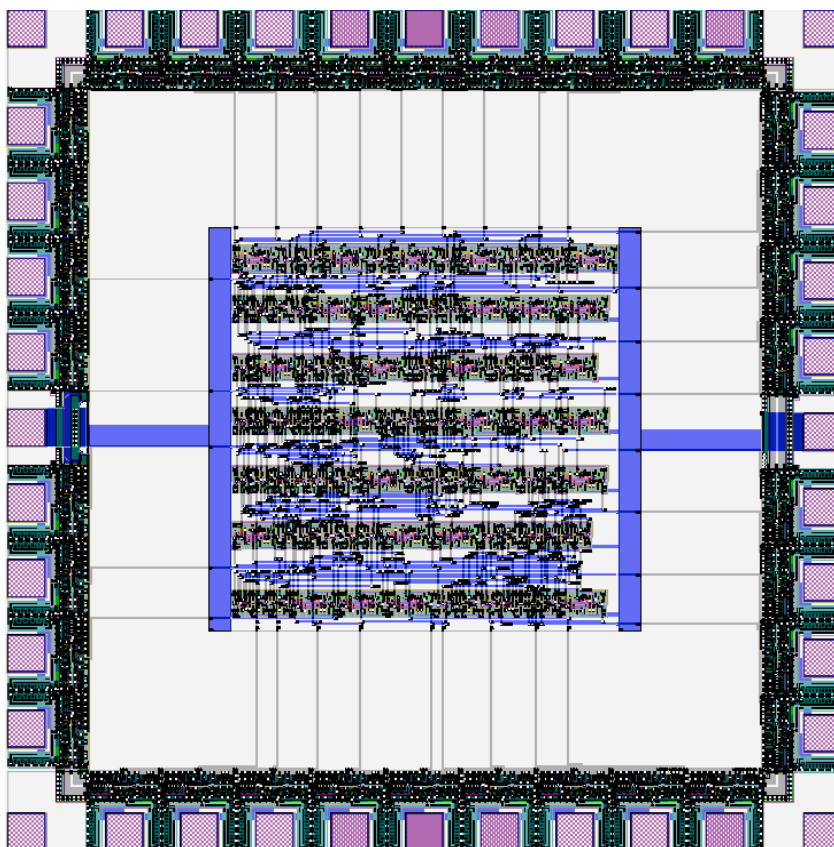




Εργαστηριακές Ασκήσεις Σχεδίασης Ψηφιακών Συστημάτων με χρήση Η/Υ



Computer Aided Design

Χρ. Καβουσιανός
Αναπληρωτής Καθηγητής

Φεβρουάριος 2018

Εισαγωγή

Το εργαλείο Quartus II παρέχει ένα περιβάλλον σχεδιασμού συστημάτων ανεξάρτητο αρχιτεκτονικής, με πολλαπλές πλατφόρμες. Δίνει την δυνατότητα ολοκληρωμένου σχεδιασμού συστημάτων, γρήγορης επεξεργασίας και άμεσου προγραμματισμού των συσκευών της Altera (πχ. Classic, MAX5000-7000-9000, FLEX6000-8000-10K, Cyclone). Καλύπτει όλο το φάσμα λογικού σχεδιασμού, με δυνατότητες δημιουργίας πολύπλοκων και ιεραρχικών σχεδιασμών, δυναμική σύνθεση, διαμέριση, λειτουργική και χρονική εξομοίωση, χρονική ανάλυση, αυτόματο εντοπισμό λαθών, προγραμματισμό συσκευών και επιβεβαίωση της λειτουργίας τους. Γίνονται αποδεκτοί σχεδιασμοί σε VHDL, Verilog, AHDL (Altera HDL) καθώς και σχηματικά διαγράμματα που δημιουργούνται από τον ειδικό γραφικό Editor του εργαλείου. Επίσης μπορεί να επικοινωνήσει και με άλλα εργαλεία χρησιμοποιώντας αρχεία netlist τύπου Edif ή Xilinx, και SDF.

Ο compiler είναι μία από τις ισχυρές δυνατότητες του Quartus και δίνει την καλύτερη δυνατή υλοποίηση του συστήματος. Με δυνατότητες αυτόματου εντοπισμού των λαθών στον αρχικό σχεδιασμό ή στην υλοποιημένη μορφή του στο FPGA καθώς και με την εκτεταμένη τεκμηρίωση λαθών διευκολύνει κατά πολύ την διαδικασία σχεδιασμού.

Διαδικασία Σχεδιασμού

Τα στάδια δημιουργίας ενός σχεδιασμού από την σύλληψη έως και την ολοκλήρωσή του είναι τα ακόλουθα:

1. Δημιουργία αρχείου σχεδιασμού ή ιεραρχίας σχεδιασμών (VHDL, Verilog, Graphic Design κλπ).
2. Επιλογή μίας προγραμματιζόμενης συσκευής (η συσκευή που θα χρησιμοποιούμε πάντα είναι η Cyclone II EP2C35F672C6).
3. Σύνθεση του σχεδιασμού με παραγωγή χρονικής πληροφορίας και εκτέλεση χρονικής εξομοίωσης και χρονικής ανάλυσης.
4. Προγραμματισμός της συσκευής με χρήση της ειδικής προγραμματιστικής μονάδας (board).

Λογισμικό

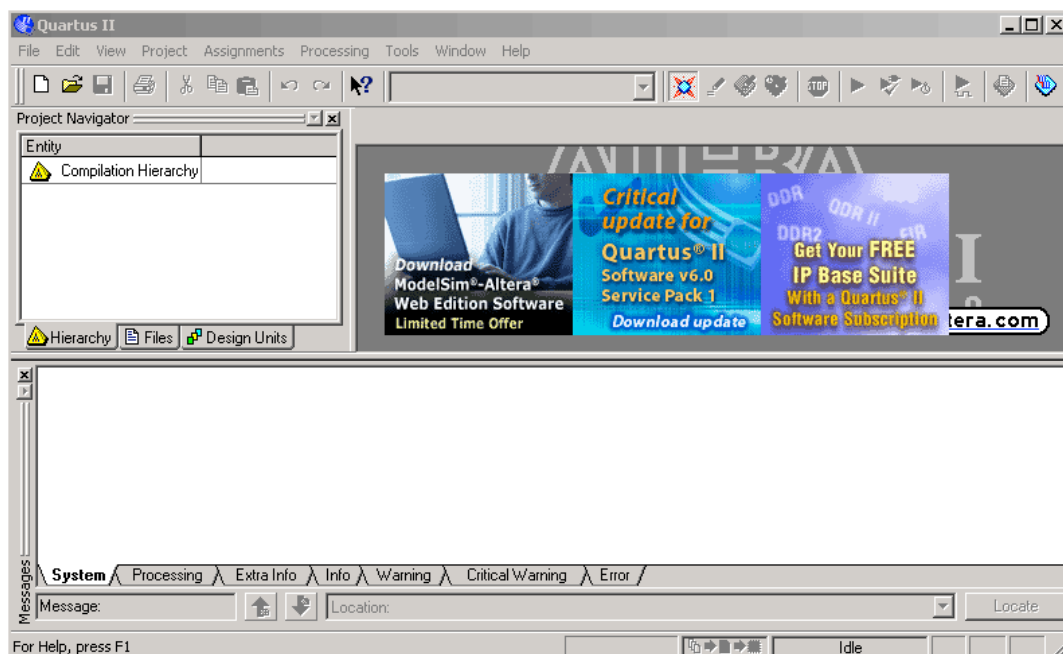
Μπορείτε να κατεβάσετε το λογισμικό (με περιορισμένες αλλά επαρκείς δυνατότητες) από την Altera (<https://www.altera.com/downloads/download-center.html>) αφού πρώτα δημιουργήσετε έναν λογαριασμό student με τα πραγματικά στοιχεία σας και το email που διαθέτετε στο τμήμα Μηχανικών Η/Υ & Πληροφορικής. Η έκδοση του προγράμματος που θα χρησιμοποιήσουμε είναι η Altera Quartus II, vs9.1, sp2 Web Edition.

Υλοποίηση Ασκήσεων

Τις παρακάτω ασκήσεις θα τις υλοποιήσετε στο σπίτι σας. Ωστόσο θα πρέπει να δείξετε τα αποτελέσματα στους επιτηρητές στο εργαστήριο σε χρόνο που θα ανακοινωθεί. Επιπλέον υπάρχουν τμήματα των ασκήσεων σε πλαίσιο που μπορείτε να εκτελέσετε μόνο στο εργαστήριο και εφόσον έχετε ολοκληρώσει και επιβεβαιώσει την χρονική εξομοίωση των αντίστοιχων κυκλωμάτων.

Εργαστηριακή Άσκηση 1: Το περιβάλλον εργασίας και απλά ψηφιακά κυκλώματα

Στα πλαίσια της πρώτης εργαστηριακής άσκησης θα κάνουμε συνοπτική παρουσίαση του εργαλείου. Για λεπτομέρειες μπορεί να ανατρέξει ο χρήστης στο εγχειρίδιο χρήσης του.



Εικόνα 1. Περιβάλλον εργασίας

Το βασικό περιβάλλον εργασίας του εργαλείου παρουσιάζεται στην Εικόνα 1. Και σε αυτήν την περίπτωση ακολουθούνται οι βασικές αρχές των παραθυρικών εφαρμογών για Windows, όπως ο τίτλος, το βασικό μενού εντολών και τα κουμπιά συντόμευσης. Κάποιες από τις βασικές λειτουργίες που παρέχονται από το εργαλείο είναι οι ακόλουθες:

- **Hierarchy Display.** Παρέχει πρόσβαση σε όλα τα τμήματα της ιεραρχίας του σχεδιασμού.
- **Graphic Editor.** Χρησιμοποιείται για την δημιουργία κυκλωμάτων ή δομικών (structural) περιγραφών με χρήση λογικών και άλλων συμβόλων.
- **Text Editor.** Είναι ειδικά διαμορφωμένος κειμενογράφος για την συγγραφή περιγραφών με χρήση γλωσσών HDL.
- **Waveform Editor.** Χρησιμοποιείται για την δημιουργία κυματομορφών και την τροφοδότηση τους στις εισόδους του κυκλώματος κατά την διαδικασία της εξομίωσης.
- **Floorplan Editor.** Παρουσιάζει την τελική εικόνα της συσκευής μετά τον προγραμματισμό της (αντιστοιχία pins-εισόδων/εξόδων, υλοποίηση λογικής στο FPGA, διασύνδεση λογικών τμημάτων κλπ).
- **Compiler.** Συνθέτει τον σχεδιασμό χρησιμοποιώντας τα resources μιας επιλεγμένης συσκευής.
- **Simulator.** Οπτικοποιεί τον σχεδιασμό χρησιμοποιώντας λογική ή και χρονική πληροφορία, με βάση τις κυματομορφές εισόδου που όρισε ο χρήστης.
- **Timing Analyzer.** Αναλύει χρονικά τον σχεδιασμό (εύρεση κρίσιμων μονοπατιών, μέγιστης συχνότητας λειτουργίας κλπ)
- **Programmer.** Προγραμματίζει την συσκευή που επιλέξαμε.

Οι περισσότερες από τις παραπάνω λειτουργίες βρίσκονται κάτω από τις επιλογές *Tools - Processing*. Από την επιλογή *File* μπορούμε να ανοίξουμε ή να δημιουργήσουμε αρχεία σχεδιασμού και project. Με την επιλογή *Assignments* μπορούμε να καθορίσουμε διάφορες παραμέτρους του σχεδιασμού.

Μέρος 1^ο: Βασικά Χαρακτηριστικά του board DE2

Ακόλουθα θα δημιουργήσουμε ένα απλό σχηματικό δύο εισόδων ώστε να εξοικειωθούμε με το περιβάλλον σχεδίασης. Το κύκλωμα αυτό θα ελέγχει δύο διακόπτες (pushbuttons) και θα σηματοδοτεί τότε ένας τουλάχιστον εκ των δύο είναι πατημένος. Θα εξομοιώσουμε το κύκλωμα και αφού επιβεβαιώσουμε την σωστή λειτουργία του θα προγραμματίσουμε την συσκευή Cyclone που βρίσκεται πάνω στο board DE2.

Η υλοποίηση ενός κυκλώματος, απαιτεί εισόδους και εξόδους. Ως εισόδους θα χρησιμοποιήσουμε δύο διακόπτες του board και σαν έξοδο θα χρησιμοποιήσουμε ένα LED (light emitting diode). Τόσο οι διακόπτες όσο και το LED αλλά και άλλες περιφερειακές συσκευές είναι συνδεδεμένες με το FPGA μέσω καλωδίωσης που παρέχεται από το board. Για τον λόγο αυτό θα πρέπει να γνωρίζουμε σε ποιο pin του FPGA βρίσκονται συνδεδεμένες ώστε να χρησιμοποιούμε τα σωστά pins του FPGA για τις λειτουργίες που θέλουμε.

Διακόπτες (PushButtons)

Τέσσερις διακόπτες της πλακέτας (pushbuttons) βρίσκονται κάτω δεξιά και έχουν τις ετικέτες KEY0-3. Στην παρούσα άσκηση θα χρησιμοποιήσουμε το KEY0 το οποίο θα ονομάσουμε PB1 και το KEY1 το οποίο θα ονομάσουμε PB2. Οι δύο αυτοί διακόπτες έχουν ήδη συνδεθεί στους ακροδέκτες (pins) PIN_G26 και PIN_N23 αντίστοιχα της προγραμματιζόμενης συσκευής Cyclone (δείτε αναλυτικά τον πίνακα στο Παράρτημα Α). Όταν πιέζουμε έναν από αυτούς τους διακόπτες οι αντίστοιχοι ακροδέκτες της συσκευής αλλάζουν κατάσταση. Συγκεκριμένα, όταν πιέζουμε τους διακόπτες παίρνουμε λογικό 0 στους ακροδέκτες ενώ όταν τους αφήνουμε ελεύθερους παίρνουμε λογικό 1. Προσέξτε αυτή την λειτουργία καθώς είναι αντιστροφή από αυτή που ίσως θα περιμέναμε.

LEDs (Light Emitting Diodes)

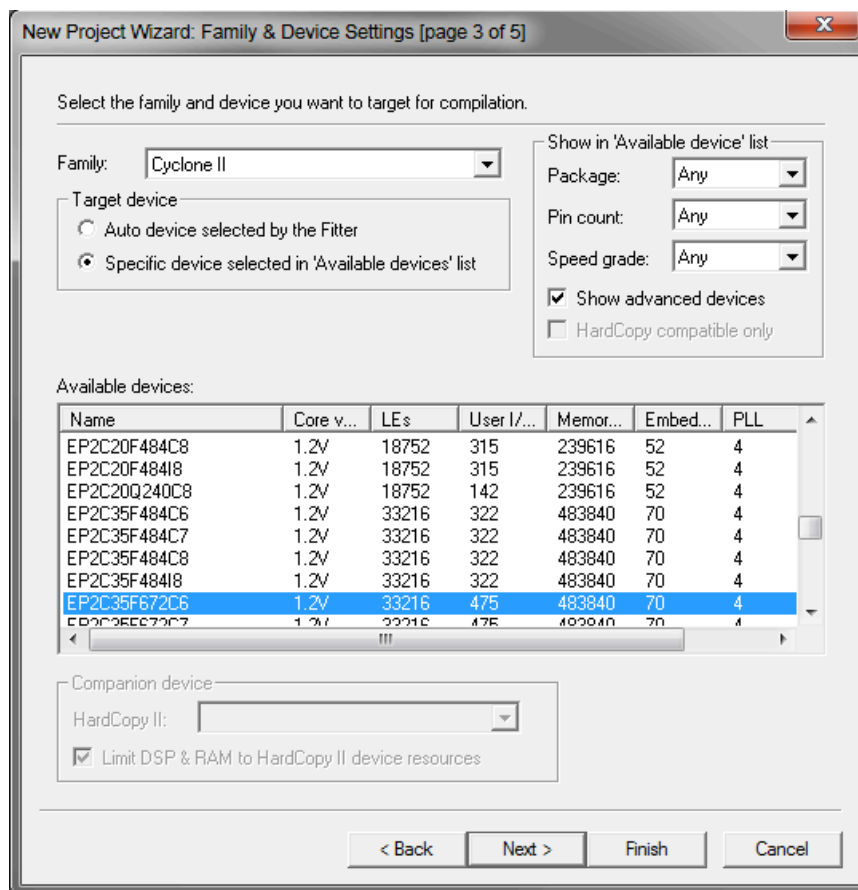
26 LEDs βρίσκονται στο κάτω μέρος της πλακέτας και έχουν ετικέτες LEDG0-7, LEDR0-17. Στην παρούσα άσκηση θα χρησιμοποιήσουμε το LEDG0 το οποίο έχει συνδεθεί στον ακροδέκτη PIN_AE22 της προγραμματιζόμενης συσκευής Cyclone (Παράρτημα Α). Όταν ο ακροδέκτης έχει την τιμή 0 τότε το LED είναι σβηστό αλλιώς ανάβει.

Το Ζητούμενο Κύκλωμα

Στην παρούσα εργαστηριακή άσκηση θα σχεδιάσουμε ένα κύκλωμα το οποίο θα δέχεται λογικές τιμές από τους δύο διακόπτες και θα ανάβει το LED όταν ένας τουλάχιστον από τους δύο πιεστεί. Με άλλα λόγια θα υλοποιεί την απλή λογική πράξη OR τροφοδοτώντας την έξοδο στο LED. Προσέξτε ότι οι διακόπτες όταν πιέζονται παρέχουν λογικό 0 αλλιώς παρέχουν λογικό 1. Για τον λόγο αυτό θα πρέπει να αντιστραφούν. Έστω PB1, PB2 οι είσοδοι από τους διακόπτες, και LED η έξοδος προς το LED. Η λογική συνάρτηση που πρέπει να υλοποιήσουμε είναι η ακόλουθη $LED = PB1' + PB2'$.

Δημιουργία Project

Πρέπει να ακολουθούμε με προσοχή τα βήματα δημιουργίας ενός Project κάθε φορά που δημιουργούμε ένα νέο Project. Σε περίπτωση που δεν εφαρμόσουμε σωστά την διαδικασία τότε δεν θα μπορούμε να αποπερατώσουμε την εργαστηριακή άσκηση.



Εικόνα 2. Ρυθμίσεις Οικογένειας

Αρχικά εκτελούμε την εντολή *File>New Project Wizard* οπότε ανοίγει το παράθυρο εισαγωγής του Project. Πιέζουμε *Next* και προχωράμε στο επόμενο παράθυρο όπου θα δηλώσουμε τον κατάλογο του Project, το όνομα του Project και το όνομα της υψηλότερης σε ιεραρχία οντότητας του σχεδιασμού μας. Προσωρινά χρησιμοποιούμε το ίδιο όνομα με αυτό που δώσαμε στο Project. Πιέζουμε *Next* και πάλι *Next* ώστε να παρακάμψουμε το επόμενο παράθυρο το οποίο προσθέτει αρχεία στο Project. Έτσι εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 2. Εδώ πρέπει να ορίσουμε ακριβώς την προγραμματιζόμενη συσκευή που θα χρησιμοποιήσουμε (Cyclone, EP2C35F672C6). Κάνουμε ακριβώς τις ρυθμίσεις που φαίνονται στην Εικόνα 2. Εάν επιλέξουμε άλλη συσκευή τότε δεν θα μπορούσαμε να εκτελέσουμε την άσκηση στο board. Πιέζουμε πάλι το *Next* δύο φορές ώστε να παρακάμψουμε το παράθυρο “EDA Tool Settings”. Το τελευταίο παράθυρο που εμφανίζεται περιέχει μία περίληψη των βασικών ρυθμίσεων του Project.

