδίασης ψηφιακών συστημάτων

1.6.1 Εισαγωγή

Η ανάγκη για τη δημιουργία μεγαλυτέρων και πολυπλοκότερων λογικών κυκλωμάτων κάνει αναγκαία τη χρήση ειδικών εργαλείων λογισμικού για τη σχεδίαση και την προσομοίωση (Cad tools). Η χρήση των εργαλείων αυτών έχουν ως σκοπό την αναλυτική προσομοίωση του κυκλώματος και τη δημιουργία απαραίτητων αρχείων για τον προγραμματισμό κατάλληλου υλικού. Αναφερόμαστε σε προγραμματιζόμενο υλικό, όπως είναι τα ολοκληρωμένα κυκλώματα CPLDs και FPGAs.

Ένας τρόπος με τον οποίο εισάγεται το κύκλωμα είναι η σχεδίαση του με τη βοήθεια βαθμίδων (blocks) που λαμβάνονται από βιβλιοθήκες. Ένας δεύτερος και εξίσου διαδεδομένος τρόπος εισαγωγής είναι τα αρχεία γλώσσας περιγραφής υλικού (Hardware Discription Language – HDL), όπως είναι και η γλώσσα VHDL.



Σχήμα 1.7 Ροή εργασιών σε ένα σύστημα σχεδίασης (από το λήμμα [1] της βιβλιογραφίας)

Στο σχήμα 1.7 φαίνονται τα βασικά βήματα σχεδίασης χρησιμοποιώντας εργαλεία CAD. Πέρα από την εισαγωγή του κυκλώματος, τα επόμενα βήματα είναι η σύνθεση (synthesis), η προσομοίωση και η φυσική σχεδίαση ή δρομολόγηση (place and route).

Η σύνθεση είναι μια αυτόματη διαδικασία που επιτελεί το σχεδιαστικό λογισμικό, ώστε να διαμορφώσει το κύκλωμα, σύμφωνα με τους κανόνες της τεχνολογίας για την οποία προορίζεται το σχέδιο μας. Διαφορετικό αποτέλεσμα θα υπάρχει ανάμεσα στον προγραμματισμό ενός κυκλώματος CPLD και ενός κυκλώματος FPGA. Αυτό συμβαίνει γιατί ένα κύκλωμα CPLD υλοποιεί τις λογικές συναρτήσεις με βάση λογικούς πίνακες πυλών (Logic Arrays), που κατασκευάζονται με πύλες AND και OR, ενώ ένα κύκλωμα FPGA χρησιμοποιεί πίνακες αναφοράς (Look – up tables).

Η προσομοίωση υλοποιείται με τη βοήθεια κατάλληλων εισόδων, που ενεργούν στους 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 αρχεία που δημιουργούνται για την ενεργοποίηση των εισόδων καλούνται «waveform vectors». Στην περίπτωση που η προσομοίωση του κυκλώματος δεν είναι επιτυχής θα χρειαστεί επανασχεδιασμός του κυκλώματος για την αποφυγή λαθών (το βέλος ανάδρασης που υπάρχει στο σχήμα 1.7).

Όταν η προσομοίωση είναι επιτυχής προχωρούμε στο επόμενο και τελευταίο βήμα, τη δρομολόγηση του κυκλώματος (place and route ή fitting). Δημιουργούνται οι κατάλληλες συνδέσεις στον πίνακα διασυνδέσεων του κυκλώματος (Interconnection matrix), όπου συνδέονται με βέλτιστο τρόπο μεταξύ τους τα λογικά στοιχεία που εκτελούν τις επιμέρους λειτουργίες του κυκλώματος.

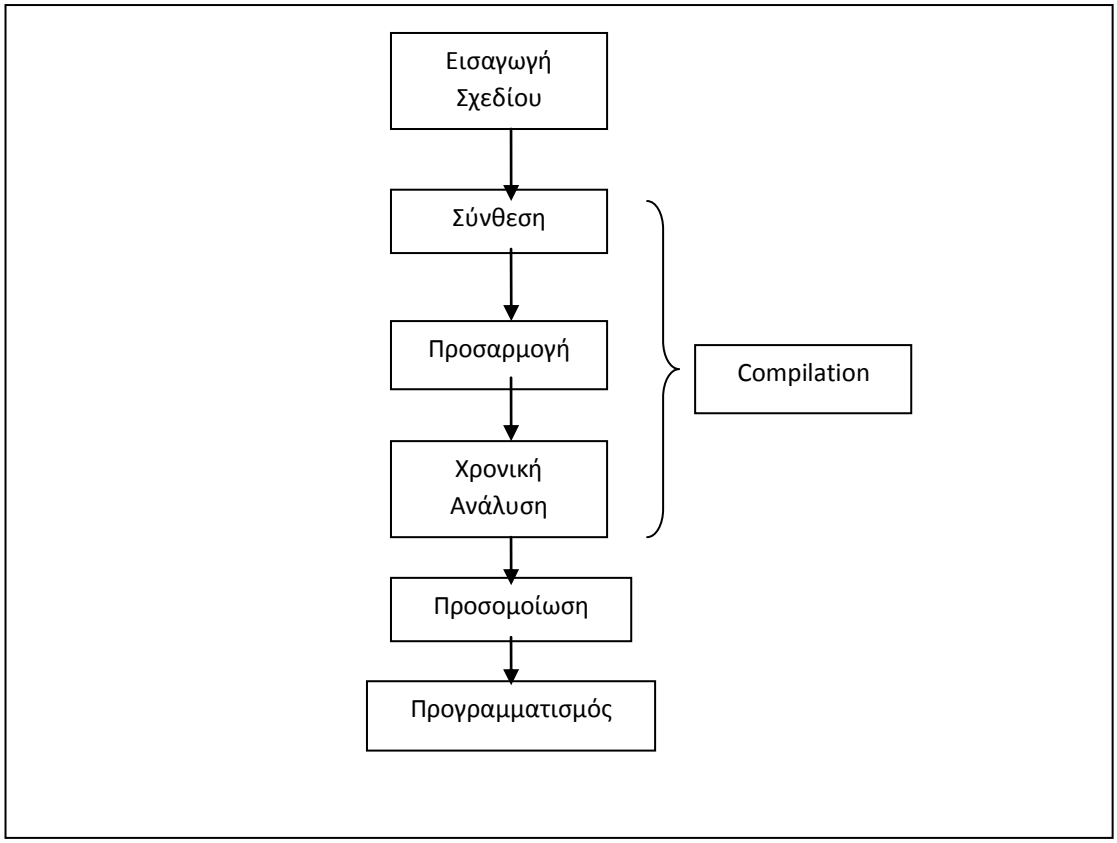
Ορισμένα λογισμικά σχεδίασης επιτρέπουν στο τέλος τον προγραμματισμό του κυκλώματος, ώστε το σχέδιο μας να αποτυπωθεί στο ολοκληρωμένο κύκλωμα για το οποίο προορίζεται, διαμορφώνοντάς το κατάλληλα.

1.6.2 Λογισμικό QUARTUS II

Το λογισμικό Quartus II είναι ένα ευρέως διαδεδομένο πρόγραμμα σχεδίασης CAD για την ανάλυση και τη σύνθεση VHDL κυκλωμάτων. Ανήκει στην εταιρεία ALTERA, η οποία είναι από τις γνωστότερες οικογένειες κατασκευής κυκλωμάτων CPLD και FPGAs. Αποτελείται από τα εξής εργαλεία:

- Εισαγωγή σχεδίου
- Λογική σύνθεση και βελτιστοποίηση
- Προσαρμογή (fitting)
- Χρονική ανάλυση
- Προσομοίωση (simulation)
- Προγραμματισμός και διαμόρφωση συσκευής

Παρακάτω στο σχήμα 1.8 φαίνεται η ροή εργασιών που ακολουθεί στο λογισμικό Quartus II.



Σχήμα 1.8 Ροή διεργασιών στο Quartus II (από το λήμμα [1] της βιβλιογραφίας)

1.6.3 Εισαγωγή σχεδίασης

Στην εισαγωγή σχεδίασης γίνεται η περιγραφή του κυκλώματος που επιθυμεί να υλοποιήσει ο χρήστης. Στα περισσότερα προγράμματα σχεδίασης υπάρχουν τρεις τρόποι περιγραφής ενός κυκλώματος. Αυτοί είναι η σχεδίαση με την εισαγωγή πινάκων αληθείας, η σχεδίαση με βάση τα σχηματικά διαγράμματα και η σχεδίαση με τη βοήθεια κάποιας γλώσσας περιγραφής υλικού (HDL).

Με 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. Ο συγκεκριμένος τρόπος δεν είναι ιδιαίτερα εύχρηστος γι’ αυτό είναι προτιμότερο να

χρησιμοποιείται μόνο για απλά και μικρά κυκλώματα. Για να δώσουμε είσοδο στο Quartus II χρησιμοποιούμε κάποιους Editors. Για την είσοδο στο Quartus II χρησιμοποιείται ο Waveform Editor.

Ο δεύτερος τρόπος εισαγωγής σχεδίασης είναι το σχηματικό διάγραμμα, όπου η σχεδίαση γίνεται με τη βοήθεια σχεδιαστικών εργαλείων και με τη χρήση υπάρχουσων βιβλιοθηκών του προγράμματος σχεδίασης. Στις βιβλιοθήκες του προγράμματος σχεδίασης ο χρήστης μπορεί να χρησιμοποιήσει από απλές πύλες έως και σύνθετα κυκλώματα. Για την αποφυγή δημιουργίας μεγάλων κυκλωμάτων, δίνεται η δυνατότητα στο χρήστη να χρησιμοποιήσει ιεραρχικές βαθμίδες (blocks). Δηλαδή, δημιουργούνται κυκλώματα που στο εσωτερικό τους περιέχουν άλλα μικρότερα κυκλώματα. Ο συγκεκριμένος τρόπος σχεδίασης είναι γνωστός και ως ιεραρχική σχεδίαση. Για την είσοδο στο Quartus II χρησιμοποιείται ο Block Editor.

Τέλος, ο τρίτος τρόπος είναι η σχεδίαση με τη βοήθεια κάποιας γλώσσας περιγραφής κυκλωμάτων. Περιγράφεται η λειτουργία του κυκλώματος με τη βοήθεια κατάλληλου κώδικα. Για την είσοδο στο Quartus II χρησιμοποιείται ο Text Editor.

1.6.4 Λογική σύνθεση και βελτιστοποίηση

Το επόμενο βήμα είναι η διαδικασία της σύνθεσης. Στο Quartus II η διαδικασία της σύνθεσης αναφέρεται ως «Analysis & Synthesis». Στη συγκεκριμένη διαδικασία η είσοδος μετατρέπεται στις κατάλληλες λογικές συναρτήσεις, έτσι ώστε να ταιριάζει στη τεχνολογία της συγκεκριμένης διάταξης που τελικά επιθυμεί να διαμορφώσει ο χρήστης. Η σύνθεση είναι ένα αυτόματο εργαλείο υλοποίησης του αρχικού κυκλώματος που δίνει ο σχεδιαστής για την επίτευξη του τελικού στόχου. Με τη σύνθεση ο χρήστης έχει και βελτιστοποίηση του κυκλώματος, δηλαδή το κύκλωμα στην έξοδο της σύνθεσης είναι καλύτερο από το αρχικό.

1.6.5 Προσαρμογή (fitting)

Η διαδικασία της προσαρμογής (fitting) λέγεται αλλιώς και δρομολόγηση (place and route). Σε αυτή τη διαδικασία χρησιμοποιείται η βάση δεδομένων που δημιουργήθηκε κατά την ανάλυση και σύνθεση και αντιστοιχίζεται η ψηφιακή λογική στους ελεύθερους πόρους της διάταξης – στόχου. Λαμβάνονται υπόψη οι χρονικές απαιτήσεις που τέθηκαν από τον χρήστη. Δηλαδή, καθορίζεται πόσα και ποια συγκεκριμένα λογικά στοιχεία (logic elements – LE 's) του ολοκληρωμένου κυκλώματος (chip) θα χρησιμοποιηθούν. Τέλος, επιλέγονται συνδέσεις από τον προγραμματιζόμενο πίνακα διασυνδέσεων (programmable interconnect), για την διασύνδεση των απαραίτητων LE 's μεταξύ τους.