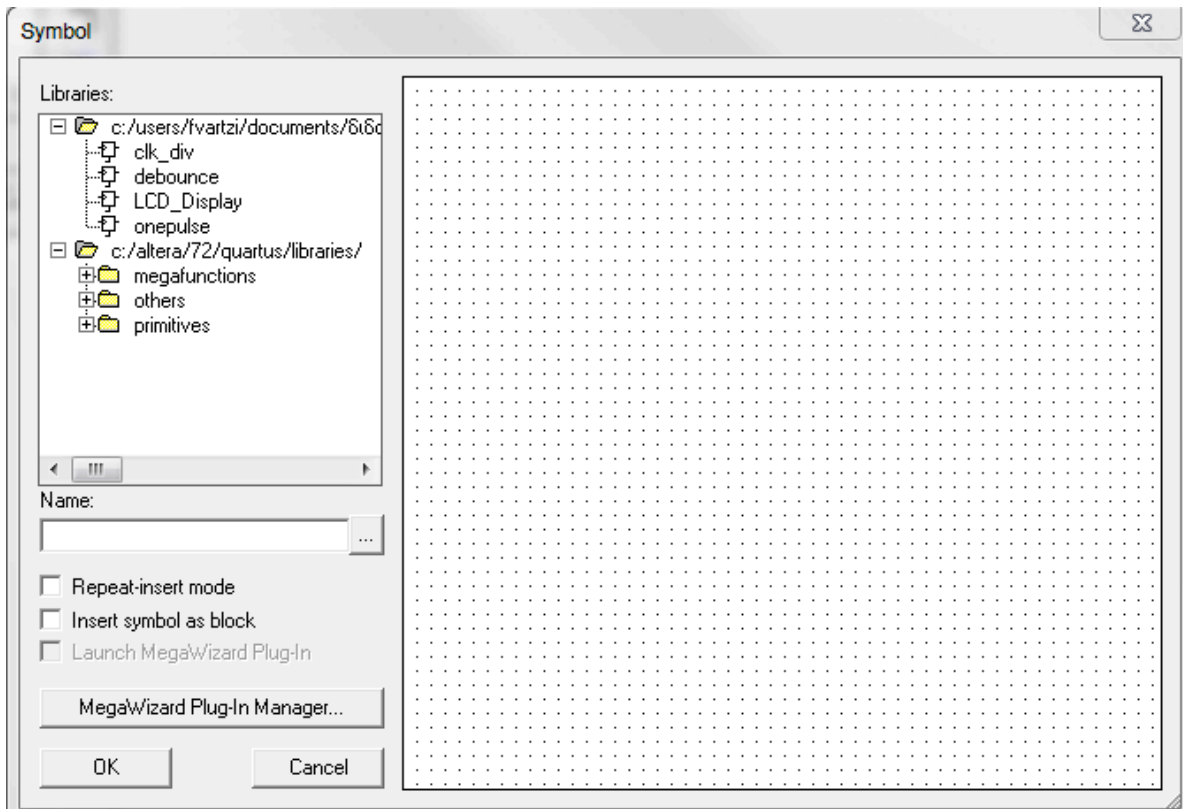
Η προγραμματιζόμενη συσκευή της Cyclone παρέχει έναν αριθμό από ακροδέκτες προκειμένου να μπορούμε να διασυνδέσουμε περιφερειακές συσκευές πάνω σε αυτήν. Στα πλαίσια των εργαστηριακών ασκήσεων μόνο ένα μικρό μέρος από αυτούς τους ακροδέκτες θα χρησιμοποιήσουμε. Οι υπόλοιποι θα είναι αχρησιμοποίητοι αλλά έχουν ήδη διασυνδεθεί σε διάφορες περιφερειακές συσκευές του board DE2 προβλέποντας την πιθανή χρήση τους. Για τον λόγο αυτό πρέπει να είμαστε πολύ προσεκτικοί ώστε να μην προκαλέσουμε βλάβη σε κάποια από αυτές και να μην κάψουμε την πλακέτα. Για τον λόγο αυτό είναι πολύ σημαντικό να καθορίσουμε όλα τα αχρησιμοποίητα pins ως εισόδους του FPGA. Εκτελούμε την εντολή “*Assignment>Device*” και στο παράθυρο που ανοίγει πιέζουμε το πλήκτρο “*Device & Pin Options*”. Επιλέγουμε την καρτέλα “*Unused Pins*” και εκεί επιλέγουμε “*As Inputs Tri-stated*” και κλείνουμε τα δύο παράθυρα. Την διαδικασία αυτή εφαρμόζουμε **ΥΠΟΧΡΕΩΤΙΚΑ** σε κάθε νέο project που δημιουργούμε.

Τελευταίο βήμα πριν ξεκινήσουμε την εισαγωγή του σχεδιασμού είναι ο καθορισμός των βιβλιοθηκών που θα χρησιμοποιήσουμε. Εκτελούμε την εντολή “*Project>Add Remove Files in Project*”. Στο πεδίο “*Category*” επιλέγουμε “*Libraries*”. Στο πεδίο του ονόματος της βιβλιοθήκης (Project Library Name) εισάγουμε την διαδρομή *c:\altera\CoreLibrary* η οποία βρίσκεται στην επιφάνεια εργασίας του

λογαριασμού σας και επιλέγουμε “Add” (μπορείτε να παρακάμψετε αυτό το βήμα καθώς έχει ήδη προστεθεί ως βιβλιοθήκη). Κλείνουμε τα παράθυρα και επανερχόμαστε στο βασικό περιβάλλον.

Εισαγωγή Σχεδίασης

Είμαστε πλέον έτοιμοι να εισάγουμε την σχεδίαση στον υπολογιστή. Στην 1^η άσκηση θα χρησιμοποιήσουμε σχηματικά πυλών. Για να υλοποιήσουμε την συνάρτηση $LED = PB1' + PB2'$ θα χρειαστούμε μία πύλη OR και δύο αντιστροφείς. Αρχικά εκτελούμε την εντολή *File>New>Block Diagram/Schematic File* οπότε εμφανίζεται το παράθυρο εργασίας που είναι ουσιαστικά ένα προκαθορισμένο πλέγμα (grid) πάνω στο οποίο μπορούμε να τοποθετήσουμε τα σύμβολα του κυκλώματος.



Εικόνα 3

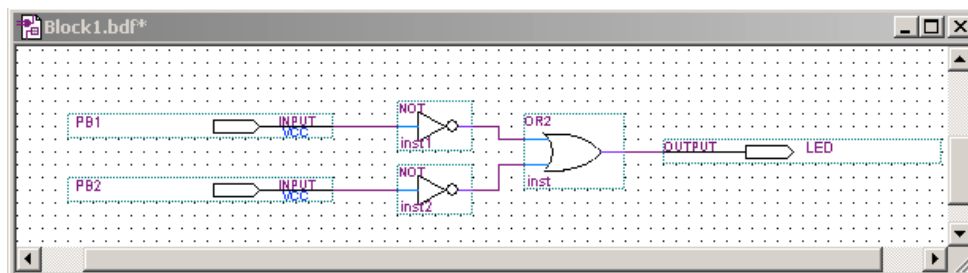
Για να τοποθετήσουμε κάποιο σύμβολο του κυκλώματος πρέπει αρχικά να τοποθετήσουμε τον ειδικό κέρσορα κάνοντας αριστερό κλικ στο ποντίκι, σε οποιοδήποτε σημείο του σχηματικού επιθυμούμε και κατόπιν να εκτελέσουμε την εντολή *Insert>Symbol* (με διπλό κλικ αριστερό ή κλικ δεξιό και επιλογή *Insert>Symbol*). Τότε ανοίγει το παράθυρο που φαίνεται στην Εικόνα 3. Στο πεδίο *Libraries* υπάρχουν όλες οι διαθέσιμες βιβλιοθήκες από τις οποίες μπορούμε να αντλήσουμε κύτταρα-πυρήνες (θα αναφερθούμε αργότερα σε αυτά). Παρέχονται οι ακόλουθες δύο βιβλιοθήκες:

1. *Altera/72/quartus/libraries*: Κύτταρα/πυρήνες που παρέχονται με την προγραμματιζόμενη συσκευή Cyclone. Αποτελείται από τις ακόλουθες τρεις υποκατηγορίες:
 - *Megafunctions*. Παραμετρικοί πυρήνες οι οποίοι υλοποιούν δομές με κανονικότητα όπως αριθμητικά κυκλώματα, μονάδες αποθήκευσης, αποκωδικοποιητές κλπ.
 - *Primitives*. Βασικά πρωταρχικά κύτταρα βιβλιοθήκης (πχ. λογικές πύλες ΚΑΙ, Η κλπ)
 - *Others*.
2. *altera/CoreLibrary*. Πρόσθετοι πυρήνες οι οποίοι χρησιμοποιούνται για την διασύνδεση της προγραμματιζόμενης συσκευής με τα περιφερειακά που βρίσκονται πάνω στο DE2 board (LCD Display, θύρες PS2, USB, RS232, Parallel κλπ).

Στην συγκεκριμένη περίπτωση θα πρέπει να εισάγουμε μία πύλη Η' και δύο αντιστροφείς για να υλοποιήσουμε την ζητούμενη συνάρτηση. Έτσι επιλέγουμε την βιβλιοθήκη *Primitives>Logic>Or2* οπότε εμφανίζεται το σχηματικό της πύλης OR. Πιέζουμε το OK οπότε παρατηρούμε ότι ο δείκτης του

ποντικιού αλλάζει στο σχηματικό της πύλης. Πατώντας το αριστερό πλήκτρο του ποντικιού σε οποιοδήποτε σημείο του πλέγματος τοποθετείται η πύλη στο σημείο εκείνο. Εάν εξακολουθεί ο δείκτης του ποντικιού να έχει το σχήμα της πύλης μπορούμε να τοποθετήσουμε και άλλες πύλες σε σημεία του πλέγματος (εάν το επιθυμούμε) ή να πατήσουμε το escape οπότε ο δείκτης επανέρχεται στην κανονική του μορφή. Με τον ίδιο τρόπο τοποθετούμε και τους αντιστροφείς (not) αριστερά της πύλης Or.

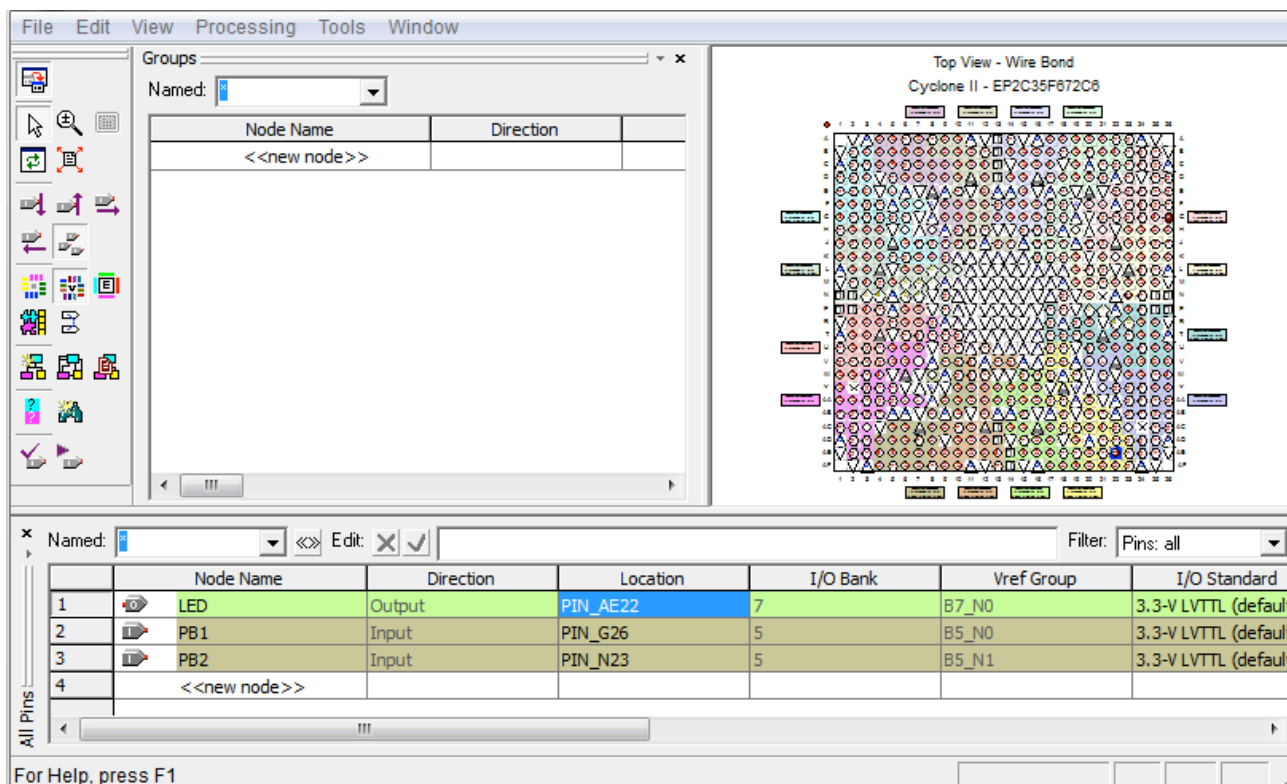
Το επόμενο βήμα είναι να συνδέσουμε τις πύλες μεταξύ τους. Συγκεκριμένα οι έξοδοι των δύο αντιστροφέων θα συνδεθούν στις εισόδους της πύλης Or. Εάν τοποθετήσουμε τον δείκτη του ποντικιού πάνω στο άκρο της εξόδου του αντιστροφέα παρατηρούμε ότι ο δείκτης αλλάζει σε σταυρό. Πατώντας το αριστερό πλήκτρο του ποντικιού και σύροντας το ποντίκι παρατηρούμε ότι διαγράφεται μία γραμμή η οποία ουσιαστικά αποτελεί την σύνδεση που επιθυμούμε. Σύρουμε το ποντίκι μέχρι την είσοδο της πύλης Or και κατόπιν το αφήνουμε ελεύθερο. Το ακριβές σημείο που πρέπει να αφήσουμε ελεύθερο το πλήκτρο του ποντικιού είναι η αρχή της εισόδου και συγκεκριμένα το σημείο εκείνο στο οποίο ο δείκτης μετατρέπεται σε τετράγωνο. Με τον ίδιο τρόπο συνδέουμε την έξοδο του άλλου αντιστροφέα στην δεύτερη είσοδο της πύλης OR. Παρατηρούμε ότι η κάθε σύνδεση μπορεί να διαγράψει μία γωνία κατά την δημιουργία της οπότε είναι πιθανό να μην μπορεί να φθάσει απευθείας στο σημείο που επιθυμείτε. Για τον λόγο αυτό μπορούμε να ελευθερώσουμε το πλήκτρο του ποντικιού σε οποιοδήποτε σημείο του πλέγματος και να ξεκινήσουμε από εκείνο το σημείο μία νέα σύνδεση η οποία ουσιαστικά θα είναι συνέχεια της προηγούμενης. Μία σύνδεση μπορεί να περάσει πάνω από μία άλλη χωρίς να δημιουργείται βραχυκύκλωμα. Αν ωστόσο θέλουμε να δημιουργήσουμε κάποιο βραχυκύκλωμα μπορούμε να μεταφέρουμε τον δείκτη του ποντικιού στο σημείο εκείνο και με δεξί κλικ στο ποντίκι και επιλογή του “Toggle Connection Dot” να τοποθετήσουμε την σύνδεση.



Εικόνα 4.

Το επόμενο βήμα είναι να δημιουργήσουμε τις εισόδους/εξόδους του κυκλώματος. Όπως τοποθετήσαμε τις πύλες, με τον ίδιο τρόπο εισάγουμε δύο εισόδους (*Primitives>Pin>Input*) στα αριστερά των δύο αντιστροφέων και μία έξοδο (*Primitives>Pin>Output*) στα δεξιά της πύλης Or. Αφού συνδέσουμε τις δύο εισόδους με τους αντιστροφείς και την έξοδο με την πύλη Or δίνουμε τα ονόματα των εισόδων/εξόδων (PB1, PB2, LED) επιλέγοντας μία προς μία τις θύρες και επιλέγοντας *Properties* με δεξιά κλικ στο ποντίκι. Στο πεδίο *Pin Name(s)* εισάγουμε το όνομα της κάθε θύρας. Το αποτέλεσμα φαίνεται στην Εικόνα 4.

Το επόμενο βήμα είναι να αντιστοιχίσουμε τις δύο εισόδους και την έξοδο σε κατάλληλους ακροδέκτες (Pins) της προγραμματιζόμενης συσκευής ώστε να μπορούμε να τροφοδοτήσουμε το κύκλωμα με εισόδους και να παρατηρήσουμε τις εξόδους όταν θα προγραμματίσουμε την συσκευή. Σε μία πραγματική υλοποίηση μπορούμε να χρησιμοποιήσουμε οποιαδήποτε pins μας βολεύουν κατά την σχεδίαση του συστήματος (PCB). Στην περίπτωση όμως του δικού μας board πρέπει να χρησιμοποιήσουμε συγκεκριμένα Pins τα οποία είναι ήδη συνδεδεμένα με διακόπτες που μπορούν να δώσουν είσοδο όπως και LEDs που μπορούν να ανάψουν όταν η λογική τιμή στο ανάλογο pin είναι η κατάλληλη. Όλα τα pins (ακροδέκτες) και οι συνδέσεις τους φαίνονται στο Παράρτημα Α. Για την είσοδο PB1 θα χρησιμοποιήσουμε το πλήκτρο KEY0 το οποίο όπως φαίνεται από το Παράρτημα Α αντιστοιχεί στον ακροδέκτη PIN_G26. Με όμοιο τρόπο βρίσκουμε ότι για την είσοδο PB2 για την οποία χρειαζόμαστε τον διακόπτη KEY1 θα χρησιμοποιήσουμε τον ακροδέκτη PIN_N23 και για το LED_G0 (έξοδος) θα χρησιμοποιήσουμε τον ακροδέκτη PIN_AE22.



Εικόνα 5

Για να αντιστοιχήσουμε τις θύρες εισόδου εξόδου σε συγκεκριμένους ακροδέκτες εκτελούμε την ακόλουθη διαδικασία: αρχικά εκτελούμε την διαδικασία μεταγλώττισης (compile) ώστε να μεταγλωττιστεί το σχηματικό διάγραμμα που δημιουργήσαμε. Εκτελούμε *Processing > Start Compilation*. Ακόλουθα εκτελούμε την εντολή *Assignments>Pin Planner* οπότε εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 5. Σε κάθε γραμμή του πίνακα ενεργοποιούμε την ανάθεση μίας θύρας σε έναν ακροδέκτη. Στο πεδίο *NodeName* πληκτρολογούμε το όνομα της θύρας και στο πεδίο *Location* το νούμερο του ακροδέκτη. Εάν έχουμε ήδη περάσει από μετάφραση (compilation) τον σχεδιασμό μας τότε με διπλό κλικ πάνω στο πεδίο *NodeName* εμφανίζεται η λίστα με τις υπάρχουσες θύρες οπότε δεν απαιτείται να πληκτρολογήσουμε τα ονόματά τους. Όταν ολοκληρώσουμε την ανάθεση των ακροδεκτών κλείνουμε το παράθυρο αποθηκεύοντας τις αλλαγές. Τώρα στο σχηματικό υπάρχουν και οι αριθμοί των ακροδεκτών σε κάθε θύρα.

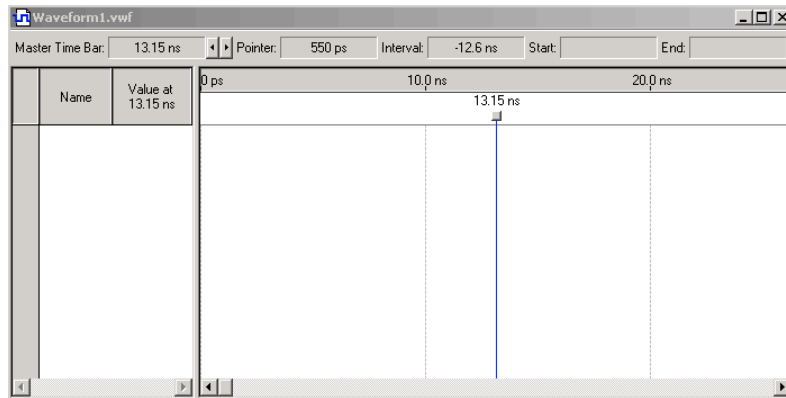
Πριν προχωρήσουμε στην εξομοίωση του σχεδιασμού πρέπει να τον αποθηκεύσουμε. **Ο σχεδιασμός θα πρέπει να έχει το ίδιο όνομα με το project. Σε περίπτωση που έχουμε κάποια ιεραρχική σχεδίαση τότε η κορυφαία οντότητα του σχεδιασμού θα πρέπει να έχει το όνομα του Project, αλλιώς ο compiler θα δώσει μήνυμα λάθους. Θα δούμε αργότερα πως μπορούμε να το παρακάμψουμε αυτό.**

Επιβεβαίωση της Σχεδίασης

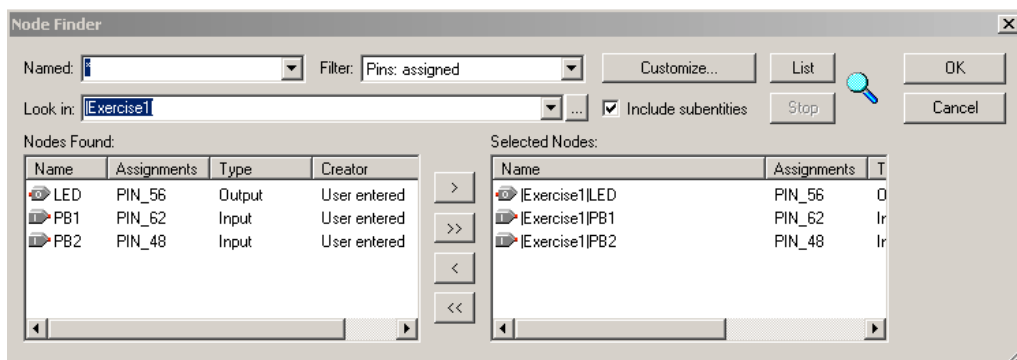
Εφόσον έχουμε τελειώσει με την εισαγωγή της σχεδίασης θα πρέπει να επαληθεύσουμε την ορθότητα της. Αρχικά πρέπει να συνθέσουμε την σχεδίαση (compilation) οπότε ελέγχονται συντακτικά λάθη, συντίθεται το κύκλωμα, παράγεται η χρονική πληροφορία για την εξομοίωση και παράγεται το απαραίτητο αρχείο για τον προγραμματισμό της συσκευής. Αυτή η διαδικασία πρέπει να επαναλαμβάνεται μετά από κάθε αλλαγή που κάνουμε στον σχεδιασμό. Εκτελούμε την εντολή *Processing>Start Compilation* οπότε ξεκινά η διαδικασία της μετάφρασης. Στο παράθυρο αναφορών πρέπει τελικά να εμφανιστεί το μήνυμα "Full Compilation was Successful" οπότε διαπιστώνουμε ότι δεν έχουμε κάνει κάποιο λάθος. Με την εντολή *Processing>Compilation Report* μπορούμε να δούμε τις λεπτομέρειες της σύνθεσης.

Το επόμενο βήμα είναι να επιβεβαιώσουμε ότι η σχεδίαση μας λειτουργεί με τον τρόπο που επιθυμούμε. Για τον λόγο αυτό θα εκτελέσουμε εξομοίωση κατά την οποία θα τροφοδοτήσουμε με

διανύσματα εισόδου το κύκλωμα και θα παρατηρήσουμε τις κυματομορφές εξόδου. Υπάρχουν δύο είδη εξομοιώσεων που γίνονται με χρήση διανυσμάτων. Η λογική εξομοίωση εξασφαλίζει ότι το κύκλωμα εκτελεί την λογική λειτουργία για την οποία έχει σχεδιαστεί αλλά δεν λαμβάνει υπόψη τις εσωτερικές καθυστερήσεις. Εκτελείται γρήγορα αλλά δεν εξασφαλίζει ότι τελικά το κύκλωμα όταν υλοποιηθεί θα λειτουργήσει σωστά. Το δεύτερο είδος της εξομοίωσης είναι η χρονική εξομοίωση η οποία προσεγγίζει κατά πολύ την πραγματική συμπεριφορά του κυκλώματος. Για να γίνει αυτό εφικτό απαιτείται να γνωρίζουμε την πλήρη δομή του κυκλώματος όπως τελικά θα υλοποιηθεί στην προγραμματιζόμενη συσκευή.



Εικόνα 6. Δημιουργία αρχείου κυματομορφών



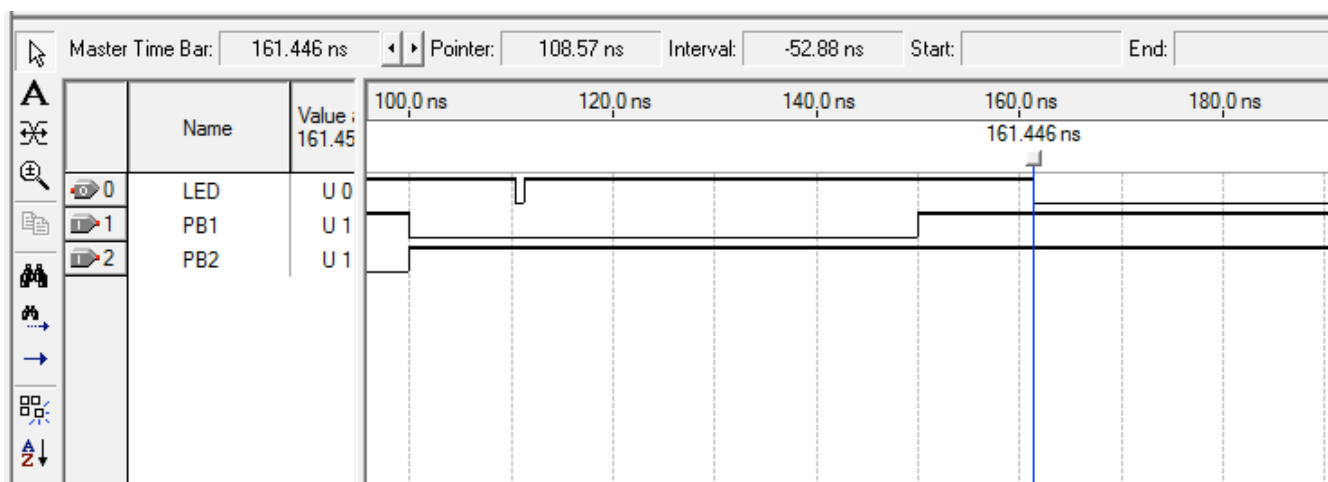
Εικόνα 7. Επιλογή σημάτων

Για να εξομοιώσουμε χρονικά το κύκλωμα απαιτείται ένα αρχείο κυματομορφών οι οποίες θα τροφοδοτήσουν τις εισόδους του κυκλώματος. Εκτελούμε την εντολή *File>New*, επιλέγουμε την καρτέλα *Other Files* και συγκεκριμένα την επιλογή *Vector Waveform File*. Έτσι εμφανίζεται το παράθυρο καθορισμού των κυματομορφών που είναι αρχικά κενό (Εικόνα 6). Κάνοντας διπλό κλικ με το ποντίκι σε κάποιο σημείο στην στήλη *Name*, εμφανίζεται το παράθυρο εισαγωγής κυματομορφών. Επιλέγουμε το *Node Finder* οπότε εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 7. Επιλέγοντας στο πεδίο *Filter* την επιλογή *Pins: Assigned* και κατόπιν πιέζοντας το *List* εμφανίζονται όλες οι θύρες. Με το πλήκτρο *>>* επιλέγονται όλες οι θύρες μαζί. Με παρόμοιο τρόπο μπορούμε να επιλέξουμε και εσωτερικούς κόμβους του κυκλώματος προκειμένου να παρακολουθήσουμε την λειτουργία τους. Πιέζουμε *OK* και πάλι *OK* προκειμένου να επιστρέψουμε στο παράθυρο εξομοίωσης.

Μία εξομοίωση απαιτεί εξωτερικά δεδομένα για να ελέγξει το κύκλωμα. Αφού οι θύρες PB1 και PB2 δεν έχουν ακόμη τιμή ο εξομοιωτής τις θέτει στο 0. Η τιμή X που εμφανίζεται στην έξοδο LED δείχνει ότι δεν έχει εκτελεστεί ακόμη εξομοίωση. Εάν μετά την εξομοίωση εξακολουθεί η έξοδος να έχει την τιμή X τότε ο εξομοιωτής δεν κατάφερε να αποδώσει τιμή στην έξοδο. Αυτό μπορεί να συμβαίνει γιατί σε κάποιο σημείο του κυκλώματος υπάρχει κάποιος κόμβος «στον αέρα» ή απλά κάποια είσοδος δεν έχει πάρει τιμή ή ακόμη κάποιο πιθανό βραχυκύκλωμα. Σε τέτοια περίπτωση πρέπει **απαραίτητα** να εντοπίσετε την αιτία του λάθους.

Με δεξιό κλικ πάνω στο PB1 επιλέγουμε *Value>Count Value*. Με την επιλογή αυτή θα δώσουμε αυτόματα τιμές μέτρησης (Count) στην είσοδο PB1 για όσο διάστημα έχουμε ορίσει ότι θα εκτελεστεί

η εξομοίωση. Επιλέγουμε την καρτέλα *Timing*. Στο πεδίο *Count Every* υπάρχει η τιμή 10.0ns η οποία σημαίνει ότι η τιμή της εισόδου PB1 θα αλλάζει σύμφωνα με την ακολουθία μέτρησης κάθε 10ns. Αλλάζουμε την τιμή αυτή σε 50ns. Στην περίπτωση αυτή πρόκειται για μέτρηση 1 bit οπότε το σήμα θα παίρνει διαδοχικά τις τιμές 0,1,0,1,0... κάθε 50ns. Στην περίπτωση ενός διαύλου 8 δυαδικών ψηφίων θα είχαμε τις τιμές 0, 1, 2, 3, ..., 255 κάθε 50ns. Καθορίζουμε τιμές για το σήμα PB2 με τον ίδιο τρόπο αλλά φροντίζουμε να αλλάζει τιμές με την μισή συχνότητα από ότι το PB1 (δηλαδή διπλή περίοδο, άρα θέτουμε καθυστέρηση 100ns). Με τον τρόπο αυτό θα δοθούν όλες οι πιθανές τιμές στις εισόδους του κυκλώματος στα πρώτα 200ns (επιβεβαιώστε το). Στην συγκεκριμένη περίπτωση αρκούν τα πρώτα 200ns για να δοθούν όλες οι δυνατές τιμές στις εισόδους του κυκλώματος (PB1-PB2=00, 01, 10, 11). Έτσι μία εξομοίωση που διαρκεί 200ns αρκεί για να ελέγξει την ορθότητα του κυκλώματος. Εάν ωστόσο σε κάποια άλλη περίπτωση ο χρόνος αυτός δεν αρκεί για να ελεγχθεί επαρκώς το κύκλωμα μπορούμε να τον αυξήσουμε με την εντολή *Edit>End Time*. Με την εντολή *Edit>Grid Size* μπορούμε να αλλάξουμε την ακρίβεια του πλέγματος εξομοίωσης.



Εικόνα 8.

Αποθηκεύουμε το αρχείο εισόδου της εξομοίωσης (*File>Save*) και εκτελούμε την εξομοίωση με την εντολή *Processing>Start Simulation*. Όταν ολοκληρωθεί η εξομοίωση εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 8. Μπορούμε να κάνουμε *Zoom In/Zoom Out* για να εστιάσουμε στο σημείο που επιθυμούμε. Παρατηρούμε ότι από την αλλαγή της εισόδου PB1 0→1 περνάει χρόνος περίπου 11.4ns (τον δείκτη χρόνου μπορούμε να τον εισάγουμε με δεξιό κλικ και κατόπιν με την εντολή *Insert Time Bar*). Αυτό είναι αναμενόμενο σε ένα κύκλωμα καθώς υπάρχει χρονική καθυστέρηση για τον υπολογισμό της εξόδου. Στην συγκεκριμένη περίπτωση η χρονική εξομοίωση μοντελοποιεί την καθυστέρηση αυτή οπότε παρατηρούμε ότι η εξομοίωση δίνει την αντίστοιχη πληροφορία.

Αυτή η εξομοίωση είναι απαραίτητη, ειδικά όταν πρόκειται να προγραμματίσουμε μία συσκευή. Ωστόσο πολλές φορές είναι ιδιαίτερα χρονοβόρα. Για τον λόγο αυτό όταν θέλουμε σε πρωταρχικά στάδια να ελέγξουμε την ορθότητα του σχεδιασμού μας γρήγορα, μπορούμε να εκτελέσουμε μόνο λογική εξομοίωση εκτελώντας αρχικά την εντολή *Processing>Generate Functional Simulation Netlist* και ακόλουθα εκτελούμε *Assignments>Settings*, επιλέγοντας την κατηγορία *Simulation* και επιλέγοντας *Functional* στο *Simulation Mode*. Η υπόλοιπη διαδικασία παραμένει όμοια. Με την λογική εξομοίωση θα βρούμε εύκολα και γρήγορα τα πιο σημαντικά λάθη της σχεδίασης μας.

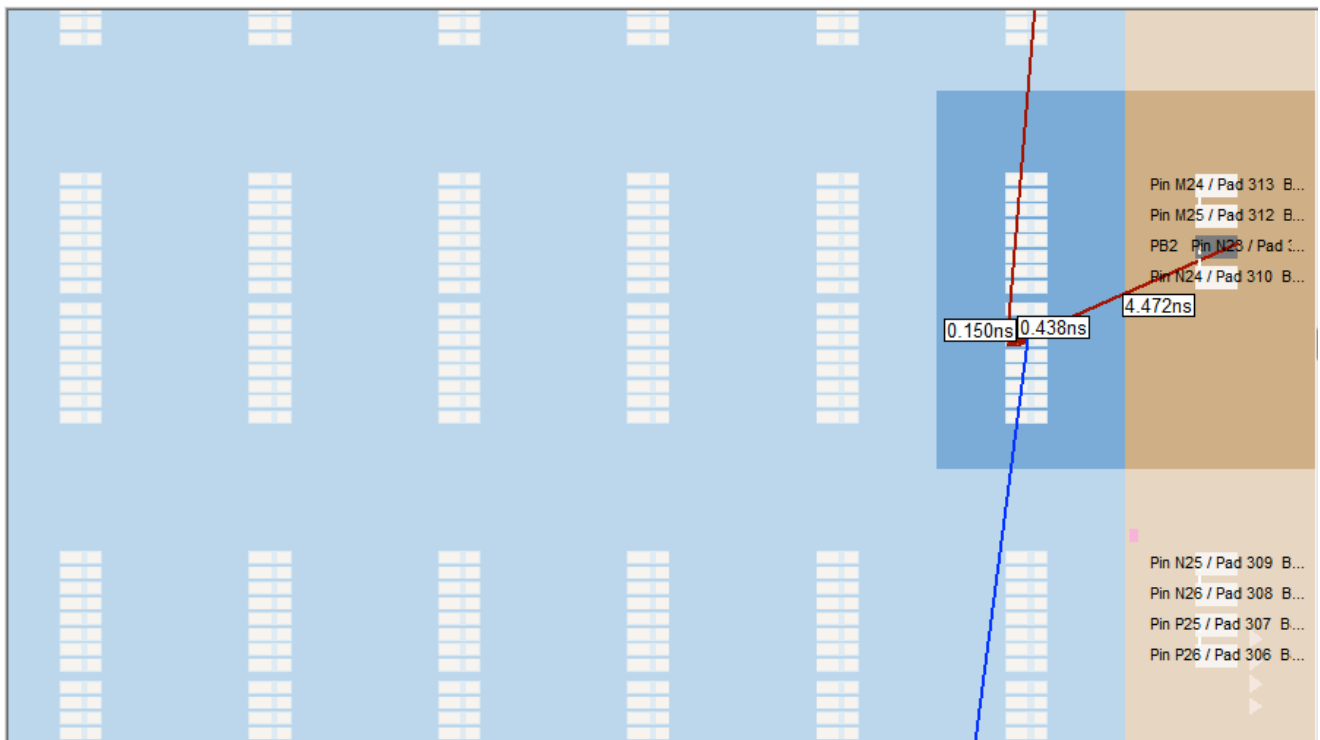
	Slack	Required P2P Time	Actual P2P Time	From	To
1	N/A	None	11.327 ns	PB1	LED
2	N/A	None	10.382 ns	PB2	LED

Εικόνα 9

Επανερχόμενοι πάλι στην χρονική εξομοίωση θα μπορούσε κάποιος να ισχυριστεί ότι επιλέξαμε τυχαία την περίοδο 50ns για το σήμα PB1 που είναι και το σήμα με την μεγαλύτερη συχνότητα του σχεδιασμού μας. Αν για παράδειγμα είχαμε επιλέξει 5ns τότε καταλαβαίνουμε ότι το κύκλωμα δεν θα λειτουργούσε σωστά αφού η έξοδος LED απαιτεί χρόνο περισσότερο από 5ns για να υπολογιστεί (περίπου 11.4ns για την περίπτωση που φαίνεται στην Εικόνα 8). Εάν το κύκλωμα απαιτούσε την ύπαρξη κάποιου ρολογιού (ακολουθιακό) τότε θα έπρεπε να είμαστε ιδιαίτερα προσεκτικοί ώστε η συχνότητα το ρολογιού να είναι μικρότερη από την μέγιστη καθυστέρηση του κυκλώματος. Για να μπορέσουμε να υπολογίσουμε την μέγιστη αυτή καθυστέρηση εκτελούμε στατική ανάλυση. Εκτελούμε την εντολή *Processing>Classic Timing Analyzer Tool* οπότε εμφανίζεται το βασικό παράθυρο της στατικής χρονικής ανάλυσης. Πιέζουμε το πλήκτρο *Start* ώστε να ξεκινήσει η εξομοίωση και αφού ολοκληρωθεί επιλέγουμε την καρτέλα *tpd* (propagation delay time) η οποία φαίνεται στην Εικόνα 9. Παρατηρούμε ότι από την είσοδο PB2 έως το LED απαιτείται χρόνος 10.382ns ενώ από την είσοδο PB1 έως το LED απαιτείται χρόνος 11.327 ns. Άρα η μικρότερη περίοδος σήματος πρέπει να είναι μεγαλύτερη από 11.327 και για ασφάλεια υπολογίζουμε και μία επιπλέον καθυστέρηση 20% (δηλ. 14 ns).

Floorplan Editor

Έχοντας πλέον ελέγξει και την ορθότητα του σχεδιασμού μας μπορούμε να δούμε οπτικά πως περίπου θα φτιαχτεί το κύκλωμα εσωτερικά στην προγραμματιζόμενη συσκευή. Πολλές φορές όταν υπάρχει η ανάλογη εμπειρία μπορούμε να αλλάξουμε και την εσωτερική τοπολογία του κυκλώματος επιδιώκοντας βελτιστοποιημένο αποτέλεσμα. Εκτελούμε *Assignments>Back Annotate Assignments*, πιέζουμε OK και κατόπιν εκτελούμε *Tools>Chip Planner*. Στο παράθυρο που ανοίγει αναζητούμε το μπλε Block και στην συνέχεια μεγεθύνουμε το σχηματικό διάγραμμα έως ότου εμφανιστεί το καφέ LE. Μεγεθύνουμε τόσο ώστε να είναι ορατές οι ετικέτες των πλαϊνών pins. Επιλέγουμε το καφέ LE και εκτελούμε *View>Generate Fan-In Connections* και κατόπιν *View>Generate Fan-out Connections*. Ακολουθήστε τα paths στο σχηματικό για να εντοπίσετε τα pins εισόδου και εξόδου. Εντοπίστε τις καθυστερήσεις στα paths. Θα εμφανιστεί η Εικόνα 10. Παρατηρούμε ότι υπάρχει αρκετός ελεύθερος χώρος αφού εμείς χρησιμοποιήσαμε μόνο ένα από τα 33.216 LE. Μπορούμε να αλλάξουμε την θέση του LE καθώς και των ακροδεκτών επηρεάζοντας φυσικά και την χρονική συμπεριφορά του κυκλώματος.



Εικόνα 10

Προγραμματισμός συσκευής (Στο Εργαστήριο)

Είμαστε πλέον έτοιμοι να προγραμματίσουμε την συσκευή και να επιβεβαιώσουμε ότι ο σχεδιασμός είναι σωστός. Αρχικά πρέπει να συνδέσουμε την πλακέτα στον υπολογιστή μας. Χρησιμοποιούμε το καλώδιο USB το οποίο συνδέουμε στην USB θύρα του υπολογιστή. Στην κάρτα το καλώδιο USB συνδέεται πάνω στον connector USB_Blaster που βρίσκεται στην πάνω αριστερή γωνία δίπλα από την τροφοδοσία. Κατόπιν συνδέουμε το τροφοδοτικό στην πρίζα αφού πρώτα τοποθετήσουμε την άλλη άκρη του στην υποδοχή στα δεξιά της κάρτας. Πιέστε τον διακόπτη τροφοδοσίας (κόκκινος διακόπτης πάνω αριστερά).