1.6.6 Χρονική ανάλυση

Το επόμενο βήμα μετά την προσαρμογή είναι η χρονική ανάλυση (timing analysis). Στη διαδικασία αυτή παράγονται οι καλύτεροι και οι χειρότεροι χρόνοι του κυκλώματος, σύμφωνα με τις καθυστερήσεις που υπάρχουν κατά μήκος των διαδρομών, όπου οφείλονται στο μήκος καλωδίων και στον αριθμό των ενδιάμεσων βαθμίδων. Οι χρόνοι που παίρνουμε από την εκτέλεση της χρονικής ανάλυσης θα πρέπει να είναι συμβατοί με αυτούς που καθορίζονται από το ρολόι του συστήματος. Στη περίπτωση που υπάρχει μη συμβατότητας των χρόνων θα πρέπει να διορθώσει ο χρήστης αυτές τις αποκρίσεις, κάνοντας κατάλληλες χρονικές εκχωρήσεις (timing assignments) στο λογισμικό και επαναλαμβάνοντας τη δρομολόγηση.

Σε αυτό το στάδιο (assembling) δημιουργείται η παραγωγή συμβολικού (.hex) και δυαδικού (.sof) κώδικα. Το δεύτερο αρχείο (.sof) το χρησιμοποιούμε και στην αναπτυξιακή πλακέτα LP – 2900 για να προγραμματιστεί το FPGA και να είναι το σύστημα που δημιουργήσαμε.

1.6.7 Προσομοίωση (simulation)

Στη διαδικασία της προσομοίωσης (simulation) ελέγχεται εάν το κύκλωμα λειτουργεί σωστά. Ο έλεγχος της σωστής λειτουργίας πραγματοποιείται στο τμήμα των διεργασιών που ονομάζεται προσομοίωση. Συνδυάζονται δύο παράμετροι. Μία είναι το αρχικό μας σχέδιο και η άλλη είναι οι 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 τις εξόδου του κυκλώματος μέσω ενός πίνακα αληθείας ή μέσω ενός διαγράμματος χρονισμού.

Υπάρχουν δύο τύποι προσομοίωσης, η λειτουργική (functional) και η προσομοίωση χρονισμού (timing). Στην πρώτη περίπτωση δε λαμβάνονται υπόψη οι καθυστερήσεις των πυλών και διασυνδέσεων του κυκλώματος, μόνο ελέγχεται αν είναι σωστή η λογική συνάρτηση του κυκλώματος. Στη δεύτερη περίπτωση επαληθεύεται η ορθότητα του κυκλώματος με βάση τους χρονικούς περιορισμούς του.

1.6.8 Προγραμματισμός και διαμόρφωση της συσκευής

Το τελευταίο βήμα για την ολοκλήρωση της δημιουργίας κυκλώματος είναι η διαδικασία του προγραμματισμού. Τα CPLDs και τα FPGAs πρέπει να διαμορφωθούν για να υλοποιήσουν το κύκλωμα που σχεδιάσαμε. Το αρχείο που χρειαζόμαστε είναι το .sof αρχείο, το οποίο δημιουργήθηκε στη διαδικασία της χρονικής ανάλυσης.

Η εταιρεία Altera επιτρέπει τον προγραμματισμό των συσκευών της με δύο τρόπους. Ο πρώτος είναι μέσω του κυκλώματος διεπαφής JTAG και ο δεύτερος 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 (παράλληλη ή USB). Η σύνδεση με τον κατάλληλο οδηγό (driver) USB – Blaster ή BYTE – BLASTER.

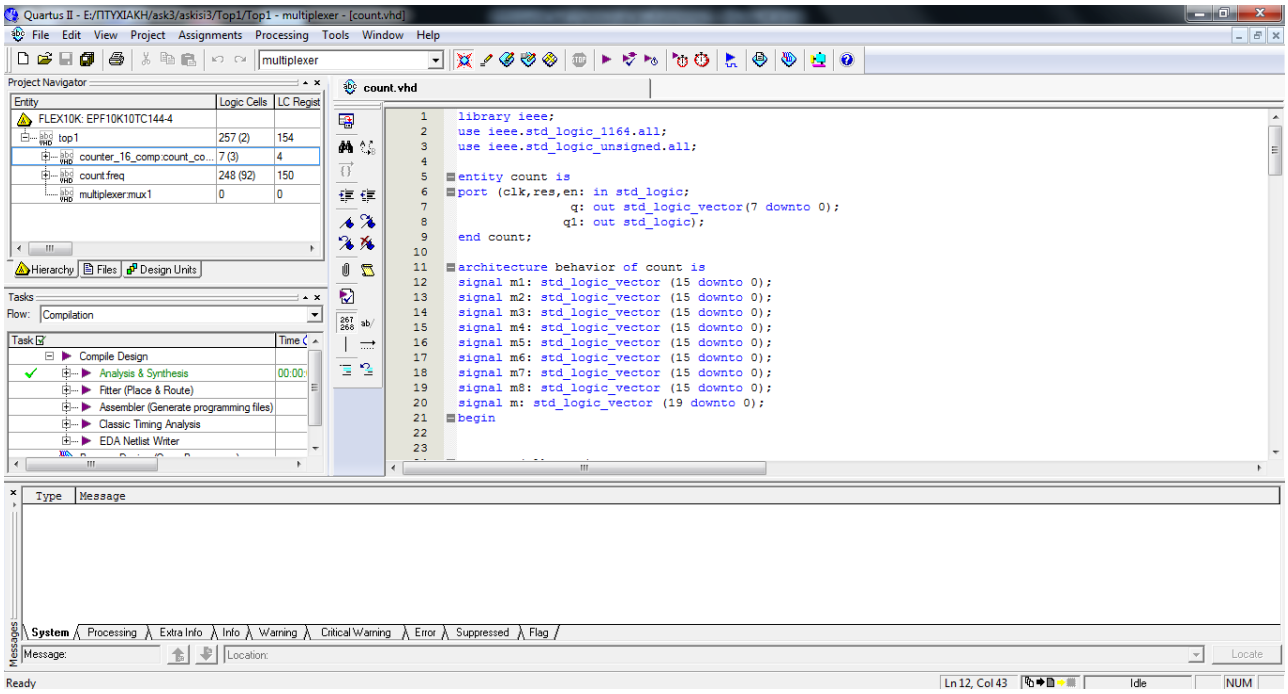
Στην πρώτη περίπτωση, όταν ο χρήστης χρησιμοποιεί την JTAG διεπαφή, τα δεδομένα πηγαίνουν κατευθείαν στο ολοκληρωμένο κύκλωμα. Το ολοκληρωμένο κύκλωμα (αν είναι FPGA) διατηρεί τη διαμόρφωση που του έχουμε δώσει για όσο διαρκεί η τροφοδοσία του. Όταν σταματήσει να τροφοδοτείται χάνεται και η διαμόρφωσή του. Το ίδιο δεν ισχύει για τα κυκλώματα CPLDs, όπως οι μνήμες flash EEPROM.

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

Τέλος, για να επιβεβαιωθεί αν το κύκλωμα λειτουργεί σωστά πρέπει ο χρήστης να δώσει κατάλληλες εισόδους 0 ή 1 από τους διακόπτες ή άλλες συσκευές εισόδου που βρίσκονται πάνω στο αναπτυξιακό κύκλωμα. Το αποτέλεσμα από τις εξόδους εμφανίζεται στους κατάλληλους ακροδέκτες εξόδου, που πρέπει να συνδεθούν με συσκευές απεικόνισης πάνω στην αναπτυξιακή πλακέτα.

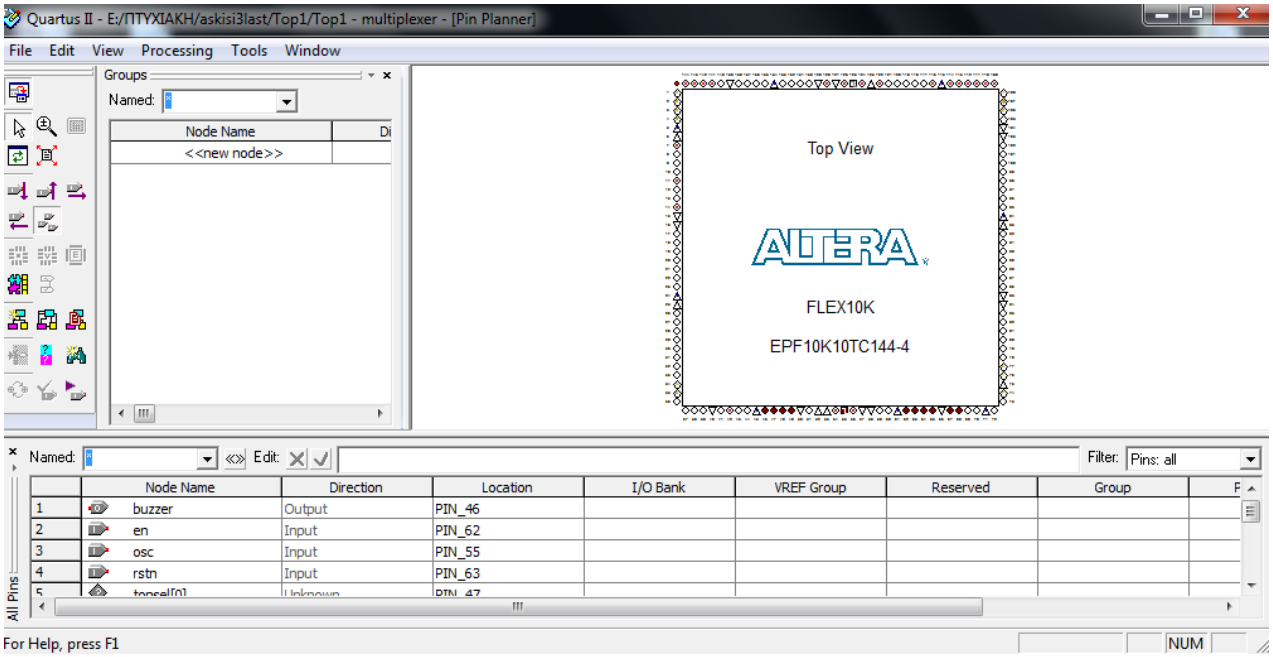
1.6.9 Περιβάλλον του QUARTUS II

Στο σχήμα 1.9 που ακολουθεί φαίνεται το περιβάλλον του Quartus II μετά τη δημιουργία του αρχείου counter.vhd.



Σχήμα 1.9 Δημιουργία αρχείου count.vhd σε περιβάλλον Quartus II

Στο σχήμα 1.10 που ακολουθεί φαίνεται η αντιστοίχηση των ακροδεκτών εισόδου και εξόδου της εφαρμογής μας για το FLEX10K.



Σχήμα 1.10 Παράθυρο ορισμού ακροδεκτών για το FLEX10K

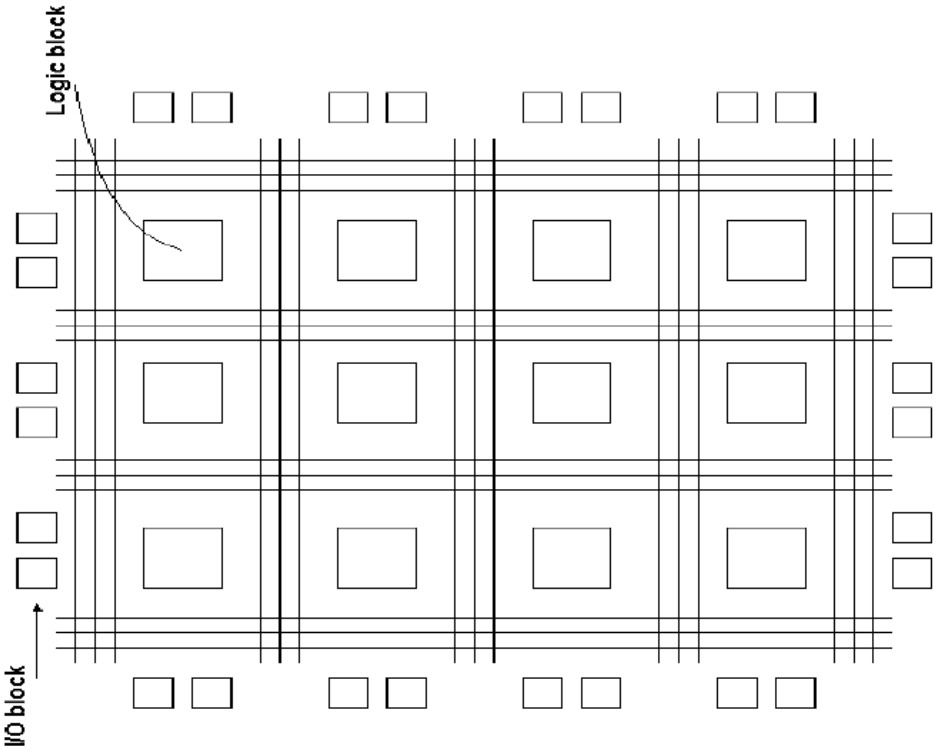
ΚΕΦΑΛΑΙΟ 2^{ο} – ΛΟΓΙΚΕΣ ΔΙΑΤΑΞΗΣ ΠΥΛΩΝ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΕΣ ΣΤΟ ΠΕΔΙΟ (FPGA)

2.1 Γενικά

Οι λογικές διατάξεις πινάκων πυλών προγραμματιζόμενων στο πεδίο (FPGAs ή Field Programmable Gate Arrays), είναι διατάξεις προγραμματιζόμενης λογικής που υποστηρίζουν την υλοποίηση μεγάλων κυκλωμάτων.