Εκτελούμε την εντολή *Tools>Programmer* και επιλέγουμε *Hardware Setup*. Στην επιλογή *Currently Selected Hardware* επιλέγουμε *USB-Blaster*. Το αρχείο με όνομα *.sof πρέπει να εμφανίζεται στο παράθυρο του προγραμματιστή. Επιλέγουμε το *Program/Configure* check box και πιέζουμε *Start*. Εάν η διαδικασία ολοκληρωθεί σωστά έχουμε προγραμματίσει την συσκευή. Για να επιβεβαιώσουμε ότι λειτουργεί σωστά το κύκλωμα πιέζουμε τα pushbuttons KEY0 ή KEY1 οπότε πρέπει το LED_G0 (κάτω δεξιά) να ανάψει.

Μέρος 2^ο: Σχεδίαση Βασικών Κυκλωμάτων

Έχοντας πλέον περιηγηθεί στο περιβάλλον του Quartus θα εξοικειωθούμε σταδιακά με την χρήση του και ταυτόχρονα θα επαναλάβουμε βασικές συνδυαστικές και ακολουθιακές δομικές μονάδες της ύλης της Ψηφιακής Σχεδίασης Ι. Μπορείτε να ανατρέξετε σε βιβλία ψηφιακής σχεδίασης καθώς και στις διαφάνειες του μαθήματος για να βρείτε τις απαραίτητες πληροφορίες. Αν και δεν απαιτείται να επιβεβαιώσετε την λειτουργία των κυκλωμάτων στο board θα πρέπει να επιβεβαιώσετε όλες τις σχεδιάσεις με χρονική εξομοίωση και να σχηματίσετε αναφορά με τις απαραίτητες κυματομορφές που θα το αποδεικνύουν.

Μέρος 1^ο. Βασικά Συνδυαστικά Κυκλώματα

Ερώτημα 1^ο

Πρώτο τμήμα της άσκησης είναι η σχεδίαση και εξομοίωση ενός αποκωδικοποιητή 2 σε 4. Ο αποκωδικοποιητής 2 σε 4 είναι ένα ψηφιακό κύκλωμα με δύο εισόδους A_1A_0 και τέσσερις εξόδους $D_0D_1D_2D_3$. Ο παρακάτω πίνακας δίνει την λειτουργία του αποκωδικοποιητή.

A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Αρχικά σχεδιάστε έναν αποκωδικοποιητή 2 σε 4 χρησιμοποιώντας λογικές πύλες. Ακολουθώντας την διαδικασία σχεδίασης της 1^{ης} εργαστηριακής άσκησης καθώς και την διαδικασία εξομοίωσης αποδείξτε ότι ο αποκωδικοποιητής σας λειτουργεί σωστά. Συμπεριλάβετε την κυματομορφή στην αναφορά που θα παραδώσετε. Εκτελέστε στατική χρονική ανάλυση και βρείτε την μέγιστη καθυστέρηση του αποκωδικοποιητή. Δείξτε την καθυστέρηση αυτή πάνω στην κυματομορφή της αναφοράς χρησιμοποιώντας τις μπάρες χρόνου (time bars).

Ερώτημα 2ο

Δεύτερο τμήμα της άσκησης είναι η σχεδίαση και εξομοίωση ενός πολυπλέκτη 4 σε 1. Ο πολυπλέκτης 4 σε 1 είναι ένα ψηφιακό κύκλωμα με τέσσερις εισόδους δεδομένων $I_3I_2I_1I_0$, δύο εισόδους επιλογής A_1A_0 , και μία έξοδο Y . Ο παρακάτω πίνακας δίνει την λειτουργία του πολυπλέκτη.

A1	A0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Αρχικά σχεδιάστε έναν πολυπλέκτη 4 σε 1 χρησιμοποιώντας λογικές πύλες. Ακολουθώντας την διαδικασία σχεδίασης της 1^{ης} εργαστηριακής άσκησης καθώς και την διαδικασία εξομοίωσης αποδείξτε ότι ο πολυπλέκτης σας λειτουργεί σωστά. Συμπεριλάβετε την κυματομορφή στην αναφορά που θα παραδώσετε. Εκτελέστε στατική χρονική ανάλυση και βρείτε την μέγιστη καθυστέρηση του πολυπλέκτη. Δείξτε την καθυστέρηση αυτή πάνω στην κυματομορφή της αναφοράς χρησιμοποιώντας τις μπάρες χρόνου (time bars).

Ερώτημα 3ο

Τρίτο τμήμα της άσκησης είναι η σχεδίαση και εξομοίωση ενός πλήρους αθροιστή. Ο πλήρης αθροιστής είναι ένα ψηφιακό κύκλωμα με τρεις εισόδους $I_2I_1I_0$, και δύο εξόδους F_1, F_0 . Ο πλήρης αθροιστής υπολογίζει τον αριθμό των μονάδων στις εισόδους $I_2I_1I_0$, και τον παρέχει σε δυαδική μορφή. Ο παρακάτω πίνακας δίνει την λειτουργία του πλήρους αθροιστή.

I2	I1	I0	F1	F0
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

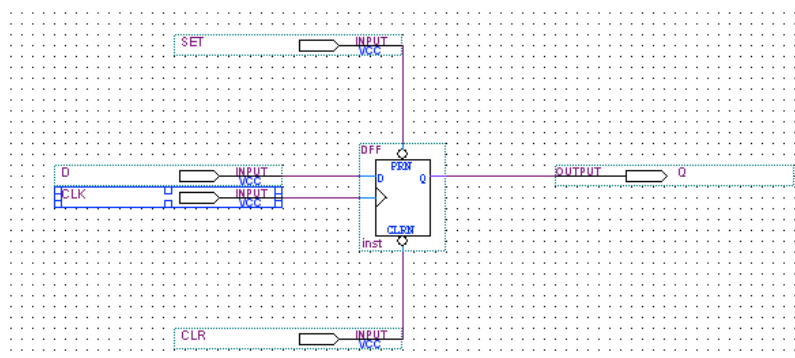
Εφαρμόστε την διαδικασία της 1^{ης} εργαστηριακής άσκησης για να σχεδιάσετε έναν πλήρη αθροιστή των τριών δυαδικών ψηφίων. Χρησιμοποιήστε για εισόδους τα SW0-2. Προσέξτε ότι οι διακόπτες στο ON (πάνω) παρέχουν την λογική τιμή 1. Για το άθροισμα χρησιμοποιήστε το LED G0 και για το κρατούμενο το LED G1 (θα ανάβουν όταν οι αντίστοιχες τιμές είναι 1). Επιβεβαιώστε την

σχεδίαση και δείξτε στον επιτηρητή τα αποτελέσματα όλης της διαδικασίας σχεδίασης (κυματομορφές, χρονική ανάλυση κλπ). Προγραμματίστε την κάρτα και επιβεβαιώστε ότι λειτουργεί σωστά. Συμπεριλάβετε στην αναφορά όλα τα απαραίτητα σχήματα (σχεδίαση, αποτελέσματα εξομοίωσης και στατικής ανάλυσης κλπ).

Μέρος 2^ο. Βασικά Ακολουθιακά Κυκλώματα

Ερώτημα 1ο

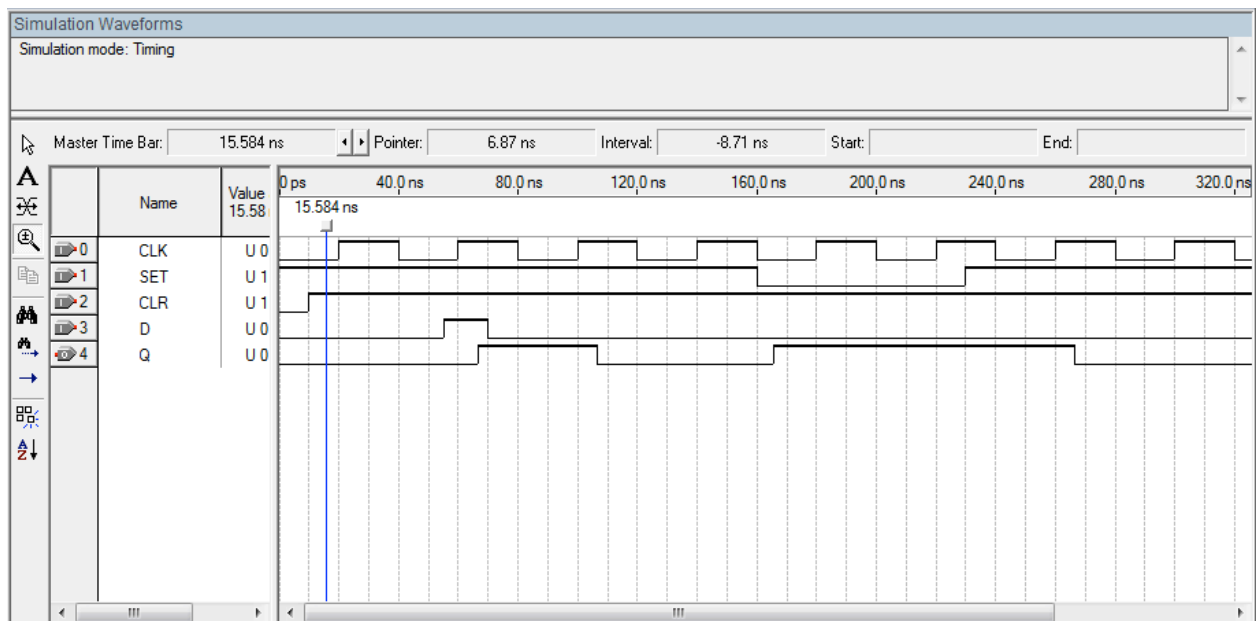
Στο πρώτο τμήμα της άσκησης θα σχεδιάσουμε τα βασικά flip flop. Με την γνωστή διαδικασία σχεδίασης δημιουργήστε ένα νέο σχηματικό στο οποίο θα τοποθετήσετε ένα D flip flop. Το D-flip flop θα το βρείτε στην βιβλιοθήκη Quartus50sp1/libraries>Primitives>Dff και θα το τοποθετήσετε στο σχηματικό με τον γνωστό τρόπο. Συγκεκριμένα δημιουργήστε το σχηματικό που φαίνεται στο ακόλουθο σχήμα:



Εικόνα 11. Σχηματικό με D flip-flop

Παρατηρήστε στην Εικόνα 11 ότι εκτός των σημάτων D, CLK και Q τα οποία πρέπει να είναι γνωστά σε εσάς από την ψηφιακή σχεδίαση, υπάρχουν δύο ακόμη σήματα, τα SET, CLR. Με το σήμα SET το οποίο ενεργοποιείται στο 0 (δείτε το κυκλάκι στην αντίστοιχη είσοδο PRN του D flip flop) το flip flop (και όλα τα flip flop) τίθεται στο λογικό 1. Αντίστοιχα με το σήμα CLR (CLRΝ) τίθεται στο μηδέν. Η λειτουργία των σημάτων αυτών είναι ασύγχρονη, δηλαδή δεν σχετίζεται με το ρολόι του συστήματος.

Κατόπιν θα θυμηθούμε την λειτουργία του D flip flop εξομοιώνοντας την λειτουργία του. Για τον λόγο αυτό εισάγουμε την κυματομορφή που φαίνεται στο ακόλουθο σχήμα:

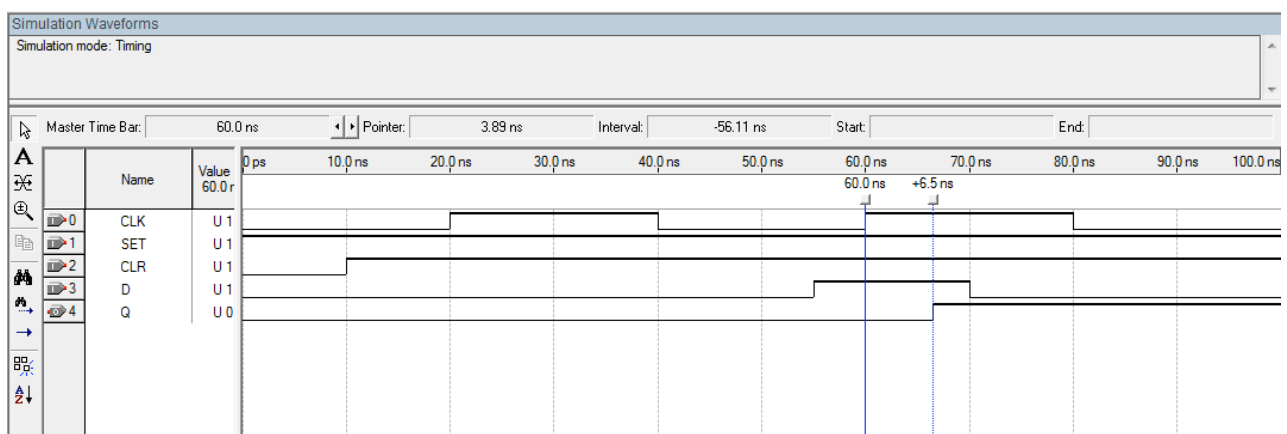


Εικόνα 12

Παρατηρήστε τις κυματομορφές στην Εικόνα 12. Αρχικά μπορούμε να δούμε το ρολόι του συστήματος το οποίο είναι ένα περιοδικό σήμα με περίοδο 40ns και duty cycle 50% (δηλαδή 50% της περιόδου είναι στην μονάδα). Ένα τέτοιο σήμα μπορείτε εύκολα να δημιουργήσετε με τις οδηγίες που έχουν δοθεί στην πρώτη άσκηση. Το δεύτερο σήμα είναι το σήμα θέσης (SET) το οποίο αρχικά είναι απενεργοποιημένο (1) και στην χρονική στιγμή 160ns ενεργοποιείται (0) μέχρι την χρονική στιγμή 230ns. Για να το επιτύχουμε αυτό ακολουθούμε τα εξής βήματα:

1. Θέτουμε το σήμα SET στην μονάδα (με δεξί κλικ πάνω στο σήμα επιλέγουμε Value>Forcing High 1).
2. Με το ποντίκι επιλέγουμε την περιοχή του σήματος (160ns-230ns) και την θέτουμε με αντίστοιχο τρόπο στο μηδέν.

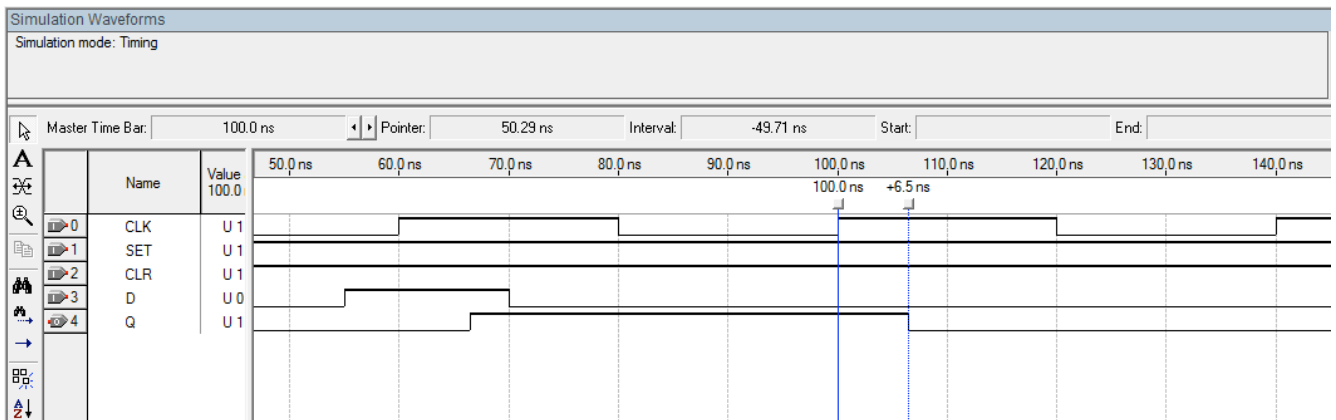
Με όμοιο τρόπο θέτουμε και τα σήματα CLR, D στις τιμές που φαίνονται στο παραπάνω σχήμα. Κατόπιν εκτελούμε εξομοίωση και παίρνουμε τα αποτελέσματα που φαίνονται στο ακόλουθο σχήμα (για τα πρώτα 100ns):



Εικόνα 13

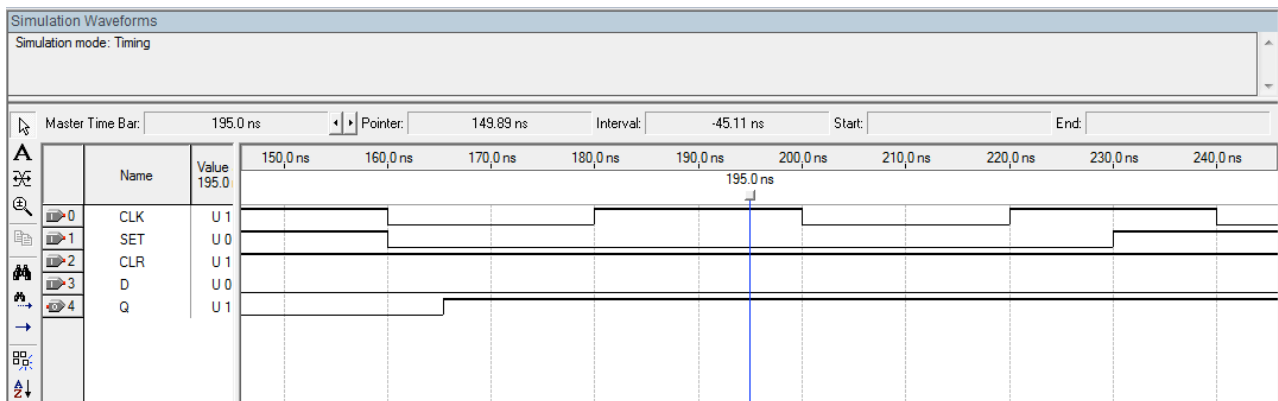
Παρατηρήστε ότι από την άνοδο του ρολογιού CLK την χρονική στιγμή 60.0ns μέχρι την άνοδο του Q που ακολουθεί την θετική ακμή του ρολογιού περνάνε 6.5ns τα οποία είναι ουσιαστικά η καθυστέρηση που εισάγει το flip flop. Παρατηρήστε ότι το σήμα D πάει στο λογικό 1 αρκετά πιο πριν (περίπου στα 55ns) αλλά αυτό δεν επηρεάζει την έξοδο καθώς η θετική ακμή του ρολογιού είναι αυτή

που καθορίζει την αποθήκευση στο flip flop. Η έξοδος του flip flop παραμένει στο λογικό 1 μέχρι την επόμενη θετική ακμή του ρολογιού στην χρονική στιγμή 100ns. Τότε η λογική τιμή 0 στην είσοδο D αποθηκεύεται στο flip flop και εμφανίζεται στην έξοδο του 6.5ns αργότερα όπως φαίνεται στο ακόλουθο στιγμιότυπο.



Εικόνα 14

Τέλος στην Εικόνα 15 μπορούμε να παρατηρήσουμε την λειτουργία του ασύγχρονου σήματος SET το οποίο ενεργοποιείται την χρονική στιγμή 160ns και διατηρείται ενεργοποιημένο μέχρι την χρονική στιγμή 230ns. Παρατηρήστε ότι η έξοδος Q μεταβαίνει στην τιμή 1 την χρονική στιγμή 165.528ns (επιβεβαιώστε το με time bar). Αυτή είναι η καθυστέρηση ασύγχρονης θέσης. Παρατηρήστε επίσης ότι αυτή η μετάβαση γίνεται πριν την επόμενη θετική ακμή του ρολογιού η οποία συμβαίνει στα 180ns. Επιπλέον παρατηρήστε ότι η θετική ακμή του ρολογιού την χρονική στιγμή 180ns δεν προκαλεί την αποθήκευση της τιμής '0' που υπάρχει στην είσοδο D. Η σύγχρονη θέση έχει υψηλότερη προτεραιότητα οπότε υπερισχύει. Άρα κατανοούμε ότι η ασύγχρονη θέση είναι ανεξάρτητη του ρολογιού και για αυτό ονομάζεται και ασύγχρονη. Όμοια συμπεριφορά θα παρατηρήσουμε εάν ενεργοποιήσουμε την ασύγχρονη μηδένιση.



Εικόνα 15. Λειτουργία ασύγχρονου σήματος SET

Ερώτημα 2ο

Στο δεύτερο μέρος της άσκησης θα επιβεβαιώσετε την λειτουργία των flip flop JK και T. Για κάθε ένα από αυτά θα επαναλάβετε την παραπάνω ανάλυση και θα σχηματίσετε αναφορά με τα διαγράμματα χρονισμού και την αντίστοιχη ανάλυση. Επιβεβαιώστε στην αναφορά ασύγχρονη θέση και μηδένιση καθώς και την λειτουργία όλων των συνδυασμών εισόδων των flip flops.