Τα σύγχρονα FPGAs αποτελούνται από πολλές λογικές πύλες και Block μνήμης (RAM) με σκοπό να υλοποιούν πολύπλοκες σύνθετες εφαρμογές. Η πυκνότητά τους σε πύλες, είναι από μερικές χιλιάδες ως μερικά εκατομμύρια, ανάλογα με την οικογένεια που ανήκει το FPGA.

Η γενική δομή μιας διάταξης FPGA φαίνεται στο σχήμα 2.1 . Περιέχει τους ακροδέκτες εισόδου / εξόδου (I / O blocks), τις λογικές βαθμίδες (logic blocks) και τις γραμμές διασύνδεσης.



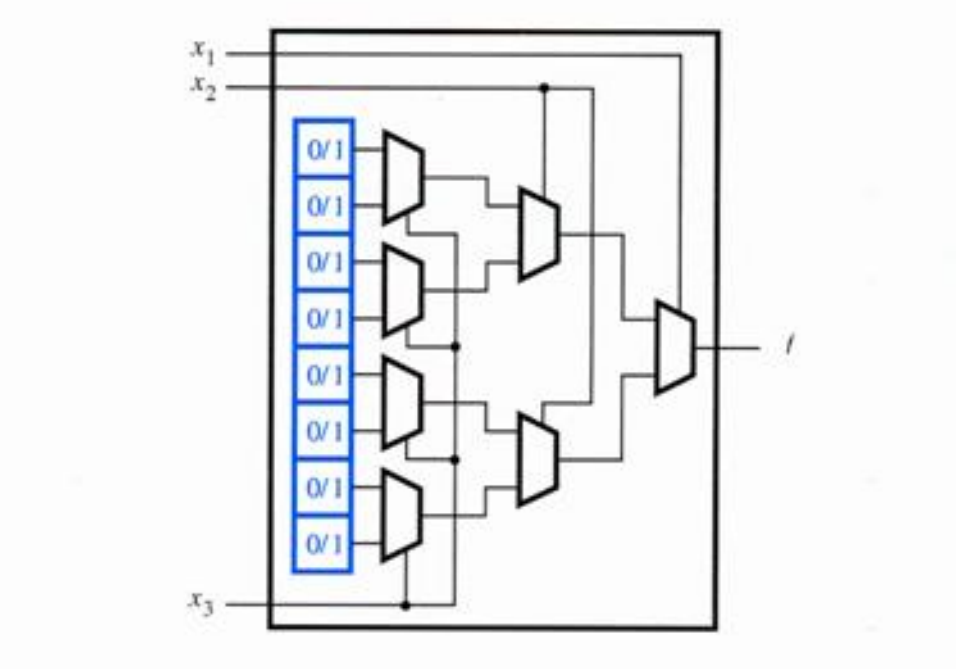
Σχήμα 2.1 Γενική δομή ενός FPGA

Οι λογικές βαθμίδες οργανώνονται με τη μορφή διάστασης σειράς. Κάθε λογική βαθμίδα ενός FPGA έχει τυπικά ένα μικρό αριθμό εισόδων και μια έξοδο. Οι γραμμές διασύνδεσης οργανώνονται ως οριζόντια και κατακόρυφα κανάλια δρομολόγησης (routing channels) ανάμεσα στις γραμμές και τις στήλες των λογικών βαθμίδων. Τα κανάλια αυτά εμπεριέχουν καλώδια και προγραμματιζόμενους διακόπτες που επιτρέπουν τις λογικές βαθμίδες να διασυνδέονται με πολλούς τρόπους. Οι διακόπτες που βρίσκονται δίπλα στις λογικές βαθμίδες συνδέουν τους ακροδέκτες εισόδου / εξόδου των λογικών βαθμίδων με τα καλώδια διασύνδεσης. Τα καλώδια διασύνδεσης συνδέονται μεταξύ τους και υπάρχουν προγραμματιζόμενες συνδέσεις ανάμεσα στις βαθμίδες εισόδου / εξόδου και τα καλώδια διασύνδεσης.

Στα περισσότερα FPGAs, οι λογικές βαθμίδες περιέχουν και στοιχεία μνήμης τα οποία μπορεί να είναι απλά flip – flops ή ακόμα και πιο ολοκληρωμένα μπλόκ μνήμης.

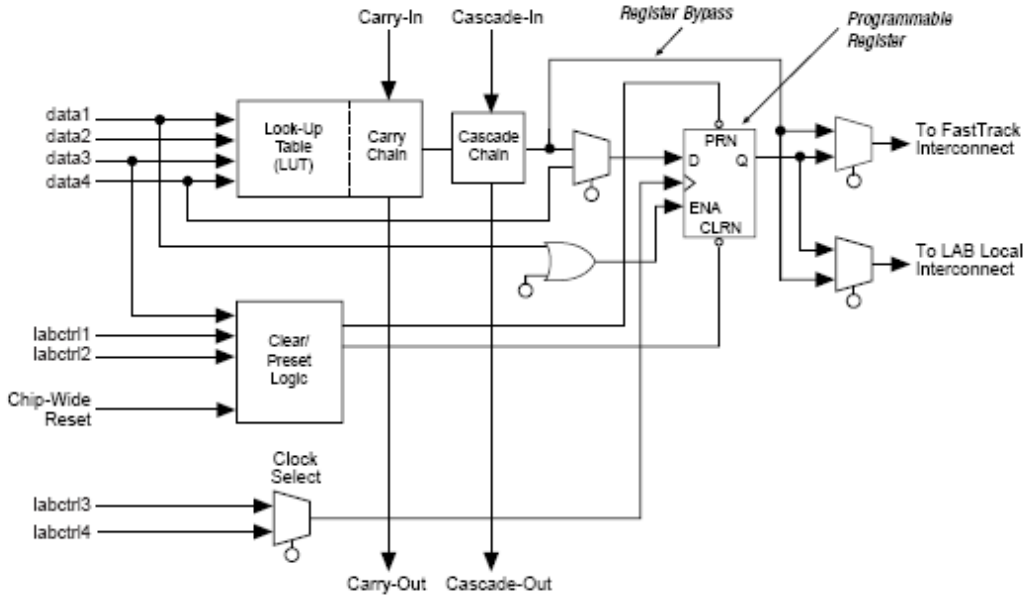
2.2 Λογικά στοιχεία (Logic Elements)

Μια διάταξη FGPA είναι κατασκευασμένη ως επί τω πλείστον από λογικά στοιχεία (Logic Elements ή LE). Κάθε ένα λογικό στοιχείο περιέχει έναν πίνακα αναφοράς (LUT ή Look-up Table), συνήθως τεσσάρων ή πέντε εισόδων, όπου χρησιμοποιείται για την υλοποίηση μιας συνάρτησης. Είναι ένας μονοδιάστατος πίνακας μνήμης όπου οι γραμμές διευθύνσεων της μνήμης είναι οι είσοδοι του λογικού στοιχείου και η έξοδος ενός bit της μνήμης είναι η έξοδος του πίνακα αναφοράς. Κάθε πίνακας αναφοράς περιέχει κυψέλες αποθήκευσης (storage cells) όπου μπορούν να κρατήσουν μια λογική τιμή, 0 ή 1, η οποία μεταφέρεται στην έξοδο της κυψέλης. Παρακάτω, στο σχήμα 2.2 φαίνεται ένας πίνακας αναφοράς τριών εισόδων X1, X2, X3 και μιας εξόδου F που υλοποιεί συναρτήσεις τριών μεταβλητών. Οι κυψέλες αποθήκευσης που περιέχει ο LUT είναι οχτώ. Οι τρεις είσοδοι του LUT χρησιμοποιούνται ως διακόπτες επιλογής των εφτά πολυπλεκτών για την επιλογή στην έξοδο μιας από τις οκτώ αποθηκευμένες τιμές.



Σχήμα 2.2 LUT τριών εισόδων

Η έξοδος από το LUT συνδέεται και στο flip – flop μέσω του ειδικού κυκλώματος διαδοχής, αλλά και στην έξοδο του LE. Τα σήματα clock, reset και clear χρησιμοποιούνται ως είσοδοι στο flip – flop του κάθε LE. Το καθένα LEs έχει δύο εξόδους, η μία κατευθύνεται στον Lab local και η άλλη στο Fast Track. Στο σχήμα 2.3 φαίνεται αναλυτικά η αρχιτεκτονική ενός λογικού στοιχείου (LE).



Σχήμα 2.3 Γενική αρχιτεκτονική ενός λογικού στοιχείου

Δε θα πρέπει να παραληφθεί ότι κάθε στοιχείο λειτουργεί σε τέσσερις διαφορετικές καταστάσεις. Αυτές είναι οι εξής:

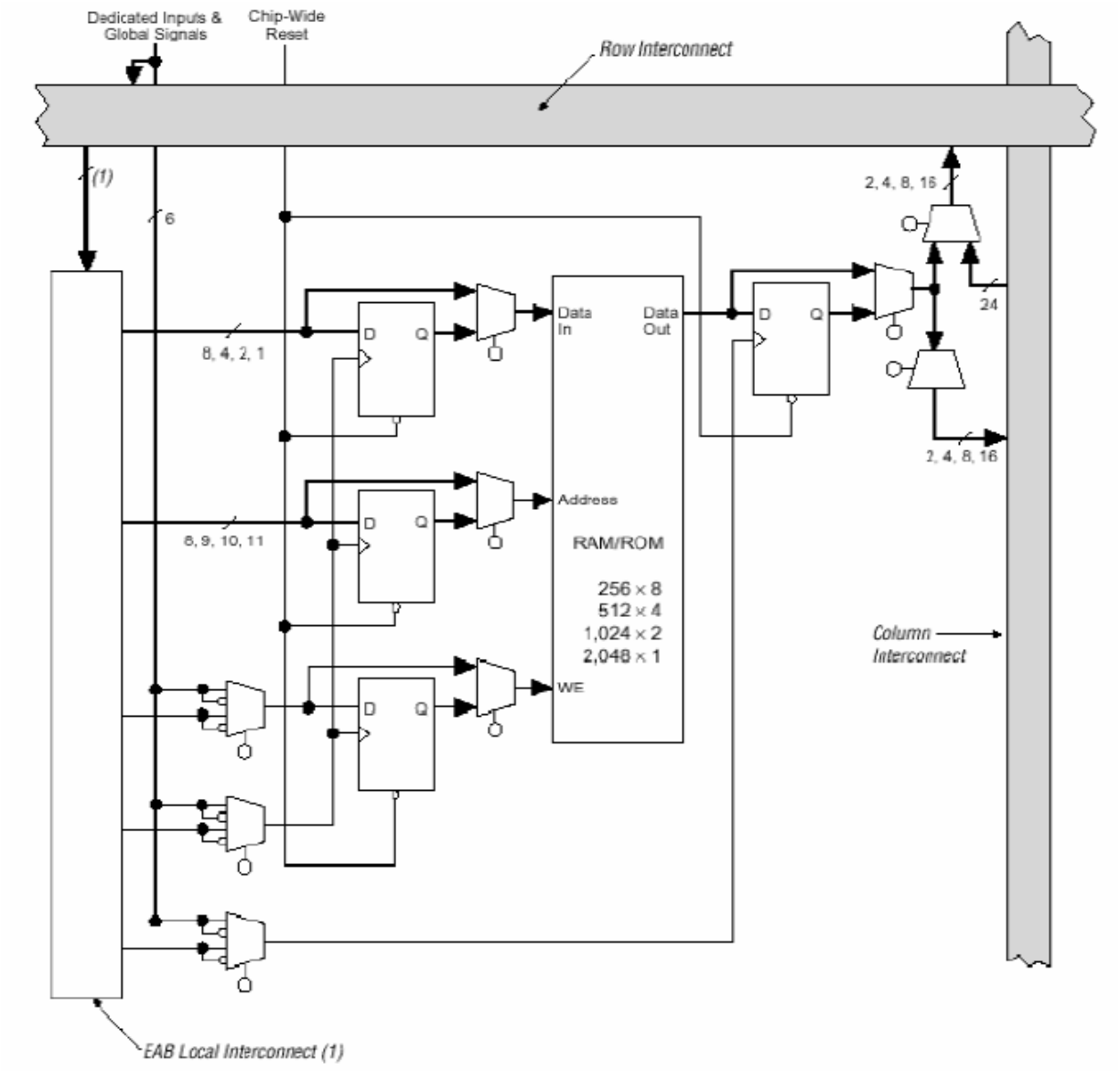
- **Normal Mode:** Χρησιμοποιείται για την υλοποίηση συναρτήσεων λογικής γενικού σκοπού και συναρτήσεων αποκωδικοποίησης με πολλές εισόδους που μπορούν να εκμεταλλευτούν την αλυσίδα διαδοχής (Cascade chain).
- **Arithmetic Mode:** Χρησιμοποιείται για την υλοποίηση αθροιστών, συσσωρευτών και συγκριτών.
- **Up – Down Counter:** Δυνατότητα μέτρησης πάνω – κάτω με δυνατότητα σύγχρονου ελέγχου και επιλογές φόρτωσης δεδομένων.
- **Clearable Counter Mode:** Χρησιμοποιείται σαν up – down counter με επιπλέον δυνατότητα σύγχρονου μηδενισμού.

2.3 Βαθμίδες ενσωματωμένων διατάξεων

Εκτός από τις βαθμίδες των LABs υπάρχουν και οι βαθμίδες ενσωματωμένων διατάξεων (Embedded Array Blocks – EAB). Κάθε EAB περιέχει από 100 ως 600 πύλες οι οποίες χρησιμοποιούνται για την εκτέλεση πολύπλοκων λειτουργιών. Όταν τεθεί σε λειτουργία μνήμης, κάθε EAB παραχωρεί 2048 bits SRAM μνήμης, όπου μπορούν να χρησιμοποιηθούν για μνήμες ROM, RAM, Dual – Port ROM ή καταχωρητές First In First Out (FIFO). Όταν χρησιμοποιείται ως μνήμη RAM μπορεί να διαμορφωθεί στα εξής μεγέθη: 256 * 8, 512 * 4, 1042 * 2 και 2048 * 1. Σε περίπτωση που χρειαστεί μεγαλύτερη μνήμη από 2048 bits θα χρησιμοποιηθούν δύο βαθμίδες EAB. Επίσης, τα EAB μπορούν να διαμορφωθούν για να υλοποιήσουν λογικές συναρτήσεις, όπως πολλαπλασιαστές, μικροελεγκτές και μονάδες ψηφιακού σήματος.

Στο σχήμα 2.5 παρατηρούμε την εσωτερική δομή ενός EAB. Οι είσοδοι των δεδομένων και των διευθύνσεων στη μνήμη παρέχονται μέσω του τοπικού διαύλου.

Αυτές οι είσοδοι καθώς και μία ακόμη είσοδος που ενεργοποιεί την εγγραφή της βαθμίδας μνήμης αποθηκεύονται σε ένα flip flop, ο αριθμός των οποίων εξαρτάται από το μέγεθος της μνήμης. Οι έξοδοι του EAB οδηγούνται στις οριζόντιες και κάθετες διασυνδέσεις που περιβάλλουν τη βαθμίδα και μπορούν να καταχωρηθούν και αυτές σε ένα flip flop. Στο σχήμα 2.5 φαίνεται η δομή ενός EAB.

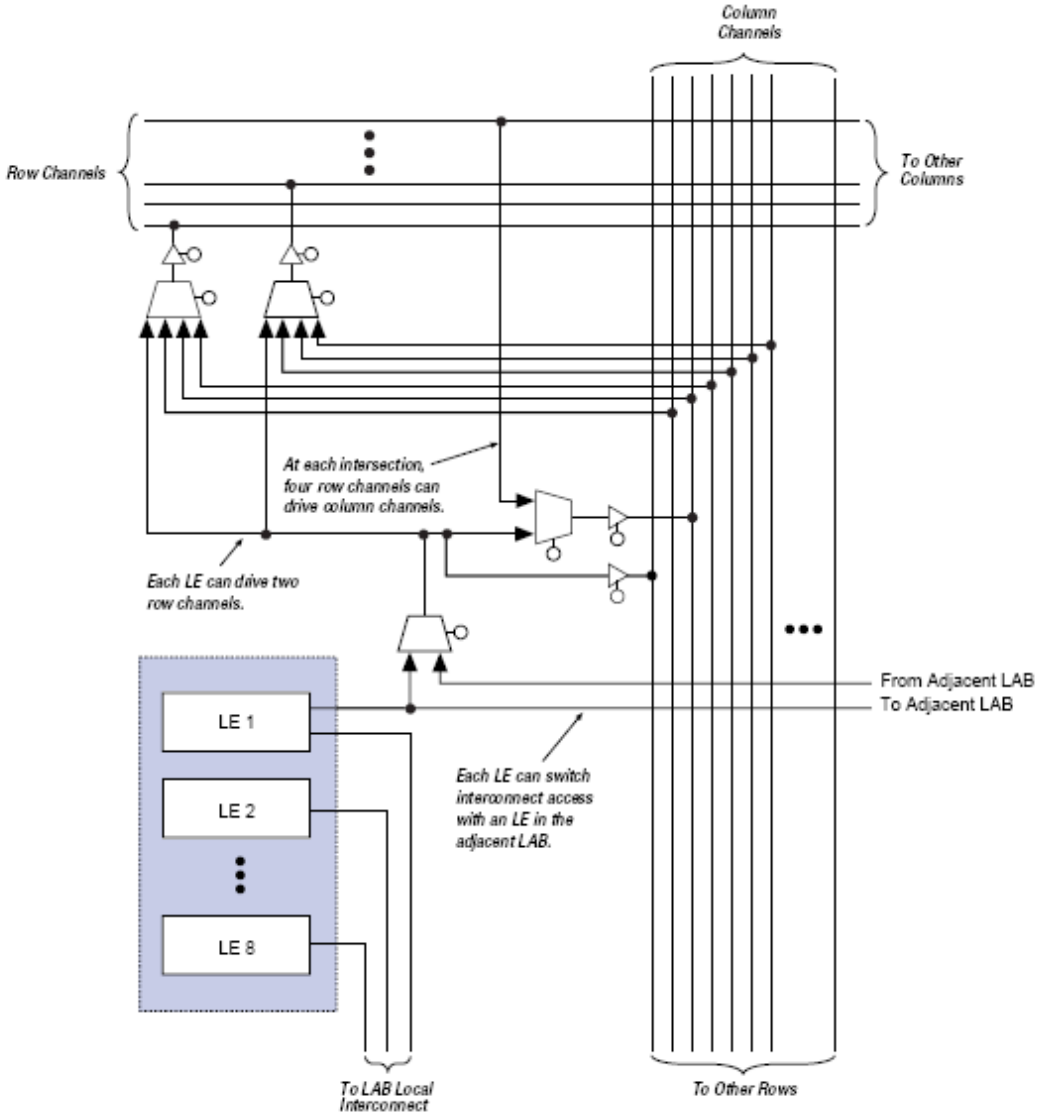


Σχήμα 2.5 Αρχιτεκτονική ενός EAB

2.4 Γραμμές διασύνδεσης (Fast Track Interconnect)

Οι γραμμές διασύνδεσης οργανώνονται ως οριζόντια και κατακόρυφα κανάλια δρομολόγησης που εκτείνονται σε όλο το μήκος και πλάτος του FPGA και συνδέουν μεταξύ τους τα EAB, τα LAB και τους διακόπτες εισόδου / εξόδου. Κάθε γραμμή LABs εξυπηρετείται από μια γραμμή του Fast Track Interconnect. Τα κάθετα κανάλια του διαδρόμου διασυνδέσεων (στήλες), ενώνουν τις γραμμές και οδηγούν σε I / O διακόπτες.

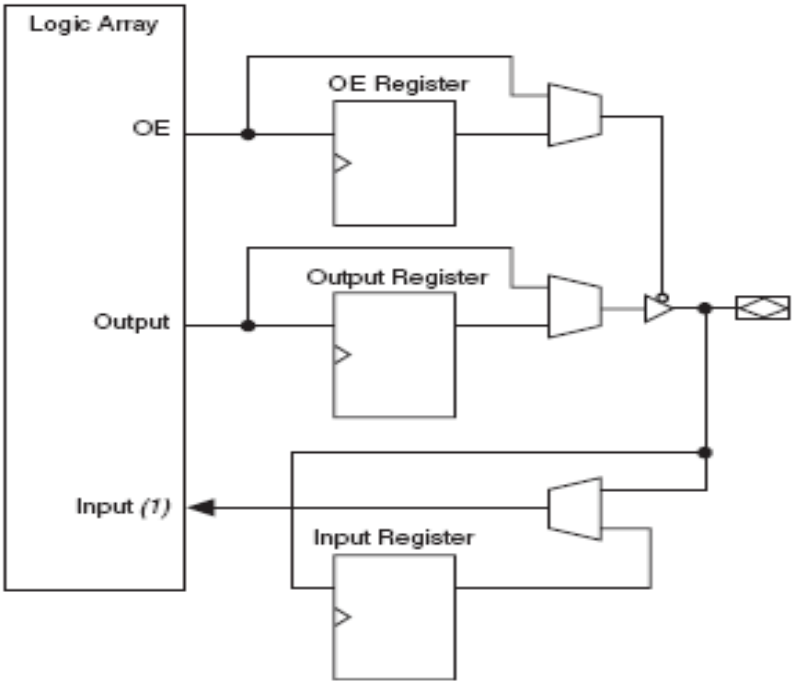
Στο σχήμα 2.6 φαίνονται οι γραμμές διασύνδεσης ενός LAB με το διάδρομο Fast Track Interconnect.



Σχήμα 2.6 Διάδρομος διασυνδέσεων Fast Track Interconnect

2.5 Ακροδέκτες εισόδου / εξόδου (I / O Block)

Οι ακροδέκτες εισόδου / εξόδου καταλαμβάνουν περιφερειακά το ολοκληρωμένο κύκλωμα. Οι ακροδέκτες μπορούν να χρησιμοποιηθούν ως είσοδοι ή έξοδοι του κυκλώματος και μπορούν να καθοριστούν να λειτουργούν και αμφίδρομα. Παρακάτω, στο σχήμα 2.7 φαίνεται η εσωτερική δομή των ακροδεκτών εισόδου / εξόδου.



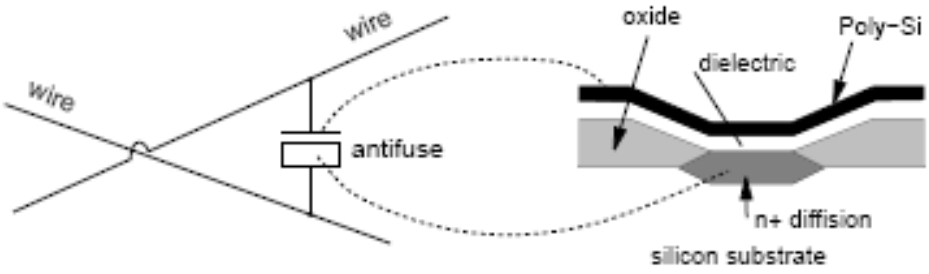
Σχήμα 2.7 Εσωτερική δομή ακροδεκτών εισόδου / εξόδου

Στην περίπτωση που ο ακροδέκτης λειτουργεί σαν είσοδος (input) και συνδέεται με τις γραμμές διασύνδεσης (οριζόντιες και κάθετες) λαμβάνει το σήμα το οποίο διοχετεύεται προς όλα τα λογικά στοιχεία που είναι συνδεδεμένα με εκείνες τις γραμμές που έρχεται σε επαφή με τη βαθμίδα I / O. Στη δεύτερη περίπτωση, δηλαδή όταν ο ακροδέκτης λειτουργεί σαν έξοδος (output), τότε το σήμα οδηγείται από έναν πολυπλέκτη που έχει εισόδους έναν διαφορετικό αριθμό από κανάλια διασύνδεσης (οριζόντιες και κάθετες γραμμές).