Ερώτημα 3ο

Στο τρίτο μέρος θα χρησιμοποιήσουμε αυτά που μάθαμε στις προηγούμενες ασκήσεις για να σχεδιάσουμε έναν καταχωρητή τεσσάρων δυαδικών ψηφίων από D flip flop με τέσσερις εισόδους

δεδομένων I_3, I_2, I_1, I_0 , τέσσερις εξόδους Q_3, Q_2, Q_1, Q_0 , και δύο εισόδους επιλογής λειτουργίας S_1, S_0 . Ο παρακάτω πίνακας καθορίζει την ακριβή λειτουργία του καταχωρητή:

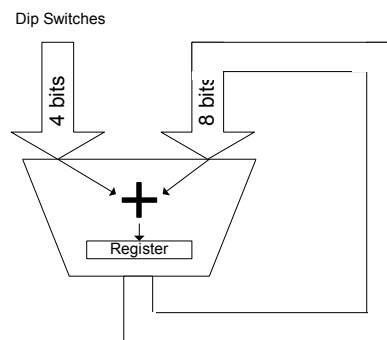
S_1	S_0	$Q_{3...0}$
0	0	0...0
0	1	$I_{3...0}$
1	0	$(Q_{3...0})'$
1	1	1...1

Χρησιμοποιήστε τέσσερις πολυπλέκτες 4 σε 1 σύμφωνα με τις μεθόδους σχεδίασης που ακολουθήσαμε στην Ψηφιακή Σχεδίαση. Σχεδιάστε μόνοι σας έναν πολυπλέκτη 4 σε 1 και αντιγράψτε τον με copy-paste τέσσερις φορές. Παρακάτω θα μάθουμε πιο αποδοτικούς τρόπους να εισάγουμε επαναλαμβανόμενες δομές και θα δούμε πως μπορούμε να χρησιμοποιήσουμε έτοιμες τέτοιες δομές που παρέχει το περιβάλλον της Altera. Επιβεβαιώστε την λειτουργία του καταχωρητή με τις κατάλληλες εξομοιώσεις και σχηματίστε την αναφορά με την σχεδίαση και τα αποτελέσματα των εξομοιώσεων.



Εργαστηριακή Άσκηση 2: Ιεραρχική σχεδίαση και προσχεδιασμένοι πυρήνες

Στην 2^η εργαστηριακή άσκηση θα ασχοληθούμε με την ιεραρχική σχεδίαση. Συγκεκριμένα θα μάθουμε να σχεδιάζουμε απλές οντότητες τις οποίες θα χρησιμοποιούμε κατόπιν ιεραρχικά προκειμένου να σχεδιάσουμε μεγαλύτερες οντότητες ιεραρχικά υψηλότερες. Έτσι θα αντιμετωπίσουμε προβλήματα όπως αυτό που συναντήσαμε στην προηγούμενη άσκηση κατά το οποίο έπρεπε να αντιγράψουμε το κύκλωμα ενός πολυπλέκτη για να το χρησιμοποιήσουμε πολλαπλές φορές. Επιπλέον θα δούμε και μερικές προσχεδιασμένες οντότητες τις οποίες συνήθως ενσωματώνουμε στους σχεδιασμούς που γίνονται πάνω στο board και βοηθούν στην αξιοποίηση των διαφόρων περιφερειακών συσκευών της (πχ. LCD display, VGA οθόνη, ποντίκι, πληκτρολόγιο κλπ). Η 4^η εργαστηριακή άσκηση δεν απαιτεί την συγγραφή αναφοράς.



Εικόνα 16. Συσσωρευτής

Ζητούμενο Κύκλωμα

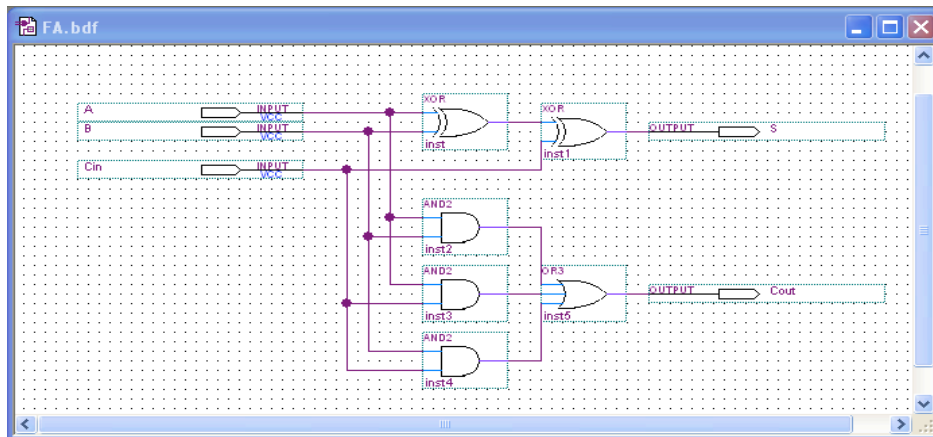
Θα σχεδιάσουμε έναν συσσωρευτή των 8 δυαδικών ψηφίων όπως φαίνεται στην Εικόνα 16. Ο ζητούμενος συσσωρευτής αποτελείται από έναν αθροιστή και έναν καταχωρητή των 8 δυαδικών ψηφίων. Παρατηρούμε ότι οι δύο εισοδοί του αθροιστή δεν έχουν το ίδιο μέγεθος. Η μία είσοδος αποτελείται από 4 δυαδικά ψηφία και θα συνδεθεί στα Dip Switches SW0-3 της πλακέτας. Η δεύτερη είσοδος αποτελείται από 8 δυαδικά ψηφία και είναι στην ουσία η τιμή που έχει αποθηκευτεί στον καταχωρητή. Έτσι ουσιαστικά το κύκλωμα αυτό προσθέτει κάθε φορά στον καταχωρητή την τιμή από τα Dip Switches. Αν υποθέσουμε για παράδειγμα ότι η τιμή στα Dip Switches είναι 0010 (2_{10}) τότε οι τιμές που θα αποθηκευτούν στον καταχωρητή είναι οι ακόλουθες: 00, 02, 04, 06, 08, 0A, 0C, ... (στο δεκαεξαδικό σύστημα). Θεωρήστε ότι υπάρχει κρατούμενο εισόδου το οποίο όμως στα πλαίσια της παρούσας άσκησης θα συνδεθεί μόνιμα στο 0.

Σχεδίαση συσσωρευτή

Οι βασικές δομικές μονάδες ενός συσσωρευτή είναι οι πλήρεις αθροιστές/ημιαθροιστές και τα στοιχεία μνήμης (flip flop). Εφόσον ο αθροιστής πρέπει να προσθέτει 4 δυαδικά ψηφία με 8 δυαδικά ψηφία, και επιπλέον έχει κρατούμενο εισόδου, καταλαβαίνουμε ότι απαιτούνται 4 πλήρεις αθροιστές και 4 ημιαθροιστές. Οι πλήρεις αθροιστές υπολογίζουν τα 4 λιγότερο σημαντικά δυαδικά ψηφία του αθροίσματος ενώ οι ημιαθροιστές υπολογίζουν τα 4 περισσότερο σημαντικά δυαδικά ψηφία του αθροίσματος. Προσέξτε ότι τα 4 περισσότερο σημαντικά δυαδικά ψηφία του αθροίσματος προκύπτουν από την άθροιση του κρατουμένου που προκύπτει από την πρόσθεση των 4 λιγότερο σημαντικών ψηφίων με τα 4 πιο σημαντικά δυαδικά ψηφία του καταχωρητή (για αυτό χρειαζόμαστε ημιαθροιστές και όχι αθροιστές). Από την μεριά του ο καταχωρητής αποτελείται από 8 στοιχεία μνήμης flip flop.

Πρώτο βήμα της σχεδίασης είναι η δημιουργία των δύο βασικών κυττάρων, του πλήρη αθροιστή και του ημιαθροιστή. Στην πραγματικότητα το Quartus II παρέχει έτοιμο τόσο τον καταχωρητή, όσο και τον αθροιστή των 8 bits. Ακόμη περισσότερο παρέχει έτοιμο συσσωρευτή σαν αυτόν που ζητάμε.

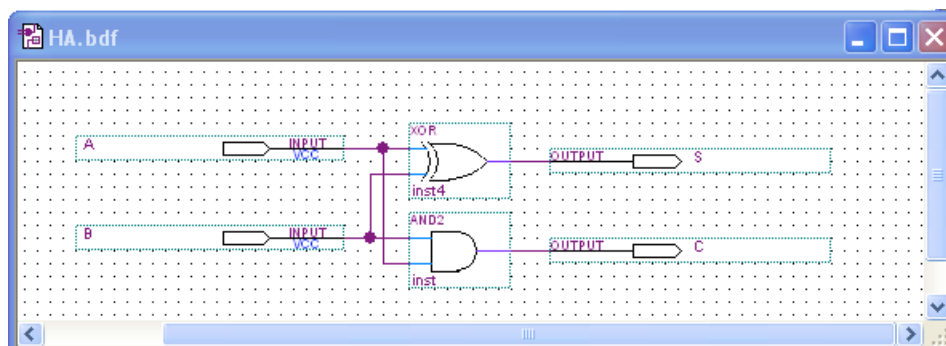
Ωστόσο για εκπαιδευτικούς λόγους θα σχεδιάσουμε μόνοι μας τις βασικές δομικές μονάδες ώστε να κατανοήσουμε την διαδικασία της ιεραρχικής σχεδίασης.



Εικόνα 17. Πλήρης Αθροιστής

Δημιουργούμε ένα νέο project κάνοντας τις αρχικές ρυθμίσεις που έχουμε αναφέρει στην 1^η εργαστηριακή άσκηση. Ανοίγουμε ένα νέο φύλλο σχηματικού και σχεδιάζουμε τον πλήρη αθροιστή όπως φαίνεται στην Εικόνα 17. Ονομάζουμε το σχηματικό FA (Full Adder) και ξεκινάμε την διαδικασία της σύνθεσης (compilation). Εμφανίζεται μήνυμα λάθους και η διαδικασία σταματά. Ο λόγος που συμβαίνει αυτό είναι γιατί το Quartus II περιμένει η υψηλότερη σε ιεραρχία μονάδα να έχει το όνομα που δώσαμε στο project. Για τον λόγο αυτό εάν θέλουμε να ελέγξουμε τμηματικά τον σχεδιασμό θα πρέπει να ορίζουμε κάθε φορά ως κορυφαία οντότητα το τμήμα που επιθυμούμε. Έτσι εκτελούμε την εντολή *Project>Set As Top Level Entity* και κατόπιν εκτελούμε πάλι σύνθεση. Αυτή τη φορά το αποτέλεσμα είναι επιτυχημένο.

Πριν προχωρήσουμε πρέπει να βεβαιωθούμε ότι ο πλήρης αθροιστής λειτουργεί σωστά. Για αυτό εκτελούμε χρονική εξομοίωση με τον τρόπο που αναλύσαμε στην άσκηση 1. Δίνουμε όλες τις δυνατές τιμές στον αθροιστή με ελάχιστη περίοδο σήματος εισόδου 50 ns. Κατά την επιλογή των σημάτων ενδεχομένως να μην εμφανίζονται *Nodes* στην Εικόνα 7. Αυτό οφείλεται στο γεγονός ότι η επιλογή *Pins>Assigned* στο πεδίο *Filter* πλέον δεν είναι σωστή αφού δεν έχουμε κάνει καμία ανάθεση θυρών I/O σε ακροδέκτες της προγραμματιζόμενης συσκευής. Η σωστή επιλογή αυτή την φορά είναι *Pins: all* ώστε να δούμε όλες τις εισόδους/εξόδους. Αφού καθορίσουμε τις κυματομορφές εισόδου με τον τρόπο που αναλύσαμε αποθηκεύουμε το αρχείο κυματομορφών με το όνομα FA.vwf και εκτελούμε την εντολή *Tools>Simulator Tools*. Στο πεδίο *Simulation Input* δίνουμε το όνομα του αρχείου κυματομορφών και εκτελούμε εξομοίωση (δείτε τα αποτελέσματα της εξομοίωσης στον επιτηρητή σας).



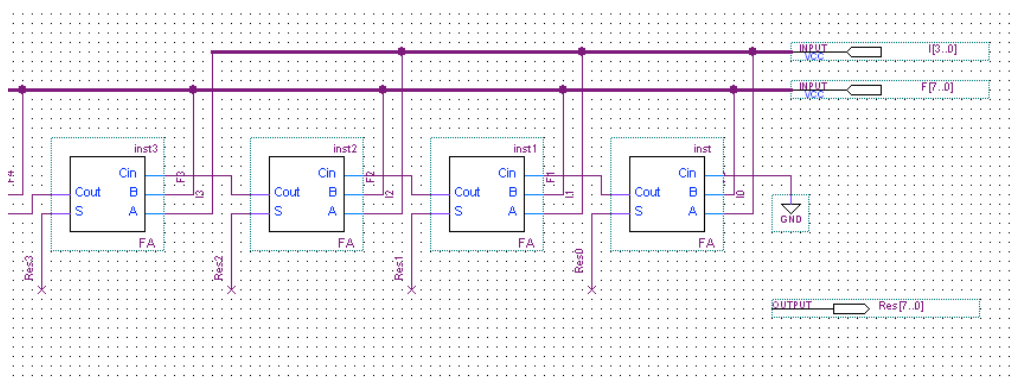
Εικόνα 18. Ημιαθροιστής

Έχοντας πλέον επιβεβαιώσει την σωστή λειτουργία του πλήρη αθροιστή μπορούμε να δημιουργήσουμε το σύμβολο του για να μπορέσουμε να τον χρησιμοποιήσουμε ιεραρχικά. Εκτελούμε την εντολή *File>Create/Update>Create Symbol Files for Current File*. Ακολουθούμε την ίδια διαδικασία

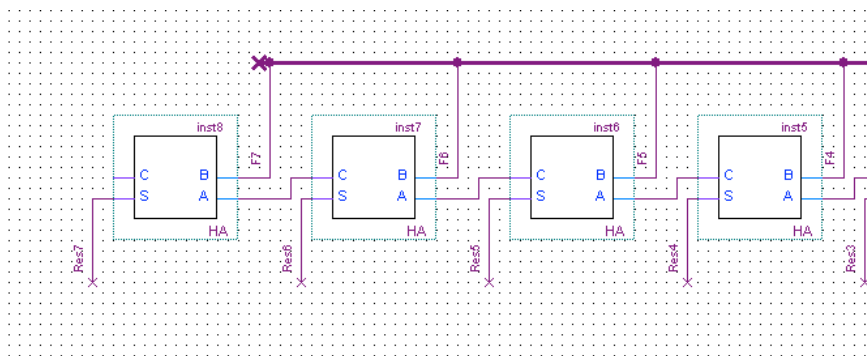
για τον ημιαθροιστή που φαίνεται στην Εικόνα 18. Επιβεβαιώνουμε την ορθότητα του και δημιουργούμε σύμβολο και για αυτόν.

Έχοντας πλέον δημιουργήσει και τα δύο βασικά κύτταρα της σχεδίασης μας μπορούμε πλέον να δημιουργήσουμε τον αθροιστή των 8 δυαδικών ψηφίων. Κλείνουμε όλα τα σχηματικά και τα παράθυρα που έχουμε ανοίξει στο project (δεν κλείνουμε το project) και δημιουργούμε ένα νέο σχηματικό με το όνομα Adder8_4bits. Εάν προσπαθήσουμε να τοποθετήσουμε κάποιο σύμβολο στην σχεδίαση θα παρατηρήσουμε ότι στο πεδίο Libraries (Εικόνα 3) εμφανίζεται επιπλέον και η βιβλιοθήκη Project και αν την επεκτείνουμε θα δούμε τα ονόματα FA, HA. Επιλέγουμε τον πλήρη αθροιστή και τον τοποθετούμε τέσσερις φορές στο σχηματικό. Οποιοδήποτε σύμβολο τοποθετείται μπορεί να περιστραφεί έτσι ώστε να διευκολύνεται η σχεδίαση. Με τον ίδιο τρόπο τοποθετούμε και τέσσερις ημιαθροιστές. Κανονικά θα πρέπει να τοποθετήσουμε 12 θύρες εισόδου (8+4 δυαδικά ψηφία δεδομένων) και 8 θύρες εξόδου (8 δυαδικά ψηφία αποτελέσματος) στον αθροιστή. Ωστόσο, μπορούμε να χρησιμοποιήσουμε δύο διαύλους εισόδου και έναν δίαυλο εξόδου για να απλοποιήσουμε την διαδικασία. Έτσι τοποθετούμε δύο θύρες εισόδου και μία θύρα εξόδου με τον γνωστό τρόπο και τις ονομάζουμε I[3..0] (για τα 4 δυαδικά ψηφία από τα Dip Switches), F[7..0] (για τα 8 δυαδικά ψηφία από τον καταχωρητή) και Res[7..0] (για τα 8 δυαδικά ψηφία προς τον καταχωρητή). Από τις θύρες αυτή την φορά δεν ξεκινούν οι απλές συνδέσεις αλλά οι συνδέσεις διαύλου (bold γραμμές από το ToolBox σχεδιασμού). Ξεκινώντας από τον δίαυλο για την θύρα I[3..0] τραβάμε μία Bold γραμμή κατά μήκος του αθροιστή χωρίς να συνδέουμε το δεύτερο άκρο της γραμμής πουθενά. Κατόπιν συνδέουμε απλές γραμμές από την είσοδο A κάθε πλήρη αθροιστή πάνω στον δίαυλο και δίνουμε τα ονόματα I[0], I[1], I[2], I[3] στην κάθε γραμμή με δεξιά κλικ του ποντικιού (*Properties*). Προσέχουμε πολύ την αντιστοιχία των ονομάτων των επιμέρους γραμμών με τον δίαυλο. Προσέχουμε επίσης να συνδέουμε το I[0] στο λιγότερο σημαντικό αθροιστή, το I[1] στον επόμενο κλπ. Η σημαντικότητα καθορίζεται από την σημαντικότητα του δυαδικού ψηφίου αθροίσματος του πλήρους αθροιστή. Επαναλαμβάνουμε το ίδιο για τις εισόδους B των αθροιστών τις οποίες θα συνδέουμε στον δίαυλο F.

Επειδή η χρήση γραμμών σε ένα σχηματικό μπορεί να είναι ιδιαίτερα περίπλοκη, το Quartus επιτρέπει και την υπονοούμενη σύνδεση χρησιμοποιώντας μόνο ονόματα. Έτσι μπορούμε να αποφύγουμε περίπλοκα σχηματικά που μάλλον θα είναι δυσανάγνωστα. Φυσικά δύο γραμμές που πρέπει να είναι βραχυκυκλωμένες πρέπει να έχουν το ίδιο όνομα και ας μην είναι ορατά συνδεδεμένες στο σχηματικό. Το ίδιο ισχύει και για συνδέσεις διαύλων. Ένα παράδειγμα φαίνεται στην Εικόνα 19 όπου ο δίαυλος Res[7..0] ο οποίος δίνει το αποτέλεσμα της πρόσθεσης φαίνεται να είναι στον αέρα αλλά στην πραγματικότητα έχει συνδεθεί ονομαστικά με τις εξόδους s των αθροιστών. Στην Εικόνα 19 φαίνεται το κύκλωμα που υπολογίζει τα 4 λιγότερο σημαντικά δυαδικά ψηφία του αθροίσματος, ενώ στην Εικόνα 20 το κύκλωμα που υπολογίζει τα 4 περισσότερα σημαντικά. Παρατηρήστε στην Εικόνα 15 ότι εάν χρησιμοποιούσαμε πλήρεις αθροιστές τότε η επιπλέον είσοδος τους θα ήταν στο 0 οπότε δεν θα έπαιζε κανένα ρόλο.