2.6 Προγραμματισμός διακοπτών στα FPGAs

Στα FPGAs ο προγραμματισμός των διακοπτών στηρίζεται είτε στην τεχνολογία των αντιασφαλειών (antifuses technology), είτε στην τεχνολογία της στατικής μνήμης (SRAM).

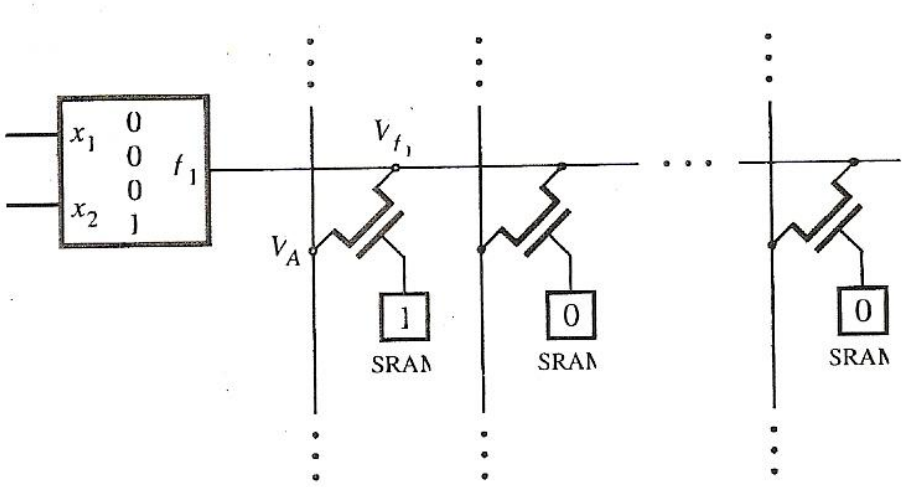
Οι αντιασφάλειες είναι ανοιχτά κυκλώματα τα οποία δεν είναι προγραμματισμένα και όταν αυτό συμβεί έχουν χαμηλή αντίσταση. Όπως φαίνεται και στο σχήμα 2.8 η αντιασφάλεια είναι τοποθετημένη μεταξύ δύο καλωδίων διασύνδεσης και αποτελείται από τρία στρώματα. Δύο αγώγιμα, όπου ανάμεσά τους βρίσκεται και ένα τρίτο μονωτικό στρώμα. Στην περίπτωση που το μονωτικό στρώμα δεν είναι προγραμματισμένο, ο μονωτής μετατρέπεται σε έναν σύνδεσμο χαμηλής αντίστασης μεταξύ των άλλων δύο στρωμάτων. Παρακάτω, στο σχήμα 2.8 ακολουθεί η γενική δομή μιας αντιασφάλειας.



Σχήμα 2.8 Γενική δομή μιας αντιασφάλειας

Η τεχνολογία της στατικής μνήμης SRAM είναι περισσότερο διαδεδομένη στα FPGAs από τις αντιασφάλειες καθώς επιτρέπει τον επαναπρογραμματισμό. Το μειονέκτημά της είναι πως με την διακοπή της τροφοδοσίας χάνονται τα δεδομένα και χρειάζεται ξανά προγραμματισμό.

Στο παρακάτω σχήμα 2.9 φαίνεται ότι ο ακροδέκτης μνήμης μπορεί να αποθηκευτεί με τη λογική τιμή 0 ή 1. Αν έχει την τιμή 0, τότε το στοιχείο μνήμης είναι απενεργοποιημένο. Αν έχει την τιμή 1, τότε είναι ενεργό και υπάρχει σύνδεση μεταξύ του οριζόντιου και κάθετου καλωδίου.



Σχήμα 2.9 Διακόπτες SRAM

2.7 Προγραμματισμός FPGA

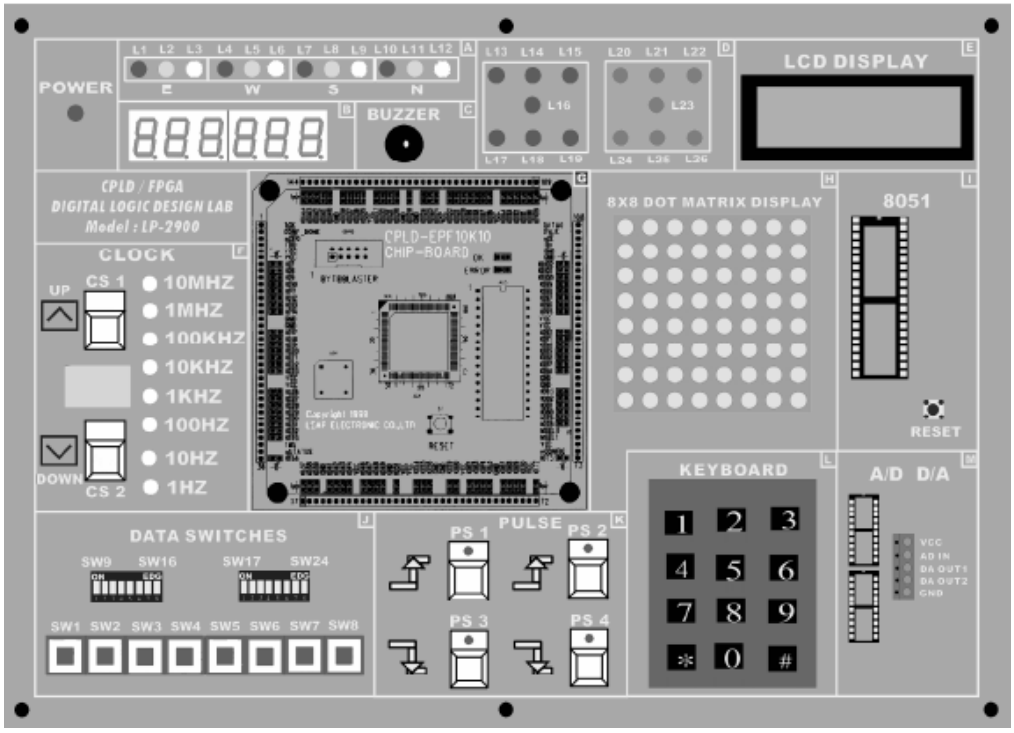
Ο προγραμματισμός ενός FPGA γίνεται όταν αυτό βρίσκεται πάνω στην PCB πλακέτα και εφαρμόζεται η μέθοδος ISP – In System Programming. Με τη συγκεκριμένη μέθοδο το κύκλωμα δεν χρειάζεται να αφαιρεθεί από την πλακέτα και να τοποθετηθεί σε άλλη συσκευή προγραμματισμού. Η πλακέτα διαθέτει έναν ειδικό συνδετήρα που ονομάζεται JTAG (Joint Test Action Group), όπου μέσω αυτού ο υπολογιστής ελέγχει τους ακροδέκτες ολοκληρωμένου κυκλώματος. Ο συνδετήρας συνδέεται μέσω ενός καλωδίου με την USB θύρα του ηλεκτρονικού υπολογιστή και με τον τρόπο αυτό μεταφέρονται τα εκτελέσιμα αρχεία του χρήστη που έχουν δημιουργηθεί με τη χρήση κάποιου λογισμικού σχεδίασης (στην περίπτωσή μας με το Quartus II). Τα αρχεία μεταφέρονται στο FPGA και έτσι ολοκληρώνεται ο προγραμματισμός.

ΚΕΦΑΛΑΙΟ 3^{ο} – ΑΝΑΠΤΥΞΙΑΚΟ ΚΥΚΛΩΜΑ LP – 2900

3.1 Γενικά

Η αναπτυξιακή πλακέτα που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής μας είναι η LP – 2900 της εταιρείας Leap Electronic Co. Το αναπτυξιακό κύκλωμα LP – 2900 βασίζεται στο ολοκληρωμένο FPGA EPF10KTC144 – 4 και ανήκει στην οικογένεια Altera. Διαθέτει 144 pins εισόδου / εξόδου από τα οποία τα 102 είναι ελεύθερα για χρήση και αποτελείται από 576 λογικά στοιχεία (logic elements) και 61444 bits μνήμης.

Στο σχήμα 3.1 που ακολουθεί φαίνεται το αναπτυξιακό κύκλωμα LP – 2900 το οποίο χωρίζεται στα εξής μέρη: στο μέρος που περιέχεται το FPGA, στο μέρος της τροφοδοσίας, στις συσκευές εισόδου / εξόδου και στη θύρα επικοινωνίας με τον υπολογιστή.



Σχήμα 3.1 Αναπτυξιακό κύκλωμα LP – 2900

Το αναπτυξιακό κύκλωμα LP – 2900 είναι εφοδιασμένο με μια σειρά περιφερειακών, όπως:

- 4 σετ από κόκκινα, κίτρινα και πράσινα leds κοινής καθόδου
- ενδείκτες επτά τομέων (7 segment displays) κοινής ανόδου
- 1 buzzer
- 2 ηλεκτρονικά ζάρια (dices) κοινής καθόδου
- 8 διακόπτες τύπου push button
- 2 X 8 διακόπτες τύπου dip switches
- 4 διακόπτες παλμών
- 1 8 X 8 dot matrix display
- 1 πληκτρολόγιο 4 X 3
- 1 οθόνη LCD
- 1 μετατροπέα σήματος από αναλογικό σε ψηφιακό (A / D και D / A)
- 1 ολοκληρωμένο κύκλωμα 8051 (microprocessor)

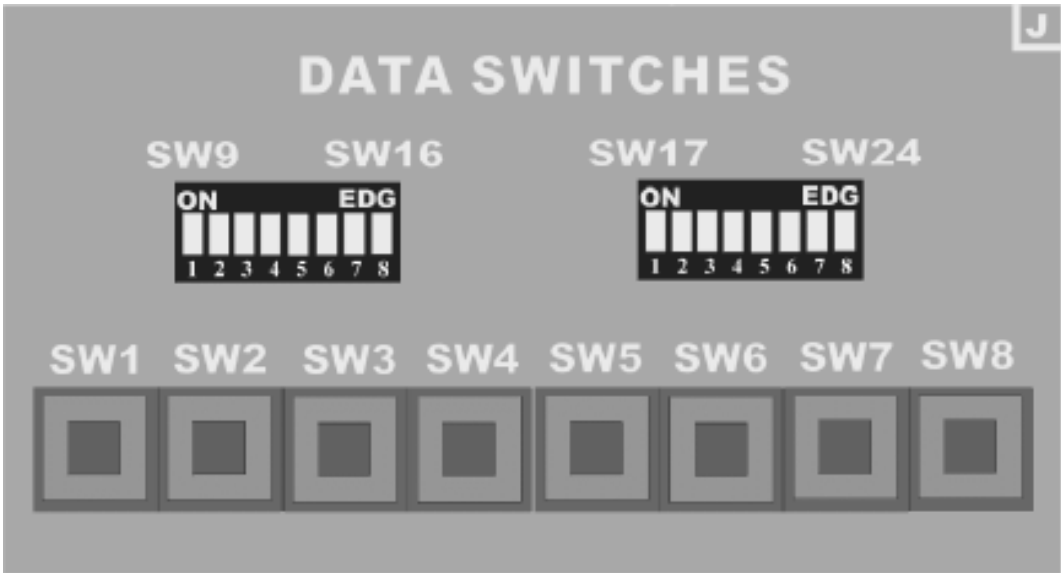
Η σύνδεση της αναπτυξιακής πλακέτας με τον υπολογιστή γίνεται μέσω ενός καλωδίου όπου το ένα άκρο του το συνδέουμε στη θύρα του αναπτυξιακού και το άλλο στην παράλληλη θύρα του υπολογιστή. Τίθεται αναγκαία η εγκατάσταση των οδηγών BYTE BLASTER της Altera για να εγκατασταθεί στο Quartus η κατάλληλη θύρα προγραμματισμού (LPT1).

Στο παρακάτω σχήμα 3.2 φαίνεται το διάγραμμα βαθμίδων του αναπτυξιακού κυκλώματος LP – 2900.

Σχήμα 3.2 Διάγραμμα βαθμίδων του LP – 2900

Το αναπτυξιακό κύκλωμα αν και διαθέτει πολλές συσκευές εισόδου / εξόδου θα αναφερθούμε μόνο σε αυτές που χρησιμοποιήσαμε στην εφαρμογή μας. Είναι οι ακόλουθες:

- **Διακόπτες Δεδομένων**

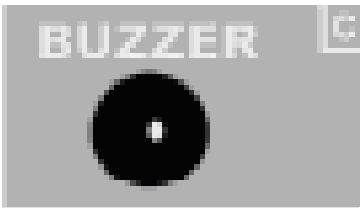


SW1	SW2	SW3	SW4	SW5	SW6	SW7	SW8
Pin 47	Pin 48	Pin 49	Pin 51	Pin 59	Pin 60	Pin 62	Pin 63

SW9	SW10	SW11	SW12	SW13	SW14	SW15	SW16
Pin **64**	Pin 65	Pin 67	Pin 68	Pin 69	Pin 70	Pin 72	Pin 73
SW17	**SW18**	**SW19**	**SW20**	**SW21**	**SW22**	**SW23**	**SW24**
Pin **78**	Pin 679	Pin 80	Pin 81	Pin 82	Pin 83	Pin 86	Pin 87

Οι διακόπτες SW1 ως SW8 είναι διακόπτες τύπου push – button, ενώ οι διακόπτες SW9 ως SW24 αντιστοιχούν σε διακόπτες τύπου dip switches.

- **Βομβητής (Buzzer)**



SP1
Pin 46

ΚΕΦΑΛΑΙΟ 4^{ο} – ΜΟΥΣΙΚΟΣ ΕΠΕΞΕΡΓΑΣΤΗΣ

4.1 Εισαγωγή

Η σχεδίαση του μουσικού επεξεργαστή για την παραγωγή ενός μουσικού σήματος ολοκληρώθηκε με τη γλώσσα περιγραφής υλικού VHDL και υλοποιήθηκε σε FPGA μέσω της αναπτυξιακής πλακέτας LP-2900 της εταιρείας Leap Electronic Co. Η αναπτυξιακή πλακέτα LP – 2900 βασίζεται στη διάταξη FPGA EPF 10K10TC144-4.

Για την ολοκλήρωση της εφαρμογής ήταν σημαντικό να καθοριστεί η διάρκεια και το επιθυμητό ηχητικό αποτέλεσμα της κάθε νότας. Η διάρκεια της κάθε νότας επιτεύχθηκε με τη σχεδίαση του κυκλώματος Count_16Comp, όπου αποτελείται από έναν απαριθμητή 16 καταστάσεων και έναν συγκριτή. Η αναλυτική περιγραφή του αρχείου Count_16Comp γίνεται στην ενότητα 4.3. Ο ήχος της κάθε νότας παράγεται από έναν κρύσταλλο 10 MHZ που βρίσκεται στην αναπτυξιακή πλακέτα LP – 2900. Για την επίτευξη του επιθυμητού ηχητικού αποτελέσματος της κάθε νότας ήταν αναγκαίο τα 10MHZ να υποδιαιρεθούν με τη σωστή συχνότητα της κάθε νότας. Η αναλυτική περιγραφή του κυκλώματος γίνεται στην ενότητα 4.4.

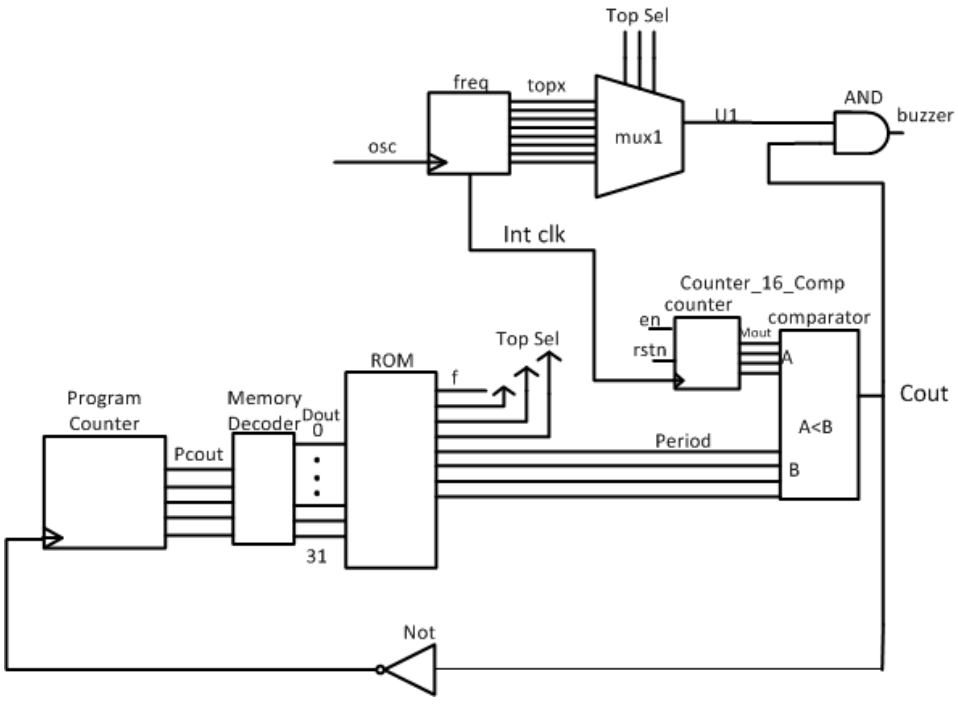
Τέλος, ήταν σημαντικό η δημιουργία μιας μνήμης ROM, όπου μέσα της περιέχονται χαρτογραφημένες νότες που αποτελούν το τραγούδι. Δηλαδή, είναι το κύριο και σημαντικότερο κύκλωμα του μουσικού επεξεργαστή όπου αναπαράγεται το τραγούδι. Η αναλυτική περιγραφή του κυκλώματος γίνεται στην ενότητα 4.4 .

4.2 Περιγραφή λειτουργίας του μουσικού επεξεργαστή

Σε κάθε κύκλο του ρολογιού εντολών εξάγεται το περιεχόμενο μιας νέας θέσης της μνήμης ROM που περιέχει το πρόγραμμα (μουσικό κομμάτι). Τα τέσσερα πιο σημαντικά Bits της ROM (b7-b4) περιέχουν τη νότα σε κωδικοποιημένη μορφή ενώ τα τέσσερα λιγότερο σημαντικά περιέχουν σε κωδικοποιημένη μορφή την διάρκεια της νότας. Ο κώδικας για τη νότα εμφανίζεται στις εισόδους επιλογής του

πολυπλέκτη mux1 ο οποίος και επιλέγει μια από τις 8 διαθέσιμες νότες. Η περίοδος εφαρμόζεται στην είσοδο B του συγκριτή (comparator) και συγκρίνεται με τη δυαδική ακολουθία που παράγει ο απαριθμητής (counter). Όσο η έξοδος είναι μικρότερη από την περίοδο, η νότα θα συνεχίζει να ακούγεται στο buzzer της αναπτυξιακής διάταξης. Αυτό το πετυχαίνουμε μέσω μιας πύλης AND, όπου επίσης εφαρμόζεται η έξοδος του πολυπλέκτη. Το άλλο σημαντικό κομμάτι της σχεδίασης περιλαμβάνει τη βαθμίδα freq. Εκεί γίνονται όλες η υποδιαιρέσεις συχνότητας ξεκινώντας από την αρχική συχνότητα του εξωτερικού ταλαντωτή (OSC) που είναι 10 MHz. Από τη βαθμίδα αυτή παράγεται το σήμα int_clk που οδηγεί τον counter της βαθμίδας counter_16_comp με συχνότητα περίπου 10 Hz. Έτσι, η συνολική διάρκεια μιας νότας η οποία περιλαμβάνει όλη τη δυαδική ακολουθία των 16 καταστάσεων του counter μπορεί να είναι περίπου 1,6sec.

Κλείνοντας την περιγραφή λειτουργίας του μουσικού επεξεργαστή αναφέρουμε ότι εκτός από τα δύο κύρια κυκλώματα για την προγραμματιστική υλοποίηση του μουσικού επεξεργαστή, δημιουργήθηκε κώδικας VHDL και για κάποια άλλα αρχεία (Packets), τα οποία περιέχουν τις δηλώσεις των μεταβλητών και θα αναφερθούν στο κεφαλαίο του περιέχει τον κώδικα. Παρακάτω στο σχήμα 4.1 φαίνεται το διάγραμμα βαθμίδων της ανώτερης ιεραρχικά οντότητας του μουσικού επεξεργαστή (Top Level).



Σχήμα 4.1 Διάγραμμα βαθμίδων της ανώτερης ιεραρχικά οντότητας του μουσικού επεξεργαστή

Παρατηρώντας το σχήμα 4.1 κατανοούμε πως το διάγραμμα βαθμίδων της ανώτερης ιεραρχικής οντότητας του μουσικού επεξεργαστή αποτελείται από τα εξής επιμέρους κυκλώματα:

- Έναν μετρητή με όνομα Freq
- Έναν πολυπλέκτη με όνομα Mux
- Μια πύλη AND
- Ένα κύκλωμα ενός απαριθμητή - συγκριτή με όνομα Counter_16Comp
- Μια μνήμη ROM
- Έναν αποκωδικοποιητή με όνομα Memory Decoder
- Και έναν μετρητή με όνομα Program Counter

Στις επόμενες ενότητες ακολουθεί ξεχωριστά η λειτουργία του κάθε κυκλώματος.

4.3 Κύκλωμα μετρητή (Freq)

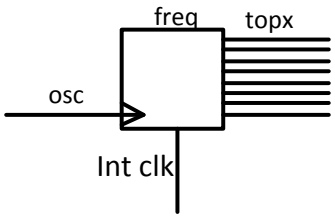
Το κύκλωμα του μετρητή όπως φαίνεται στο σχήμα 4.3 αποτελείται από μία είσοδο osc του ενός Bit, μία έξοδο topx των 8 Bits και μία έξοδο ενός εσωτερικού ρολογιού Int clk του ενός Bit.

Στην εφαρμογή χρησιμοποιηθήκαν οι οχτώ νότες του μουσικού πενταγράμμου, από το Ντο της $4^{ης}$ Οκτάβας έως το Ντο της $5^{ης}$ Οκτάβας. Στον παρακάτω πίνακα, σχήμα 4.2 φαίνονται οι μουσικές νότες με τη συχνότητά τους.

ΟΚΤΑΒΑ	**ΝΤΟ**	**ΡΕ**	**ΜΙ**	**ΦΑ**	**ΣΟΛ**	**ΛΑ**	**ΣΙ**
$4^{η}$	523,2511Hz	587,3295Hz	659,2551Hz	698,4565Hz	783,9909Hz	880Hz	987,7666Hz
$5^{η}$	1046,502Hz						

Σχήμα 4.2 Πίνακας συχνοτήτων $4^{ης}$ και $5^{ης}$ Οκτάβας

Το κύκλωμα του μετρητή (freq) παίρνει σαν είσοδο ένα σήμα από τον εξωτερικό ταλαντωτή Osc, ο οποίος έχει 10 MHZ. Υποδιαιρέθηκαν τα 10MHZ του Osc τόσες φόρες όσες ήταν απαραίτητες έτσι ώστε η έξοδος topx να βγάζει τις συχνότητες της κάθε μουσικής νότας. Η έξοδος Int clk περνάει σαν ένα εσωτερικό ρολόι στο κύκλωμα Count_16Comp το οποίο θα αναλυθεί παρακάτω μιας και είναι υπεύθυνο για την περίοδο της κάθε νότας. Είναι σημαντικό να αναφερθεί ότι η είσοδος δεν δίνεται από τον χρήστη.