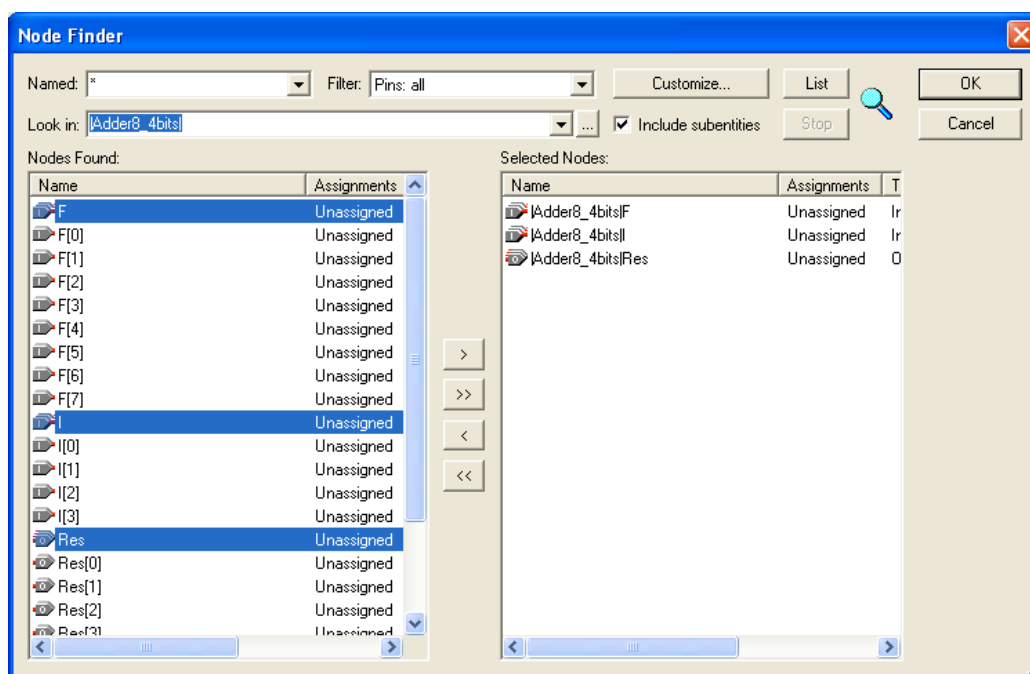


Εικόνα 19. Αθροιστής 8 bits (α)



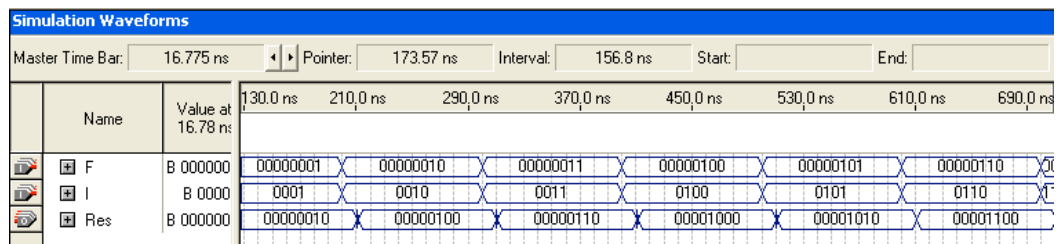
Εικόνα 20. Αθροιστής 8 bits (β)

Πριν ελέγξουμε την σχεδίαση του αθροιστή, κάνουμε διπλό κλικ σε κάποιον πλήρη αθροιστή ή ημιαθροιστή. Θα παρατηρήσουμε τότε ότι ανοίγει το σχηματικό που σχεδιάσαμε προηγουμένως. Με άλλα λόγια κατεβήκαμε ένα επίπεδο στην ιεραρχία. Κλείνουμε το σχηματικό που ανοίξαμε και εκτελούμε σύνθεση.



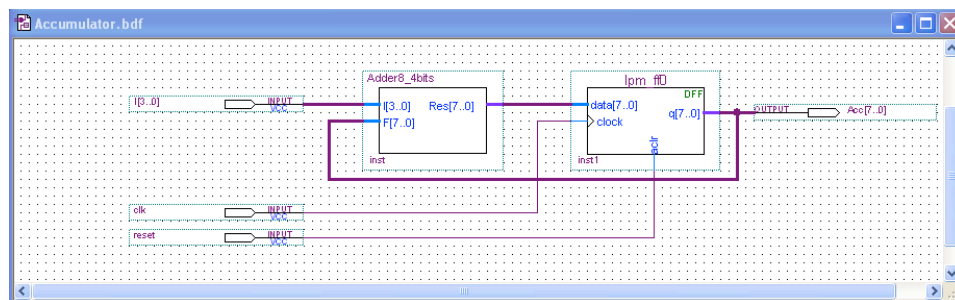
Εικόνα 21

Ελέγχουμε την ορθότητα της σχεδίασης με χρονική εξομοίωση. Κατά την εισαγωγή των σημάτων εισόδου/εξόδου δεν χρειάζεται να παρακολουθούμε $2 \times 8 + 4 = 20$ δυαδικά σήματα αλλά μπορούμε να παρακολουθούμε τους διαύλους I, F, Res σαν ομάδες σημάτων σε δυαδική, δεκαδική ή και δεκαεξαδική μορφή. Για να το πετύχουμε αυτό επιλέγουμε τις ομάδες αυτές όπως φαίνεται στην Εικόνα 21. Τα σήματα αυτή την φορά θα είναι μεταβαλλόμενα με πολύ μικρότερη συχνότητα από ότι ήταν ως τώρα αφού το κύκλωμα έχει πολύ μεγαλύτερη καθυστέρηση από έναν ημιαθροιστή ή πλήρη αθροιστή. Πριν ξεκινήσουμε την εξομοίωση δημιουργούμε ένα νέο αρχείο κυματομορφών και το ορίζουμε ως το αρχείο κυματομορφών της εξομοίωσης όπως έχουμε ήδη αναφέρει. Ένα μικρό τμήμα της εξομοίωσης φαίνεται στην Εικόνα 22. Τέλος δημιουργούμε ένα σύμβολο για τον αθροιστή που σχεδιάσαμε.



Εικόνα 22. Εξομοίωση Αθροιστή

Αυτό που απομένει είναι να εισάγουμε τον καταχωρητή του αποτελέσματος. Θα μπορούσαμε να σχεδιάσουμε τον καταχωρητή με τον ίδιο τρόπο με τον οποίο σχεδιάσαμε και τον αθροιστή. Παρόλα αυτά καλό είναι να δούμε μία από τις δυνατότητες που παρέχει το Quartus II ώστε να διευκολύνει την σχεδίαση κυκλωμάτων. Θα χρησιμοποιήσουμε μία από τις παραμετρικές συναρτήσεις (Megafunctions) που παρέχονται. Συγκεκριμένα το Quartus δίνει την δυνατότητα χρήσης διαφόρων παραμετρικών συναρτήσεων οι οποίες υλοποιούν αποθηκευτικές μονάδες (καταχωρητές διαφόρων δυνατοτήτων, μνήμες RAM, ROM, FIFO, αριθμητικές μονάδες, μονάδες I/O κλπ). Έτσι πριν αποφασίσουμε να σχεδιάσουμε μια οποιαδήποτε μονάδα, αναζητούμε στις βιβλιοθήκες πιθανή Megafunction η οποία παρέχει την συνάρτηση που θέλουμε. Σημαντικός παράγοντας είναι ότι οι Megafunctions είναι βελτιστοποιημένες για χρήση στην οικογένεια συσκευών της Altera οπότε είναι μάλλον απίθανο να καταφέρουμε να σχεδιάσουμε κάτι πιο αποδοτικό μόνοι μας.



Εικόνα 23. Συσσωρευτής

Δημιουργούμε ένα νέο σχηματικό με το όνομα Accumulator. Τοποθετούμε αρχικά τον αθροιστή των 8 δυαδικών ψηφίων στο σχηματικό χρησιμοποιώντας το σύμβολο που δημιουργήσαμε. Η Megafunction η οποία υλοποιεί τον καταχωρητή είναι η LPM_FF. Φυσικά υπάρχουν και άλλες οι οποίες όμως έχουν πρόσθετες δυνατότητες που δεν μας είναι απαραίτητες (πχ. ολίσθηση). Πριν εισάγουμε την Megafunction την αναζητούμε στην “Βοήθεια” του εργαλείου προκειμένου να κατανοήσουμε τον πίνακα αλήθειας του καταχωρητή καθώς και τον τρόπο με τον οποίο πρέπει να συνδέουμε τα διάφορα σήματα εισόδου/εξόδου. Με την γνωστή διαδικασία εισαγωγής συμβόλου επιλέγουμε *Libraries> quartus50sp1/libraries/> Megafunctions> Storage> LPM_FF* και πιέζουμε OK. Σε αντίθεση με ότι γινόταν έως τώρα παρατηρούμε ότι ξεκινά ο *MegaWizard Plug-In Manager* ο οποίος μας βοηθάει να ορίσουμε τις παραμέτρους της Megafunction. Στο πρώτο παράθυρο επιλέγουμε την μορφή εξόδου AHDL (εάν θέλουμε να επεξεργαστούμε την Megafunction σε επίπεδο σχεδίασης πρέπει να επιλέξουμε VHDL με την οποία θα ασχοληθούμε παρακάτω). Στο επόμενο παράθυρο επιλέγουμε 8 D-Flip Flops χωρίς δυνατότητα *Clock Enable Input*. Στο επόμενο παράθυρο επιλέγουμε μόνο την ασύγχρονη μηδένιση (*Asynchronous Clear*) και ολοκληρώνουμε την παραμετροποίηση. Τοποθετούμε τον καταχωρητή στο σχηματικό και προσθέτουμε θύρες εισόδου/εξόδου με ονόματα I[3..0] για την είσοδο από τα *Dip Switches* και Acc[7..0] για την έξοδο του συσσωρευτή που θα είναι ουσιαστικά η έξοδος του καταχωρητή. Κατόπιν δημιουργούμε μία είσοδο ρολογιού και μία είσοδο αρχικοποίησης. Το κύκλωμα αυτό φαίνεται στην Εικόνα 23. Εκτελούμε σύνθεση και επιβεβαιώνουμε ότι δεν υπάρχουν λάθη.

Πριν ελέγξουμε την ορθότητα της σχεδίασης με χρονική εξομοίωση, εκτελούμε στατική χρονική ανάλυση προκειμένου να βρούμε την συχνότητα ρολογιού που θα χρησιμοποιήσουμε. Η συχνότητα αυτή είναι περίπου 420.17 MHz ή εναλλακτικά η περίοδος ρολογιού πρέπει να είναι μεγαλύτερη από

περίπου 2.38 ns (βρήκατε τόσο;). Προσέξτε όμως ότι δεν έχουν συμπεριληφθεί οι καθυστερήσεις των pins οι οποίες είναι περίπου 5-6 ns. Αυτό συμβαίνει γιατί δεν έχουμε ορίσει ακόμη pins οπότε η παραγόμενη συχνότητα αφορά την λειτουργία του κυκλώματος εσωτερικά στην προγραμματιζόμενη συσκευή. Για τον λόγο αυτό επιλέγουμε περίοδο 20 ns ώστε να μην συμβεί λάθος κατά την εξομοίωση. Εκτελούμε χρονική εξομοίωση με αυτά τα δεδομένα. Θέτουμε το reset στη μονάδα για 20ns αρχικά και κατόπιν το αφήνουμε μόνιμα στο 0. Προσέχουμε ότι αυτή την φορά πρέπει να επιλέξουμε μόνο τα πρώτα 20 ns στην κυματομορφή και κατόπιν να εκτελέσουμε την εντολή Value>Invert. Μπορούμε να δοκιμάσουμε εναλλακτικούς τρόπους για να δώσουμε την τιμή που επιθυμούμε. Κατόπιν ορίζουμε το ρολόι ως περιοδικό σήμα με περίοδο 20 ns και αλλάζουμε τις τιμές στην είσοδο I[3..0] κατά την αρνητική ακμή του ρολογιού. Με αυτόν τον τρόπο θα αποφύγουμε προβλήματα χρονισμού (setup-time κατά την θετική ακμή). Επιβεβαιώνουμε ότι το κύκλωμα λειτουργεί σωστά.

Υλοποίηση στο Board

Μετά την εξομοίωση του σχεδιασμού και την επιβεβαίωση της σωστής λειτουργίας του, ακολουθεί η επιβεβαίωση του τελικού κυκλώματος στο board, σε πραγματική λειτουργία. Για να το πετύχουμε αυτό θα πρέπει να βρούμε κάποιον τρόπο ώστε μετά τον προγραμματισμό να τροφοδοτούμε το υλοποιούμενο κύκλωμα με δεδομένα και να παρακολουθούμε τα αποτελέσματα. Για τον λόγο αυτό χρησιμοποιούμε τους διακόπτες PushButtons, τους διακόπτες (DipSwitches), καθώς και την οθόνη LCD της πλακέτας. Πριν όμως αρχίσουμε να σχεδιάζουμε το τελικό κύκλωμα, θα αναφερθούμε σε διάφορα προσχεδιασμένα κυκλώματα (πυρήνες) τα οποία βοηθούν στην χρήση περιφερειακών συσκευών του board.

Χρήση του LCD Display για τον συσσωρευτή.

Ας δούμε πως μπορούμε να χρησιμοποιήσουμε το LCD για να προβάλουμε το περιεχόμενο του συσσωρευτή. Το LCD Display είναι μία μικρή οθόνη υγρών κρυστάλλων, η οποία έχει την δυνατότητα εμφάνισης ASCII χαρακτήρων και αριθμών. Αν και το board DE2 έχει την δυνατότητα να επικοινωνεί με κανονική οθόνη μέσω θύρας VGA, η χρήση του LCD διευκολύνει την υλοποίηση κυκλωμάτων και τον έλεγχο περιορισμένου αριθμού αποτελεσμάτων. Το LCD έχει την δυνατότητα εμφάνισης 32 χαρακτήρων σε δύο γραμμές, όπως φαίνεται στον ακόλουθο πίνακα:

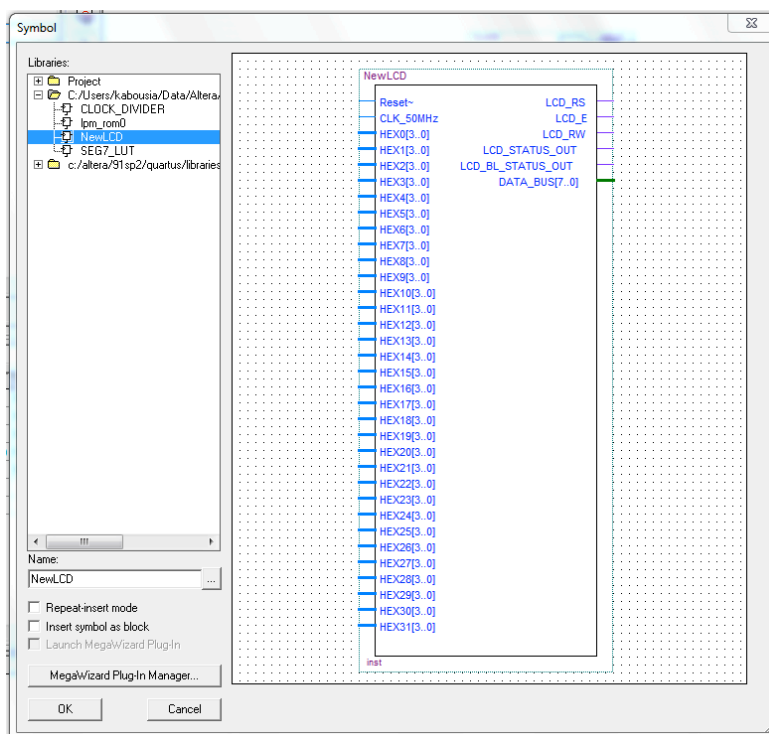
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Η κάθε θέση είναι αριθμημένη με συγκεκριμένη διεύθυνση ώστε να μπορούμε να την διαχειριστούμε χωριστά από τις υπόλοιπες. Το LCD display έχει διασυνδεθεί σε συγκεκριμένα pins του FPGA, τα οποία θα πρέπει να οδηγήσουμε εμείς χρησιμοποιώντας τα κατάλληλα κυκλώματα, και συγκεκριμένα πρωτόκολλα επικοινωνίας ώστε να εμφανιστούν οι ζητούμενοι χαρακτήρες στην οθόνη. Για να αποφύγουμε αυτή την περίπλοκη (για εμάς) διαδικασία, έχει προσχεδιαστεί ένας πυρήνας με πολύ απλή διεπαφή ο οποίος υλοποιεί το απαιτούμενο πρωτόκολλο επικοινωνίας, ενώ ταυτόχρονα επιτρέπει την πολύ απλή χρήση του από άλλα κυκλώματα. Παρακάτω περιγράφουμε αναλυτικά την χρήση του πυρήνα οδήγησης του LCD.

Αρχικά δημιουργούμε ένα νέο σχηματικό με όνομα AccBoard.bdf όπου εισάγουμε τον πυρήνα διαχείρισης του LCD Display, ο οποίος βρίσκεται στην βιβλιοθήκη *CoreLibrary*. Μπορείτε να εισάγετε τον πυρήνα όπως ακριβώς εισάγετε τις βασικές πύλες (αντί για την βιβλιοθήκη *Primitives* επιλέξτε την βιβλιοθήκη *CoreLibrary* και κατόπιν το component *NewLCD*, όπως φαίνεται στην Εικόνα 24). Οι βασικές θύρες του προσχεδιασμένου πυρήνα είναι οι ακόλουθες:

1. Reset~: Όταν ενεργοποιείται στο 0 αρχικοποιεί το LCD (συνδέεται στο γενικότερο σήμα Reset του συστήματος με την κατάλληλη πολικότητα).
2. CLK_50MHz: Συνδέεται στο PIN_N2 του board το οποίο παρέχει ρολόι με συχνότητα 50 MHz.
3. HEX0[3..0], HEX1[3..0],..., HEX31[3..0]: 32 αριθμοί των 4 δυαδικών ψηφίων ο κάθε ένας που αντιστοιχούν στις 32 θέσεις του LCD display όπως απαριθμούνται παραπάνω.

4. LCD_RS, LCD_E, LCD_RW, LCD_STATUS_OUT, LCD_BL_STATUS_OUT, DATA_BUS[7..0]: έξοδοι που καθορίζουν σήματα χρονισμού του πρωτοκόλλου επικοινωνίας με το Display, και πρέπει να οδηγήσουν προκαθορισμένα pins του FPGA.



Εικόνα 24. Το βασικό component NewLCD.

Για να χρησιμοποιήσουμε το LCD display θα πρέπει να αντιστοιχίσουμε την κάθε θέση του LCD σε στατικά ή δυναμικά δεδομένα που θα καθορίζουν τους χαρακτήρες που θα εμφανίζονται στην οθόνη. Τα στατικά δεδομένα είναι ASCII χαρακτήρες που παραμένουν σταθεροί κατά την διάρκεια της λειτουργίας του κυκλώματος, ενώ δυναμικά δεδομένα είναι δεκαεξαδικοί αριθμοί που αλλάζουν κατά την διάρκεια της λειτουργίας του κυκλώματος. Σε κάθε εργαστηριακή άσκηση μπορείτε να επιλέγετε την κάθε περίπτωση χωριστά ή και τις δύο μαζί. Η διαχείριση των δύο αυτών περιπτώσεων γίνεται με διαφορετικό τρόπο, όπως περιγράφεται παρακάτω.

ASCII HEX TABLE																
Hex	Low Hex Digit															
Value	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
--H	2		SP	!	"	#	\$	%	&	'	()	*	+	,	-
--i	3		0	1	2	3	4	5	6	7	8	9	:	;	<	=
--g	4		@	A	B	C	D	E	F	G	H	I	J	K	L	M
--h	5		P	Q	R	S	T	U	V	W	X	Y	Z	[\]
--	6		`	a	b	c	d	e	f	g	h	i	j	k	l	m
--	7		p	q	r	s	t	u	v	w	x	y	z	{		}
															~	DEL

Εικόνα 25. ASCII χαρακτήρες

Στατικά δεδομένα: δηλώνονται ως χαρακτήρες ASCII και αποθηκεύονται εσωτερικά σε μνήμη ROM που ανήκει στο LCD display. Η σειρά τους ξεκινά από την πάνω αριστερή γωνία του Display και καταλήγει στην κάτω δεξιά όπως φαίνεται στον πίνακα του LCD που βρίσκεται στην αρχή της ενότητας. Η μνήμη ROM έχει 32 θέσεις των 8 δυαδικών ψηφίων η κάθε μία, οι οποίες αντιστοιχούν στις 32 θέσεις του LCD σύμφωνα με την αρίθμηση που έχει δοθεί (η θέση 0 της μνήμης αντιστοιχεί στην θέση 0 του LCD, η θέση 1 αντιστοιχεί στην θέση 1 του LCD κλπ). Για να τυπώσουμε έναν σταθερό χαρακτήρα, πρέπει στην αντίστοιχη θέση της μνήμης ROM να αποθηκεύσουμε τον ASCII κωδικό του τον οποίο μπορούμε να βρούμε στην Εικόνα 25. Συγκεκριμένα, ο κωδικός αποτελείται

από δύο δεκαεξαδικά ψηφία, όπου το περισσότερο σημαντικό είναι ο αριθμός γραμμής και το λιγότερο σημαντικό είναι ο αριθμός στήλης (στην τομή της γραμμής με την στήλη βρίσκεται ο ζητούμενος χαρακτήρας). Για παράδειγμα, ο χαρακτήρας 'Α' έχει κωδικό 41₁₆ (προσοχή, η αναπαράσταση του στον πίνακα είναι στο 16αδικό σύστημα και η ίδια αναπαράσταση χρησιμοποιείται και στην σχεδίαση του κυκλώματος). Στις θέσεις που δεν θέλουμε να εμφανίζεται τίποτα βάζουμε τον κενό χαρακτήρα ("20" στο δεκαεξαδικό σύστημα).