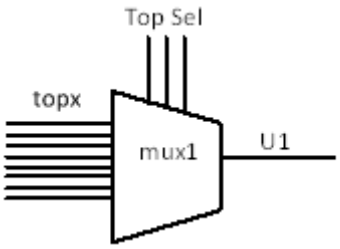


Σχήμα 4.3 Μετρητής freq

4.4 Πολυπλέκτης 8:1 (Mux1)

Ο πολυπλέκτης 8 προς 1 αποτελείται από δύο εισόδους, την είσοδο Topx των 8 Bits, την είσοδο Topsel των 3 Bits και μία έξοδο την U1 του ενός Bit. Σκοπός του είναι ποια από τις οχτώ νότες θα βγαίνει κάθε φορά στην έξοδο του κυκλώματος.

Η είσοδος Topx περιέχει τις νότες που δημιουργήθηκαν από τον μετρητή freq. Η είσοδος Topsel είναι οι γραμμές επιλογής του πολυπλέκτη, όπου καθορίζουν ποια από τις οχτώ μουσικές νότες που παράγει ο μετρητής freq θα βγει στην έξοδο U1. Η έξοδος U1 συνδέεται με μια πύλη AND, η οποία αναλύεται στην ενότητα 4.6 . Παρακάτω στο σχήμα 4.4 φαίνεται το κύκλωμα του πολυπλέκτη (mux1) 8 προς 1.



Σχήμα 4.4 Πολυπλέκτης Mux1

4.5 Απαριθμητής – Συγκριτής (Counter_16Comp)

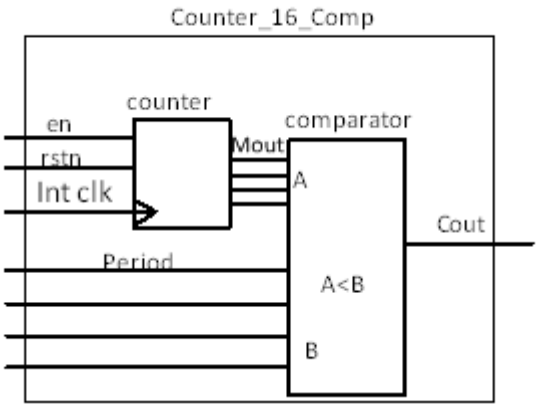
Το κύκλωμα αυτό αποτελείται από έναν απαριθμητή 16 καταστάσεων με όνομα Counter και έναν συγκριτή με όνομα Comparator, δηλαδή είναι ένα ενιαίο κύκλωμα, όπου ονομάζεται Counter_16Comp.

Ο απαριθμητής (Counter) αποτελείται από τρεις εισόδους, την en του ενός Bit, την είσοδο rstn του ενός Bit, την είσοδο Int clk του ενός Bit με συχνότητα 10MHZ (όπως φαίνεται στο σχήμα 4.1) και μια έξοδο Mout των τεσσάρων Bits. Οι είσοδοι en και rstn πρέπει να είναι σε λογική κατάσταση 1 (en=1 και rstn=1) για να μπορεί ο απαριθμητής (counter) να απαριθμήσει όλες τις καταστάσεις. Σε διαφορετική περίπτωση (π.χ. en=0 και rstn=1) η απαρίθμηση επανέρχεται στην αρχική κατάσταση. Οι δύο είσοδοι δίνονται αρχικά από το χρήστη με το πάτημα δύο εξωτερικών διακοπτών. Είναι σημαντικό να αναφερθεί πως η συνολική διάρκεια μιας νότας η οποία περιλαμβάνει όλη τη δυαδική ακολουθία των 16 καταστάσεων του απαριθμητή (Counter) μπορεί να διαρκεί περίπου 1,6 δευτερόλεπτα.

Ο συγκριτής (Comparator) αποτελείται από δύο εισόδους, την A και την B των τεσσάρων Bits η κάθε μία και από μία έξοδο Cout του ενός Bit. Η πρώτη είσοδος, η A, προέρχεται από την έξοδο του απαριθμητή (Counter). Η δεύτερη είσοδος, η B, προέρχεται από το σήμα Period, το οποίο περιέχεται στο Op Code κάθε εντολής που διαβάζεται από τη ROM. Όταν η είσοδος A είναι μικρότερη από την είσοδο B (A < B) η έξοδος του συγκριτή γίνεται ένα. Έτσι, η πύλη AND που δέχεται για είσοδο την έξοδο Cout του συγκριτή βγάζει στην έξοδο τη λογική τιμή ένα. Στην αντίθετη περίπτωση, δηλαδή όταν είσοδος A είναι μεγαλύτερη από την είσοδο B (A > B) η έξοδος του συγκριτή γίνεται μηδέν. Τότε η έξοδος Cout οδηγείται σε μια πύλη NOT, η οποία καταφτάνει στον Program Counter (σχήμα 4.1) και δεν εκτελείται η επόμενη νότα. Με αυτόν τον τρόπο το σήμα που οδηγείται στο Buzzer δε διαρκεί περισσότερο από αυτό που προβλέπεται στο σήμα Period.

Το κύκλωμα Program_Counter είναι απαραίτητο να λάβει θετικό μέτωπο παλμού, έτσι ώστε να προχωρήσει στην επόμενη εντολή-νότα. Αυτό συμβαίνει όταν έχει εξαντληθεί πλέον η διάρκεια της προηγούμενης νότας, οπότε ο συγκριτής Counter_16_Comp παράγει στην έξοδο μηδέν. Μέσω μια πύλης NOT η μετάβαση

στο μηδέν γίνεται μετάβαση στο λογικό ένα, οπότε ο program counter εκτελεί το επόμενο βήμα. Η μνήμη ROM, τότε, εξάγει την επόμενη προς εκτέλεση νότα-εντολή. Παρακάτω ακολουθεί το σχήμα 4.5 όπου φαίνεται το κύκλωμα του απαριθμητή – συγκριτή (Counter_16Comp).

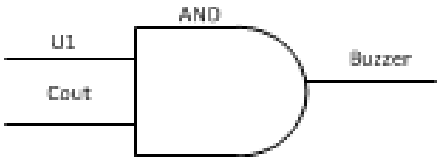


Σχήμα 4.5 Κύκλωμα Απαριθμητή – Συγκριτή (Counter_16Comp)

4.6 Πύλη AND

Η πύλη AND όπως φαίνεται στο σχήμα 4.6 αποτελείται από δύο εισόδους, την U1 και Cout και από μία έξοδο την Buz. Η είσοδος U1 είναι η έξοδος του πολυπλέκτη mux1 και η είσοδος Cout είναι η έξοδος του κυκλώματος απαριθμητή – συγκριτή (Counter_16Comp) όταν λαμβάνει την λογική τιμή 1. Η έξοδος Buzzer οδηγεί στον βομβητή όπου αναπαράγεται η κάθε νότα.

Συγκεκριμένα, στην πρώτη είσοδο της πύλης AND (U1) καθορίζεται ποια νότα θα αναπαράγεται κάθε φορά και στη δεύτερη είσοδο της πύλης AND (Cout) ορίζεται πόση θα είναι η περίοδος της συγκεκριμένης νότας (π.χ. 1/4, 1/8, 1/16 κ.τ.λ.) . Παρακάτω στο σχήμα 4.6 ακολουθεί η πύλη AND.



Σχήμα 4.6 Πύλη AND

4.7 Μνήμη ROM

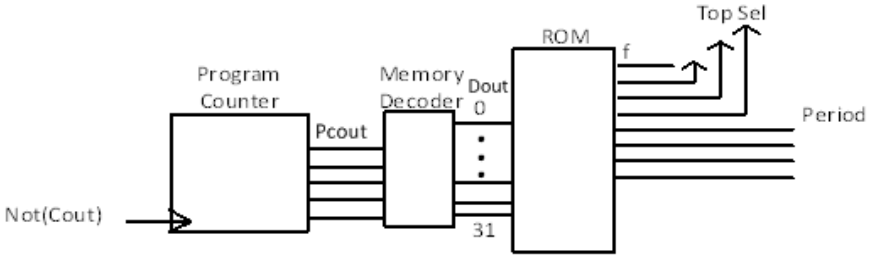
Η μνήμη ROM όπως φαίνεται στο σχήμα 4.7 αποτελείται από έναν απαριθμητή με όνομα Program Counter, έναν αποκωδικοποιητή με όνομα Memory Decoder και μία μνήμη ROM.

Ο απαριθμητής (Program Counter) αποτελείται από την είσοδο NOT, όπου είναι η αντιστραμμένη έξοδος Cout του κυκλώματος Counter_16Comp που αναλύθηκε στη ενότητα 4.5 και από μία έξοδο με όνομα Pcout των πέντε Bits. Η έξοδος του Program Counter συνδέεται ως είσοδος στον αποκωδικοποιητή Memory Decoder. Ο Program Counter αυξάνεται κάθε φορά που παρέρχεται η διάρκεια της κάθε νότας.

Ο αποκωδικοποιητής Memory Decoder αποτελείται από μία είσοδο με όνομα Pcout των πέντε Bits και μία έξοδο με όνομα Dout των τριάντα δύο Bits (Decoder 5 to 32). Στον αποκωδικοποιητή γίνεται η επιλογή της νότας που έρχεται κάθε φορά και επιλέγεται η κάθε μία από τις 32 θέσεις του πίνακα ROM.

Η μνήμη ROM αποτελείται από μία είσοδο των τριάντα δύο Bits και τρεις εξόδους, την f του ενός Bit, την topsel των τριών Bits και την Period των τεσσάρων Bits. Η είσοδος της μνήμης ROM είναι η έξοδος του Memory Decoder. Η έξοδος topsel είναι τα τρία bits της γραμμής επιλογής του πολυπλέκτη (mux1) όπου καθορίζουν το είδος της νότας και η έξοδος Period είναι η δεύτερη είσοδος B του συγκριτή (Comparator) όπου ορίζεται η περίοδος της επιλεγμένης νότας.

Η μνήμη ROM είναι το κύκλωμα του μουσικού επεξεργαστή που περιέχει αποθηκευμένες τις μουσικές νότες του τραγουδιού που αναπαράγεται. Συγκεκριμένα, επιλέχτηκε εξαιτίας της μουσικής του απλότητας το τραγούδι του εθνικού μας ύμνου. Παρακάτω στο σχήμα 4.7 ακολουθεί το κύκλωμα της μνήμης ROM.



Σχήμα 4.7 Μνήμη ROM

5^{ο} ΚΕΦΑΛΑΙΟ – ΚΩΔΙΚΑΣ ΤΟΥ ΜΟΥΣΙΚΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

5.1 Γενικά

Όπως αναφέρθηκε σε προηγούμενα κεφάλαια η γλώσσα περιγραφής υλικού που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής είναι η VHDL. Στις παρακάτω ενότητες ακολουθεί ο κώδικας του κάθε κυκλώματος που αναφέρθηκε στο προηγούμενο κεφάλαιο.

Για την υλοποίηση του μουσικού επεξεργαστή δημιουργήθηκαν δέκα αρχεία VHDL. Τα επτά αρχεία περιέχουν τον κώδικα των κυκλωμάτων (βλέπε κεφάλαιο 4^{ο}), τα δύο αρχεία περιέχουν τις δηλώσεις μεταβλητών των κυκλωμάτων και το ένα αρχείο περιέχει τη δήλωση του πίνακα ROM του κυκλώματος της μνήμης ROM. Ονομαστικά τα αρχεία αυτά είναι τα εξής:

- Το αρχείο top_package.vhd
- Το αρχείο my_Package1.vhd
- Το αρχείο ROM.vhd
- Το αρχείο Program_counter.vhd
- Το αρχείο Rom_Decoder.vhd
- Το αρχείο multiplexer.vhd
- Το αρχείο counter_16_comp.vhd
- Το αρχείο Count.vhd
- Το αρχείο top1.vhd
- Το αρχείο and_gate.vhd

5.2 Αρχείο top_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
PACKAGE top_package IS

TYPE OneD IS ARRAY(NATURAL RANGE<>) OF std_logic_vector(7 downto 0);
End PACKAGE;
```

Στο συγκεκριμένο αρχείο περιέχεται η δήλωση του πίνακα της ROM.

5.3 Αρχείο my_package1.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use work.top_package.all;

PACKAGE my_package1 IS

COMPONENT multiplexer IS
port(  x: in std_logic_vector(7 downto 0);
    sel: in std_logic_vector(2 downto 0);
     y:out std_logic);
end COMPONENT;

COMPONENT counter_16_comp IS
port(  clk:  in std_logic;
    resetn:  in std_logic;
   enable:  in std_logic;
   period:  in std_logic_vector(3 downto 0);
    leq : OUT std_logic);
end COMPONENT;
```

```
COMPONENT Count is
port(clk, res, en : in std_logic;
      q:out std_logic_vector(7 downto 0);
      q1:out std_logic);
end COMPONENT;

COMPONENT ROM is
port(song : out OneD(0 to 31));
end COMPONENT;

COMPONENT ROM2 is
port(song : out OneD(0 to 31));
end COMPONENT;

COMPONENT Program_Counter is
port(clk, rset : in std_logic;
      q:out std_logic_vector(4 downto 0));
end COMPONENT;

COMPONENT Rom_Decoder is
port(dec_in : in std_logic_vector(4 downto 0);
    f : out std_logic_vector(7 downto 0));
end COMPONENT;

END PACKAGE;
```

Σε αυτό το αρχείο περιέχονται οι δηλώσεις των μεταβλητών των κυκλωμάτων του μουσικού επεξεργαστή.

5.4 Αρχείο count.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count is
port (clk,res,en: in std_logic;
                          q: out std_logic_vector(7 downto 0);
                         q1: out std_logic);
end count;

architecture behavior of count is
signal m1: std_logic_vector (15 downto 0);
signal m2: std_logic_vector (15 downto 0);
signal m3: std_logic_vector (15 downto 0);
signal m4: std_logic_vector (15 downto 0);
signal m5: std_logic_vector (15 downto 0);
signal m6: std_logic_vector (15 downto 0);
signal m7: std_logic_vector (15 downto 0);
signal m8: std_logic_vector (15 downto 0);
signal m: std_logic_vector (19 downto 0);
begin


process (clk, res)
begin
        if res= '0' then
          m1<= (OTHERS=>'0');
        elsif clk 'event and clk='1' then
            if en='1' then
               if m1<19111 then
```

```
 m1<= m1+1;
    if m1<9555 then
    q(0)<='0';
    else
                    q(0)<='1';
                    end if;
            else
  m1<=(OTHERS=>'0');
  end if;
 end if;
 if en='1' then
            if m2<17036 then
  m2<= m2+1;
    if m2<8518 then
    q(1)<='0';
    else
                    q(1)<='1';
                    end if;
            else
  m2<=(OTHERS=>'0');
            end if;
     end if;

if en='1' then
            if m3<15168 then
  m3<= m3+1;
    if m3<7584 then
    q(2)<='0';
    else
                    q(2)<='1';
                    end if;
            else
```

```
        m3<=(OTHERS=>'0');
                end if;
            end if;

if en='1' then
                if m4<14317 then
        m4<= m4+1;
            if m4<7158 then
            q(3)<='0';
            else
                        q(3)<='1';
                        end if;
                else
        m4<=(OTHERS=>'0');
                end if;
            end if;

if en='1' then
                if m5<12755 then
        m5<= m5+1;
            if m5<6377 then
            q(4)<='0';
            else
                        q(4)<='1';
                        end if;
                else
        m5<=(OTHERS=>'0');
                end if;
            end if;

if en='1' then
                if m6<11363 then
```

```
        m6<= m6+1;
            if m6<5681 then
            q(5)<='0';
            else
                                q(5)<='1';
                                end if;
                        else
        m6<=(OTHERS=>'0');
                        end if;
            end if;

if en='1' then
                        if m7<10123 then
        m7<= m7+1;
            if m7<5061 then
            q(6)<='0';
            else
                                q(6)<='1';
                                end if;
                        else
        m7<=(OTHERS=>'0');
                        end if;
            end if;

if en='1' then
                        if m8<9555 then
        m8<= m8+1;
            if m8<4777 then
            q(7)<='0';
            else
                                q(7)<='1';
                                end if;
```

```
                    else
            m8<=(OTHERS=>'0');
                    end if;
                end if;
        end if;
 end process;

process (clk)
begin
        if clk 'event and clk='1' then
        m<= m+1;
        end if;
end process;
q1<=m(19);
end behavior;
```

Σε αυτό το αρχείο υποδιαιρούνται τα 10MHZ του clk τόσες φορές έτσι ώστε η έξοδος q να βγάζει τις συχνότητες της κάθε νότας.

5.6 Αρχείο multiplexer.vhd

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity multiplexer is
Port ( x:in STD_LOGIC_VECTOR (7 downto 0);
    sel:in STD_LOGIC_VECTOR (2 downto 0);
     y: out STD_LOGIC);
end multiplexer;
architecture Behaviour of multiplexer is
begin
```

```
process (x,sel)
begin
case sel is
when "000"=>y<=x(0);
when "001"=>y<=x(1);
when "010"=>y<=x(2);
when "011"=>y<=x(3);
when "100"=>y<=x(4);
when "101"=>y<=x(5);
when "110"=>y<=x(6);
when "111"=>y<=x(7);
when others=> null;
end case;
end process;
end Behaviour;
```

Σε αυτό το αρχείο η είσοδος sel ορίζει ποια από τις οχτώ νότες θα βγαίνει στην έξοδο y.

5.7 Αρχείο counter_16_comp.vhd

```
Library ieee ;
use ieee.std_logic_1164.all;
 use ieee.std_logic_unsigned.all;

entity counter_16_comp is
port(  clk:  in std_logic;
     resetn:  in std_logic;
   enable:  in std_logic;
   period:  in std_logic_vector(3 downto 0);
```

```
    leq : OUT std_logic);
end counter_16_comp;

architecture behavior of counter_16_comp is
signal m: std_logic_vector(3 downto 0);
begin
 process(clk, enable, resetn)
  begin
     if resetn = '0' then
      m <= "0000";
     elsif (clk='1' and clk'event) then
       if enable = '1' then

        if (m<period OR m=period) then
                              m <= m + 1;
                              leq<='1';
                              else
                              m<="0000";
                              leq<='0';
                              end if;
      end if;
     end if;
 end process;
 --   leq <= m;
end behavior;
```

Αυτό το αρχείο περιέχει ένα ενιαίο κύκλωμα με όνομα counter_16comp και αποτελείται από έναν απαριθμητή δεκαέξι καταστάσεων και έναν συγκριτή.

5.8 Αρχείο ROM.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use work.top_package.all;

ENTITY ROM IS
port(song : out OneD(0 to 31));
END ENTITY;

ARCHITECTURE my_song OF ROM IS
Begin

song(0)<="01000010";--sol
song(1)<="01010010";--la
song(2)<="01100010";--si
song(3)<="01110110";--nto
song(4)<="01100010";--si
song(5)<="01010010";--la
song(6)<="01000110";--sol
song(7)<="00110110";--fa
song(8)<="00101010";--mi
song(9)<="01000110";--sol
song(10)<="00110010";--fa
song(11)<="00100010";--mi
song(12)<="00010110";--re
song(13)<="00010010";--re
song(14)<="00100010";--mi
song(15)<="00110010";--fa
song(16)<="01000010";--sol
song(17)<="01010010";--la
```

```
song(18)<="01000110";--sol
song(19)<="01010110";--la
song(20)<="01000110";--sol
song(21)<="00001000";--STOP

End ARCHITECTURE;
```

Αυτό το αρχείο αποτελείται από έναν πίνακα ROM τριάντα δύο θέσεων, όπου περιέχονται οι νότες του τραγουδιού.

5.9 Αρχείο Program_counter.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Program_Counter is
port(clk, rset : in std_logic;
      q:out std_logic_vector (4 downto 0));
end Program_Counter;

architecture behavior of Program_Counter is
 signal m: std_logic_vector(4 downto 0);
 begin
    process(clk,rset)
     begin
        if rset='0' then
        m<="00000";
                    elsif(clk 'event and clk='1') then
                                if m>"10011" then
```

```
                    m<="00000";
                    else
                    m<= m+1;
                    end if;
                end if;
    end process;
     q<=m;
  end behavior;
```

Σε αυτό το αρχείο ο απαριθμητής αυξάνεται κάθε φορά που παρέρχεται η διάρκεια της κάθε νότας.

5.10 Αρχείο ROM_Decoder.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use work.top_package.all;
use work.my_package1.all;

ENTITY Rom_Decoder is
port (dec_in : in std_logic_vector(4 downto 0);
    f : out std_logic_vector(7 downto 0));
END ENTITY;

ARCHITECTURE decode OF Rom_Decoder IS

signal u : OneD(0 to 31);
BEGIN
inst_song: ROM Port Map (u);
with dec_in SELECT
```

```
f<=u(0) when "00000",
   u(1) when "00001",
   u(2) when "00010",
   u(3) when "00011",
   u(4) when "00100",
   u(5) when "00101",
   u(6) when "00110",
   u(7) when "00111",
   u(8) when "01000",
   u(9) when "01001",
   u(10) when "01010",
   u(11) when "01011",
   u(12) when "01100",
   u(13) when "01101",
   u(14) when "01110",
   u(15) when "01111",
   u(16) when "10000",
   u(17) when "10001",
   u(18) when "10010",
   u(19) when "10011",
   u(20) when "10100",
   u(21) when "10101",
   u(22) when "10110",
   u(23) when "10111",
   u(24) when "11000",
   u(25) when "11001",
   u(26) when "11010",
   u(27) when "11011",
   u(28) when "11100",
   u(29) when "11101",
   u(30) when "11110",
   u(31) when others;
```

```
End ARCHITECTURE;
```

Αυτό το αρχείο αποτελείται από έναν αποκωδικοποιητή τριάντα δύο θέσεων. Για τη νότα που έρχεται κάθε φορά επιλέγεται η κάθε μία από τις τριάντα δύο θέσεις του πίνακα ROM.

5.11 Αρχείο and_gate.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_unsigned.all;

entity AND_gate is
port(a : in std_logic;
    b : in std_logic;
    o : out std_logic);
end AND_gate;

architecture behavior OF AND_gate IS
begin
        o<=(a AND b);
end behavior;
```

Το αρχείο αυτό αποτελείται από την πύλη AND όπου η έξοδος της πηγαίνει στον Buzzer.

5.12 Αρχείο top1.vhd

```
Library ieee;
use ieee.std_logic_1164.all;
use work.my_package1.all;

ENTITY top1 is
port(osc :in std_logic;
--          topsel :in std_logic_vector(2 downto 0);
            en: in std_logic;
    rstn:  in std_logic;
--          per :  in std_logic_vector(3 downto 0);
            buzzer: out std_logic);
end top1;
ARCHITECTURE Behavior Of top1 is
Signal u1 ,u2:std_logic;
Signal topx : std_logic_vector(7 downto 0);
Signal int_clk, rst: std_logic;
Signal count_out: std_logic_vector(4 downto 0);
Signal Rom_Out : std_logic_vector(7 downto 0);
Signal topsel : std_logic_vector(2 downto 0);
Signal per :  std_logic_vector(3 downto 0);
Begin

freq: Count Port Map (osc, rstn, en, topx,int_clk);
mux1: multiplexer Port Map(topx, topsel,u1);
count_comp: counter_16_comp Port Map(int_clk, rstn, en, per,u2);
buzzer<=(u1 AND u2);
inst_count: Program_Counter Port Map (not(u2),rstn,count_out);
inst_dec: Rom_Decoder Port Map (count_out,Rom_Out);
per<=Rom_Out(3 downto 0);
```

```
topsel<=Rom_Out(6 downto 4);

END Behavior;
```

Το αρχείο Top1.vhd αποτελεί το top level entity του μουσικού επεξεργαστή. Δέχεται την είσοδο του OSC καθώς και τις εισόδους en και rstn και βγάζει την έξοδο της πύλης AND στον Buzzer για να ακουστεί η νότα. Επίσης στο αρχείο αυτό γίνονται όλες οι συνδέσεις των προηγούμενων αρχείων.

Βιβλιογραφία

1. Εισαγωγή στη VHDL, Ιωάννης Καλόμοιρος, ΤΕΙ Κεντρικής Μακεδονίας, Σέρρες, 2012.

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

3. Stephen Brown and Zvonko Vranesic, Σχεδίαση Ψηφιακών Συστημάτων Με Την Γλώσσα VHDL, Εκδόσεις Τζιόλας, 2002.