Πριν δούμε πως μπορούμε να καθορίσουμε τα περιεχόμενα της μνήμης ROM, θα δούμε πως μπορούμε να φτιάξουμε ένα αρχείο που θα περιέχει τους ζητούμενους χαρακτήρες. Δημιουργούμε ένα δεκαεξαδικό αρχείο με 32 bytes τα οποία αντιστοιχούν στις 32 θέσεις του LCD Display. Από το κυρίως μενού επιλέγουμε *File>New>Hexadecimal (Intel-Format) File*. Στην ερώτηση που ακολουθεί καθορίζουμε το μέγεθος της μνήμης να είναι ίσο με 32 θέσεις, και το μέγεθος κάθε θέσης ίσο με 8 δυαδικά ψηφία. Αμέσως μετά ανοίγει ένας ειδικός κειμενογράφος όπου η κάθε μία θέση της ROM είναι αριθμημένη με τους δείκτες γραμμής/στήλης, όπως φαίνεται στην Εικόνα 26 (εαν προσθέσετε τον δείκτη γραμμής με τον δείκτη στήλης προκύπτει η θέση της μνήμης που αντιστοιχεί στο κελί που βρίσκεται στην τομή της γραμμής με την στήλη). Επιλέξτε μία οποιαδήποτε γραμμή και πατήστε δεξιό κλικ στο ποντίκι πάνω στην διεύθυνση της γραμμής ώστε να σας δοθεί η δυνατότητα να καθορίσετε την κωδικοποίηση των δεδομένων της μνήμης (Memory Radix > Hexadecimal). Επιλέξτε την δεκαεξαδική κωδικοποίηση. Τώρα είστε σε θέση να εισάγετε δεδομένα σε κάθε κελί χωριστά.

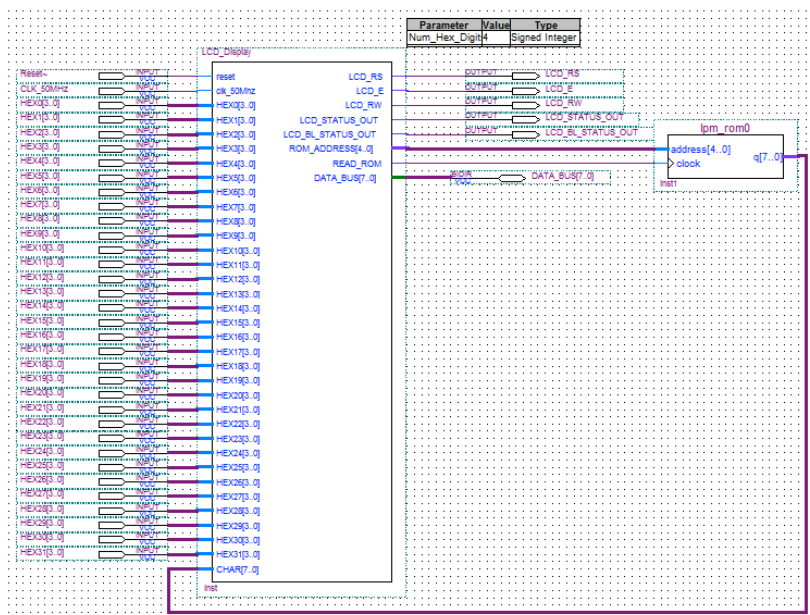
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00
16	00	00	00	00	00	00	00	00
24	00	00	00	00	00	00	00	00

Εικόνα 26. Δεκαεξαδικό αρχείο της μνήμης ROM.

Για τις ανάγκες της τρέχουσας άσκησης ας χρησιμοποιήσουμε τις αριστερότερες θέσεις της πρώτης γραμμής του LCD για να τυπώσουμε την λέξη "ACCUMULATOR", και τις αριστερότερες θέσεις της δεύτερης γραμμής του LCD για να τυπώσουμε το περιεχόμενο του Accumulator σε κάθε χρονική στιγμή (θα δούμε πως θα επιτύχουμε το τελευταίο σε λίγο). Η λέξη "ACCUMULATOR" έχει 11 γράμματα και καταλαμβάνει τις 11 αριστερότερες θέσεις του LCD (0..10). Οι κωδικοί των αντίστοιχων χαρακτήρων όπως περιγράφονται στην Εικόνα 25 είναι οι ακόλουθοι (επιβεβαιώστε το):

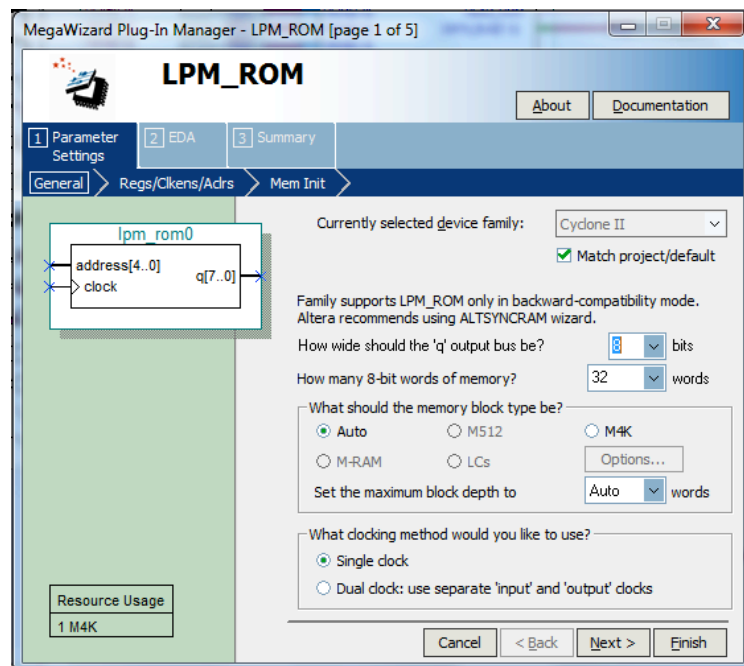
0:41	1:43	2:43	3:55	4:4D	5:55	6:4C	7:41	8:54	9:4F	10:52	11:20	12:20	13:20	14:20	15:20
16:00	17:00	18:20	19:20	20:20	21:20	22:20	23:20	24:20	25:20	26:20	27:20	28:20	29:20	30:20	31:20

Μετά την θέση 10 γεμίζουμε όλη την γραμμή με τον κωδικό 20 ο οποίος αντιστοιχεί στον κενό χαρακτήρα, και το ίδιο κάνουμε και για τις θέσεις 18-31 της δεύτερης γραμμής. Προσωρινά, και για λόγους που θα γίνουν σύντομα κατανοητοί αφήνουμε τις θέσεις 16, 17 στο 00. Εισάγετε τα παραπάνω δεδομένα στην μνήμη ROM και αποθηκεύστε το αρχείο με το όνομα ROM.hex στον κατάλογο του project.

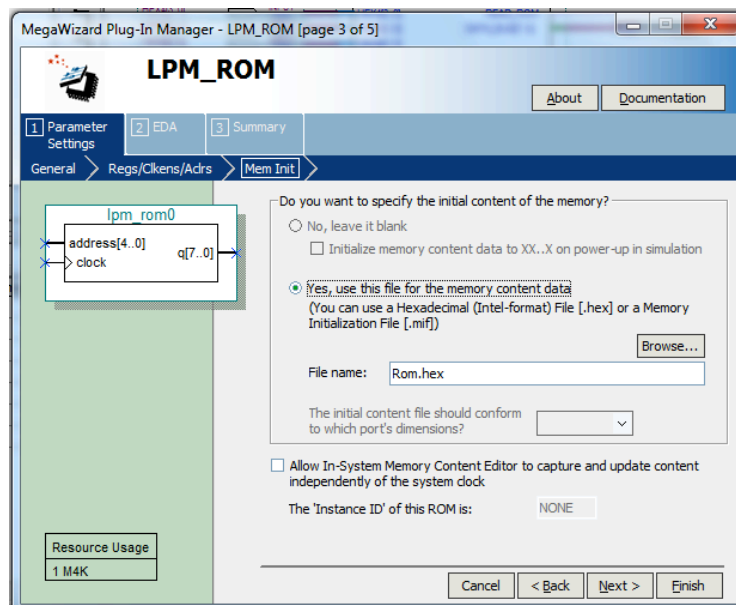


Εικόνα 27. Το εσωτερικό του πυρήνα NewLCD

Εχοντας πλέον δημιουργήσει το δεκαεξαδικό αρχείο μπορούμε να καθορίσουμε τα περιεχόμενα της μνήμης ROM του LCD. Κάνουμε διπλό κλικ στο σύμβολο του LCD που έχουμε τοποθετήσει στο σχηματικό και ανοίγουμε το εσωτερικό του LCD, όπως φαίνεται στην Εικόνα 27. Στην πάνω δεξιά γωνία έχει τοποθετηθεί η μνήμη ROM την οποία και θα μεταβάλλουμε. Κάνουμε διπλό κλικ στην μνήμη ROM και ανοίγει ένα παράθυρο διαλόγου το οποίο φαίνεται στην Εικόνα 28. Προχωρούμε επιλέγοντας “Next” χωρίς να πειράζουμε τις ρυθμίσεις που έχουν προεπιλεγεί, έως ότου φθάσουμε στο παράθυρο που φαίνεται στην Εικόνα 29. Εισάγουμε την διαδρομή που βρίσκεται το δεκαεξαδικό μας αρχείο, μαζί με το όνομα του αρχείου και επιλέγουμε Finish. Το δεκαεξαδικό αρχείο έχει ήδη ανανεώσει τα σταθερά δεδομένα της μνήμης ROM.

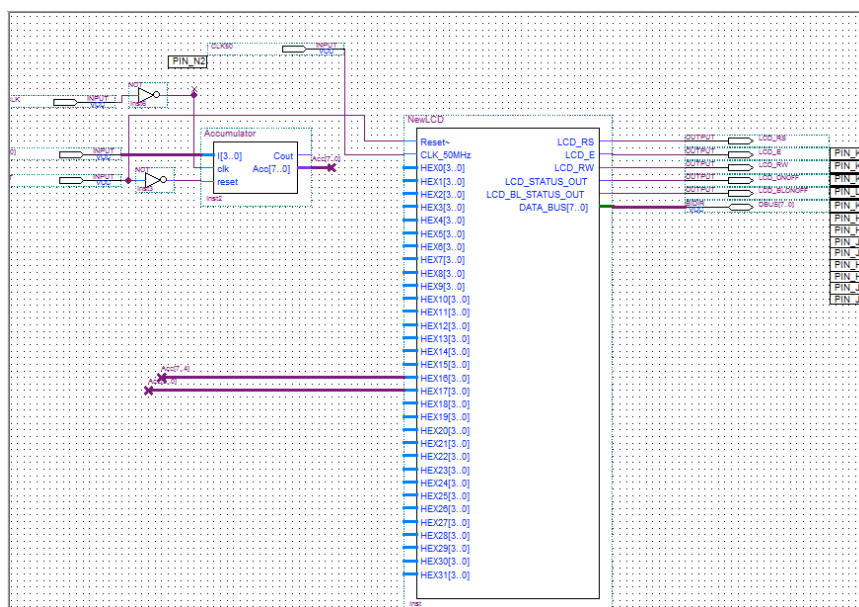


Εικόνα 28. Ρυθμίσεις μνήμης ROM.



Εικόνα 29. Εισαγωγή δεκαεξαδικού αρχείου μνήμης.





















Δυναμικά δεδομένα: δηλώνονται ως δυαδικοί αριθμοί οι οποίοι εμφανίζονται στις αντίστοιχες θέσεις του LCD σε δεκαεξαδική μορφή. Οι αριθμοί αυτοί δεν τοποθετούνται στην μνήμη ROM καθώς μεταβάλλονται δυναμικά κατά την λειτουργία του κυκλώματος. Αντίθετα ορίζονται από τον σχεδιαστή και συνδέονται στις εισόδους $HEX1[3..0]$, ..., $HEX31[3..0]$. Για παράδειγμα, αν θέλουμε να τυπώνουμε το περιεχόμενο ενός διαύλου $BUS[3..0]$ των τεσσάρων δυαδικών ψηφίων στην αριστερότερη θέση της δεύτερης γραμμής του LCD, αρκεί να συνδέσουμε τον δίαυλο BUS στην είσοδο $HEX16[3..0]$ του LCD (προσέξτε ότι οι εισοδοί $HEX0...HEX31$ αντιστοιχούν στην αρίθμηση που έχουμε στις θέσεις του LCD: $0...31$). Προσοχή χρειάζεται στην σειρά με την οποία συνδέουμε την τιμή που θέλουμε να τυπώσουμε στην είσοδο HEX. Το πιο σημαντικό δυαδικό ψηφίο συνδέεται στην είσοδο $HEXxx[3]$ και η λιγότερο σημαντική στην είσοδο $HEXxx[0]$. Για παράδειγμα, στην περίπτωση του διαύλου $BUS[3..0]$ το πλέον σημαντικό ψηφίο είναι το $BUS[3]$ και θα συνδεθεί στο $HEX31[3]$, το $BUS[2]$ θα συνδεθεί στο $HEX31[2]$ κλπ. Ωστόσο, πρέπει ακόμη να παρακαμφθεί η ROM στις περιπτώσεις που σε μία θέση επιθυμούμε δυναμικά και όχι στατικά δεδομένα. Αυτό επιτυγχάνεται με τοποθέτηση της τιμής "00" στην αντίστοιχη θέση της ROM (η αποκωδικοποίηση εσωτερικά παρακάμπτει σε αυτή την περίπτωση την ROM και τροφοδοτεί την αντίστοιχη είσοδο HEX). Στην περίπτωση του συσσωρευτή, επιθυμούμε να βλέπουμε το περιεχόμενο του σε κάθε κύκλο ρολογιού στην δεύτερη γραμμή στις δύο αριστερότερες θέσεις. Για τον λόγο αυτό, θα πρέπει στις θέσεις 16, 17 της ROM να αποθηκευτεί η τιμή 00, και να συνδεθεί η έξοδος $Acc[7..0]$ του συσσωρευτή (βλέπε Εικόνα 23) στις εισόδους $HEX16[3..0]$ και $HEX17[3..0]$ του LCD. Επειδή θέλουμε να τυπώνεται στην αριστερότερη θέση το πιο σημαντικό δεκαεξαδικό ψηφίο του αριθμού $Acc[7..0]$, δηλαδή το $Acc[7..4]$, συνδέουμε το $Acc[7..4]$ στο $HEX16[3..0]$ και το $Acc[3..0]$ στο $HEX17[3..0]$. Οι υπόλοιπες εισοδοί HEX μπορούν να μείνουν ασύνδετες.



Σχηματικό συσσωρευτή με LCD.

Στο σχηματικό AccBoard.bdf τοποθετούμε και τον συσσωρευτή χρησιμοποιώντας το σύμβολο που δημιουργήσαμε σε προηγούμενο βήμα. Οδηγούμε την έξοδο Acc[7..0] του συσσωρευτή στις εισόδους HEX του Display όπως περιγράψαμε πιο πριν. Τοποθετούμε τέσσερις εισόδους με ονόματα: **Reset**, **CLK50**, **HAND_CLK**, και **DIP[3..0]**. Η είσοδος Reset θα συνδεθεί με PushButton οπότε θα έχει διαρκώς την τιμή 1 και όταν το PushButton πιέζεται θα έχει την τιμή 0. Για τον λόγο αυτό το οδηγούμε ως έχει στο core του Display (εκεί ενεργοποιείται στην λογική τιμή '0') και ανεστραμμένο στον συσσωρευτή (εκεί ενεργοποιείται στην λογική τιμή '1'). Το ίδιο συμβαίνει και με τις εισόδους DIP[3..0] οι οποίες θα συνδεθούν στα DipSwitches και τις οποίες τροφοδοτούμε στις εισόδους I[3..0] του συσσωρευτή. Η τιμή που θα δοθεί στα DipSwitches καθορίζει το βήμα αύξησης του συσσωρευτή. Η είσοδος HAND_CLK θα συνδεθεί σε PushButton και θα αποτελέσει το ρολόι του συσσωρευτή. Έτσι κάθε φορά που θα πιέζουμε το PushButton θα αποθηκεύεται η επόμενη τιμή στον συσσωρευτή και μαζί θα εμφανίζεται στο LCD Display. Η είσοδος CLK50 αποτελεί το ρολόι του LCD σύμφωνα με τις προδιαγραφές του. Προσέξτε ότι τα PushButton παρέχουν την λογική τιμή '0' όταν είναι πατημένα οπότε απαιτείται η εισαγωγή ενός αντιστροφέα. Τοποθετούμε τρεις εξόδους με ονόματα **LCD_RS**, **LCD_E**, **LCD_RW** για τις ομώνυμες εξόδους του Display οι οποίες θα συνδεθούν σε προκαθορισμένα pins. Επίσης τοποθετούμε μία θύρα διπλής κατεύθυνσης με όνομα **DBUS[7..0]** η οποία θα συνδεθεί στην θύρα DATA_BUS[7..0] του LCD. Το σχηματικό του κυκλώματος φαίνεται στην Εικόνα 30. Τέλος κάνουμε τις αναθέσεις των pins που φαίνονται στην Εικόνα 31. Επιβεβαιώνουμε ότι είναι οι σωστές αναθέσεις από τον πίνακα του παραρτήματος Α. Τα σήματα Reset, HAND_CLK έχουν συνδεθεί στα PushButtons KEY0, KEY1 αντίστοιχα. Επιπλέον τα σήματα εισόδου DIP[3..0] έχουν συνδεθεί στα SW3-SW0 αντίστοιχα, ώστε να τοποθετείται ο αριθμός από το περισσότερο προς το λιγότερο σημαντικό δυαδικό ψηφίο (αριστερά προς τα δεξιά).

Ακολουθήστε τα βήματα των προηγούμενων ασκήσεων και ελέγξτε την λειτουργία στο board. Δείξτε την στον επιτηρητή σας.

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
 CLK50	Input	PIN_N2	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[7]	Output	PIN_H3	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[6]	Output	PIN_H4	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[5]	Output	PIN_J3	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[4]	Output	PIN_J4	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[3]	Output	PIN_H2	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[2]	Output	PIN_H1	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[1]	Output	PIN_J2	2	B2_N1	3.3-V LVTTTL (default)	
 DBUS[0]	Output	PIN_J1	2	B2_N1	3.3-V LVTTTL (default)	
 DIP[3]	Input	PIN_AE14	7	B7_N1	3.3-V LVTTTL (default)	
 DIP[2]	Input	PIN_P25	6	B6_N0	3.3-V LVTTTL (default)	
 DIP[1]	Input	PIN_N26	5	B5_N1	3.3-V LVTTTL (default)	
 DIP[0]	Input	PIN_N25	5	B5_N1	3.3-V LVTTTL (default)	
 HAND_CLK	Input	PIN_N23	5	B5_N1	3.3-V LVTTTL (default)	
 LCD_BL_STATUS_OUT	Output	PIN_K2	2	B2_N1	3.3-V LVTTTL (default)	
 LCD_E	Output	PIN_K3	2	B2_N1	3.3-V LVTTTL (default)	
 LCD_RS	Output	PIN_K1	2	B2_N1	3.3-V LVTTTL (default)	
 LCD_RW	Output	PIN_K4	2	B2_N1	3.3-V LVTTTL (default)	
 LCD_STATUS_OUT	Output	PIN_L4	2	B2_N1	3.3-V LVTTTL (default)	
 Reset	Input	PIN_G26	5	B5_N0	3.3-V LVTTTL (default)	

Εικόνα 31.

Εργαστηριακή Άσκηση 3

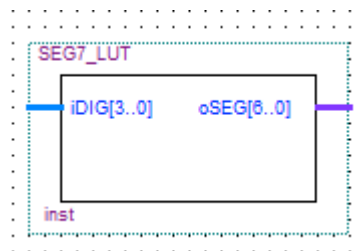
Στόχος της 3^{ης} εργαστηριακής άσκησης είναι η εφαρμογή της ιεραρχικής σχεδίασης και η χρήση των πυρήνων του συστήματος για την σχεδίαση ενός data-path. Με αυτό τον στόχο θα σχεδιάσουμε πάλι το data-path που υλοποιήσαμε στο breadboard. Ωστόσο τώρα το περιβάλλον σχεδίασης μας επιτρέπει να διορθώσουμε την σχεδίαση μας και να αποφύγουμε το πρόβλημα χρονισμού που παρατηρήσαμε εκεί, αλλά και να την επεκτείνουμε με τέτοιο τρόπο ώστε να είναι πιο εύχρηστη. Σχεδιάστε λοιπόν στο Quartus το datapath που περιγράψαμε στην 3^η εργαστηριακή άσκηση που φτιάξαμε στο breadboard με τις ακόλουθες τροποποιήσεις:

1. Οι καταχωρητές A, B θα φορτώνονται εναλλάξ όταν το σήμα Start = 0 από τέσσερα dip switches της επιλογής σας.
2. Τα περιεχόμενα των καταχωρητών A, B θα προβάλλονται στο LCD συνεχώς.
3. Το αποτέλεσμα της ALU θα προβάλλεται σε ένα από τα 7-segment displays του board (δείτε παρακάτω) και η τιμή της σημαίας Z θα προβάλλεται σε ένα από τα LEDs συνεχώς.
4. Οι καταχωρητές A, B θα επανασχεδιαστούν χωρίς την χρήση Gated Clock (χρησιμοποιήστε τις βασικές αρχές Ψηφιακής Σχεδίασης I για τους καταχωρητές).

Σχεδιάστε **ΒΗΜΑΤΙΚΑ** και **ΙΕΡΑΡΧΙΚΑ** τις μονάδες του data path πρώτα και κατόπιν την μονάδα ελέγχου. Κάθε μονάδα (πχ καταχωρητή, αθροιστή, data path, μονάδα ελέγχου κλπ) πρέπει να την σχεδιάζετε χωριστά και να την επιβεβαιώνετε με χρήση χρονικής εξομοίωσης. Μπορείτε να βρείτε τα σχηματικά των ολοκληρωμένων που χρησιμοποιήσαμε και στο breadboard στην βιβλιοθήκη "others/Maxplus2".

Χρήση του 7-segment display

Το σύμβολο του 7-segment display μπορείτε να το αναζητήσετε στην βιβλιοθήκη "altera\CoreLibrary" με το όνομα SEG7_LUT. Κάθε σύμβολο μπορεί να αναπαραστήσει ένα ψηφίο του ρολογιού σε δεκαεξαδική μορφή. Όπως φαίνεται και στο σχήμα παρακάτω, έχει μια είσοδο δεδομένων 4-bit και μια έξοδο 7-bit, η οποία οδηγεί το 7-segment display του board με βάση την είσοδο.



Συνεπώς, για να απεικονίσουμε το πρώτο ψηφίο του ρολογιού στο 7-segment display του board με την ετικέτα HEX7, θα πρέπει να οδηγήσουμε την επιθυμητή ακολουθία bit στην είσοδο και να συνδέσουμε την έξοδο του συμβόλου SEG7_LUT με τα αντίστοιχα pin εξόδου HEX7[6..0], όπως αυτά δίνονται στο παράρτημα.

Για την απεικόνιση του ρολογιού στο board χρησιμοποιήστε τα 7-segment display HEX7 έως HEX2.

(στο εργαστήριο)

Σχεδιάστε όλες τις μονάδες του κυκλώματος και δείξτε στον υπεύθυνο τις κυματομορφές κάθε υπομονάδας χωριστά. Προγραμματίστε το board και δείξτε στον υπεύθυνο την λειτουργία του.



Εργαστηριακή Άσκηση 4: Σχεδίαση με VHDL

Στην παρούσα εργαστηριακή άσκηση θα δούμε πως μπορούμε να χρησιμοποιήσουμε VHDL για να δημιουργήσουμε κυκλώματα απλά. Φυσικά η VHDL δεν έχει σαν στόχο απλά κυκλώματα, αλλά να απλοποιήσει την σχεδίαση πολύπλοκων κυκλωμάτων. Αρχικά όμως θα πρέπει να εξοικειωθούμε με την χρήση της VHDL.

Μέρος 1^ο. Συνδυαστικά Κυκλώματα

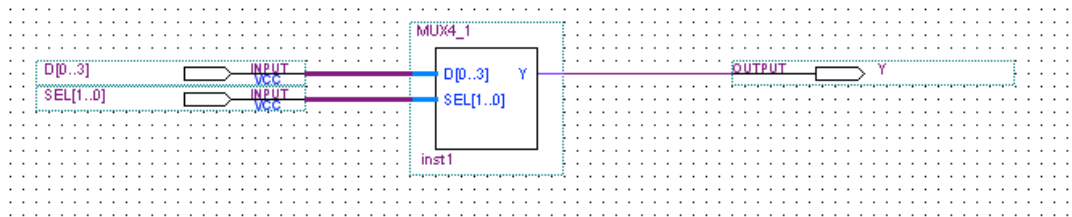
Ερώτημα 1ο

Αρχικά θα σχεδιάσουμε και θα εξομοιώσουμε το κύκλωμα ενός πολυπλέκτη 4 σε 1 με χρήση της γλώσσας VHDL. Δημιουργούμε ένα νέο project και εισάγουμε ένα νέο VHDL αρχείο (επιλογή *File>New>VHDL File*). Αποθηκεύουμε το κενό ακόμη αρχείο με το όνομα MUX4_1.vhd. Στον κειμενογράφο που έχει ανοίξει εισάγουμε τον κώδικα ενός πολυπλέκτη 4 σε 1 όπως φαίνεται στο ακόλουθο σχήμα.

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity MUX4_1 is
5     port(
6         D : in std_logic_vector (0 to 3);
7         SEL:in std_logic_vector (1 downto 0);
8         Y: out std_logic);
9 end MUX4_1;
10
11 architecture RTL of MUX4_1 is
12 begin
13     Y <= D(0) when SEL="00" else D(1) when SEL="01" else D(2) when SEL="10" else D(3);
14 end RTL;
```

Στο σημείο αυτό αξίζει να δούμε κάποιες δυνατότητες που μας δίνει το Quartus για να επιταχύνουμε την διαδικασία συγγραφής περιγραφών σε VHDL. Το Quartus επιταχύνει την διαδικασία σχεδίασης παρέχοντας πρότυπα VHDL. Με δεξί κλικ ποντικιού επιλέγουμε *Insert Template* και κατόπιν επιλέγουμε από το συντακτικό της VHDL την επιλογή *Entity Declaration*. Τότε εμφανίζεται στον κειμενογράφο το συντακτικό της δήλωσης οντότητας από την οποία λείπουν τα συγκεκριμένα στοιχεία της οντότητας που θέλουμε να εισάγουμε. Με όμοιο τρόπο μπορούμε να εισάγουμε και σώμα αρχιτεκτονικής. Πειραματιστείτε με τον παραπάνω κώδικα ώστε να εξοικειωθείτε με τις δυνατότητες του Quartus.

Επόμενο βήμα είναι η δημιουργία συμβόλου για τον παραπάνω πολυπλέκτη, με την γνωστή διαδικασία που μάθαμε στην προηγούμενη άσκηση. Με την δημιουργία συμβόλου γίνεται ταυτόχρονα και μεταγλώττιση οπότε μπορούμε να διορθώσουμε και τυχόν λεκτικά ή συντακτικά σφάλματα. Αφού ολοκληρώσουμε αυτή την διαδικασία δημιουργούμε ένα σχηματικό στο οποίο θα τοποθετήσουμε έναν πολυπλέκτη σαν αυτόν που σχεδιάσαμε. Σε αυτό το σημείο πρέπει να πούμε ότι θα μπορούσαμε επίσης να δημιουργήσουμε μία νέα οντότητα σε VHDL όπου θα χρησιμοποιούσαμε τον πολυπλέκτη ως component. Οποιαδήποτε από τις δύο μεθόδους οδηγεί στο ίδιο αποτέλεσμα. Το σχηματικό το ονομάζουμε όπως και το project και τοποθετούμε τον πολυπλέκτη με τον ίδιο τρόπο που τοποθετούμε ένα οποιοδήποτε σύμβολο της σχεδίασης μας (δείτε τις προηγούμενες εργαστηριακές ασκήσεις). Δημιουργήστε έτσι το σχηματικό που φαίνεται στην Εικόνα 32:



Εικόνα 32

Εκτελέστε compile (αφού πρώτα θέσετε το τρέχων κύκλωμα ως κορυφαία οντότητα του σχεδιασμού (top level entity) και δημιουργήστε αρχείο κυματομορφών για να ελέγξετε την σωστή λειτουργία του κυκλώματος που σχεδιάσατε. Τρέξτε στατική χρονική ανάλυση για να υπολογίσετε την μέγιστη καθυστέρηση του κυκλώματος. Δείξτε αυτή την καθυστέρηση πάνω στην κυματομορφή της εξομοίωσης.

Ερώτημα 2ο

Στο 2^ο μέρος της άσκησης θα υλοποιήσουμε σε VHDL έναν πολυπλέκτη 16 σε 1 χρησιμοποιώντας τον πολυπλέκτη 4 σε 1 που σχεδιάσαμε στο 1^ο μέρος και θα χρησιμοποιήσουμε structural VHDL. Δημιουργήστε ένα νέο αρχείο VHDL και εισάγετε την VHDL που φαίνεται στο ακόλουθο σχήμα:

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 ENTITY MUX16_1 is port(
5     D : in std_logic_vector (0 to 15);
6     SEL:in  std_logic_vector (3 downto 0);
7     F: out std_logic);
8 END MUX16_1;
9
10 ARCHITECTURE RTL OF MUX16_1 IS
11 COMPONENT MUX4_1
12     port(
13         D : in std_logic_vector (0 to 3);
14         SEL:in  std_logic_vector (1 downto 0);
15         Y: out std_logic);
16 END COMPONENT;
17
18 SIGNAL Y : STD_LOGIC_VECTOR (0 to 3)
19
20 BEGIN
21
22 u0: MUX4_1 port map(D=>D(0 to 3), SEL=>Sel(1 downto 0), Y=>Y(0));
23 u1: MUX4_1 port map(D=>D(4 to 7), SEL=>Sel(1 downto 0), Y=>Y(1));
24 u2: MUX4_1 port map(D=>D(8 to 11), SEL=>Sel(1 downto 0), Y=>Y(2));
25 u3: MUX4_1 port map(D=>D(12 to 15), SEL=>Sel(1 downto 0), Y=>Y(3));
26 u4: MUX4_1 port map(D=>Y, SEL=>Sel(3 downto 2), Y=>F);
27
28 END RTL;
29

```

Παρατηρήστε ότι τον πολυπλέκτη MUX4_1 που δημιουργήσαμε στο 1^ο μέρος τον δηλώνουμε σαν component ενώ στο σώμα της αρχιτεκτονικής τοποθετούμε πέντε πολυπλέκτες που δημιουργούν τον ζητούμενο πολυπλέκτη 16 σε 1 (επιβεβαιώστε το). Δημιουργήστε σύμβολο για τον πολυπλέκτη 16 σε 1. Κατόπιν αλλάξτε το σχηματικό που δημιουργήσατε στο 1^ο μέρος, και εισάγετε έναν πολυπλέκτη 16 σε 1. Δημιουργήστε το κατάλληλο αρχείο κυματομορφών και δείξτε ότι ο πολυπλέκτης αυτός λειτουργεί σωστά. Βρείτε την μέγιστη καθυστέρηση του και δώστε την στην αναφορά σας.

Ερώτημα 3ο

Χρησιμοποιώντας την παραπάνω διαδικασία περιγράψτε έναν πλήρη αθροιστή σε VHDL και κατόπιν περιγράψτε σε VHDL έναν αθροιστή των 8 δυαδικών ψηφίων χρησιμοποιώντας 8 πλήρεις αθροιστές. Εκτελέστε εξομοίωση και αποδείξτε την σωστή λειτουργία του. Βρείτε με στατική χρονική ανάλυση την μέγιστη καθυστέρηση του. Σχηματίστε αναφορά με την σχεδίαση και τα αποτελέσματα όλων των εξομοιώσεων που εκτελέσατε.

Ερώτημα 4ο

Περιγράψτε έναν αθροιστή των 8 δυαδικών ψηφίων σε VHDL χρησιμοποιώντας τον τελεστή της πρόσθεσης "+" (θα βρείτε παραδείγματα στις σημειώσεις). Με στατική χρονική ανάλυση και τις κατάλληλες εξομοιώσεις συγκρίνετε τον αθροιστή του προηγούμενου ερωτήματος με αυτόν. Υποδείξτε ποιος από τους δύο είναι ο πιο γρήγορος.

Μέρος 2°.

Ερώτημα 1ο

Αρχικά θα μελετήσουμε την λειτουργία ενός D flip-flop το οποίο είναι η βασική μονάδα μνήμης που χρησιμοποιείται σε ακολουθιακά κυκλώματα (αν και η χρήση του συνήθως υπονοείται και δεν δηλώνεται ρητά). Όπως έχουμε δει στην θεωρία, ο κώδικας σε VHDL που μοντελοποιεί ένα D-flip flop είναι ο ακόλουθος:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity D_ff is
    port(    CLK, D, CLR, SET: in  std_logic;
           Q, Qn : out std_logic);
end D_ff;

architecture RTL of D_ff is
    signal DFF : std_logic;
begin
    seq0 : process(CLK, CLR, SET )
    begin
        if (CLR='1') then DFF<='0';
        elsif (SET='1') then DFF<='1';
        elsif (CLK'event and CLK = '1') then DFF <=D;
        end if;
    end process;
    Q <= DFF; Qn <= not DFF;
end RTL;
```

όπου CLK το ρολόι, D η είσοδος δεδομένων, CLR η ασύγχρονη μηδένιση, SET η σύγχρονη θέση (για να κατανοήσετε πλήρως την μοντελοποίηση του D flip flop μπορείτε να αναφερθείτε στις σημειώσεις και τις διαφάνειες). Δημιουργήστε ένα νέο project και δημιουργήστε κατόπιν ένα σύμβολο για το D flip flop όπως ακριβώς έχετε μάθει. Με χρήση σχηματικού και αρχείου κυματομορφών (με την γνωστή διαδικασία) δείξτε ότι το D flip flop που σχεδιάσατε λειτουργεί σωστά σε όλες τις λειτουργίες του. Παραθέστε τις κυματομορφές στην αναφορά σας. Εκτελέστε στατική χρονική ανάλυση και υπολογίστε την μέγιστη συχνότητα ρολογιού που μπορεί να χρησιμοποιηθεί. Δώστε τα αποτελέσματα στην αναφορά σας.

Ακόλουθα σχεδιάστε ένα latch και παράγετε σύμβολο. Υπενθυμίζουμε ότι το latch είναι ένα αποθηκευτικό στοιχείο με δυνατότητα αποθήκευσης κατά την μισή περίοδο ρολογιού και όχι μόνο κατά την θετική ή αρνητική ακμή του (όπως γίνεται στο flip flop). Παραθέτουμε ακόλουθα τον κώδικα ενός Latch με ασύγχρονη θέση-μηδένιση:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MyLatch is
    port(    Rst, Set, LE, D : in  std_logic;
           Q, Qn: out std_logic);
end MyLatch;

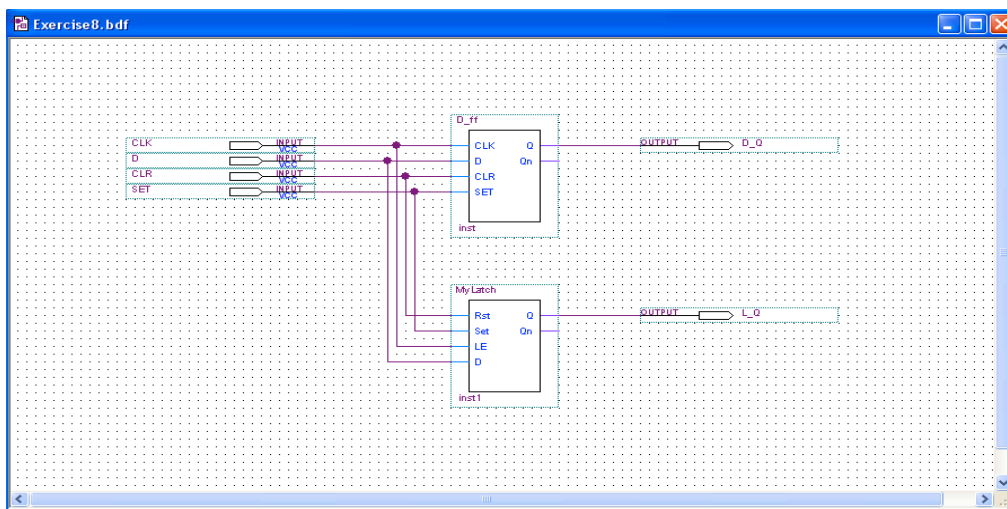
architecture RTL of MyLatch is
    signal FF: std_logic;
begin
```

```

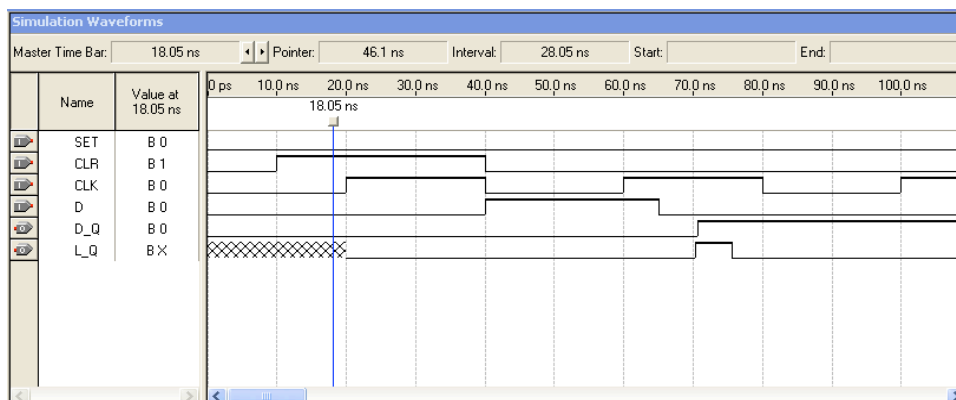
seq0 : process (Rst, Set, D, LE)
begin
    if Rst = '1' then FF <= '0';
    elsif Set = '1' then FF <= '1';
    elsif LE = '1' then FF <= D;
    end if;
end process;
Q <= FF;
Qn <= not FF;
end RTL;

```

Κατόπιν σχεδιάστε το σχηματικό που φαίνεται στην Εικόνα 33, στο οποίο έχουμε τοποθετήσει ένα D flip flop και ένα latch με χρήση των συμβόλων που έχουμε δημιουργήσει. Προσέξτε ότι έχουμε συνδέσει και τις δύο εισόδους D των στοιχείων μνήμης σε κοινή είσοδο D του κυκλώματος. Το ίδιο συμβαίνει και για το ρολόι, την ασύγχρονη θέση και την μηδένιση. Εκτελέστε την εξομοίωση που φαίνεται στο ακόλουθο σχήμα.



Εικόνα 33



Εικόνα 34

Παρατηρήστε από τα αποτελέσματα της εξομοίωσης την διαφορά λειτουργίας του D flip flop και του Latch. Συγκεκριμένα προσέξτε την χρονική περίοδο 60ns-80ns. Την χρονική στιγμή 60ns συμβαίνει η θετική ακμή του ρολογιού. Τότε το D flip flop αποθηκεύει την τιμή '1' που έχει στην είσοδο του η οποία παραμένει μέχρι το τέλος του κύκλου (100 ns). Παρότι αλλάζει η τιμή του D την χρονική στιγμή 65ns δεν επηρεάζεται καθόλου η τιμή που αποθηκεύεται στο Flip flop. Από την άλλη μεριά παρατηρούμε ότι στο latch συμβαίνει κάτι διαφορετικό. Αρχικά αποθηκεύεται η τιμή 1 στο latch αλλά η αλλαγή της τιμής της εισόδου D σε 0 κατά το χρονικό διάστημα 60ns-80ns όπου το ρολόι είναι στην τιμή 1 προκαλεί αποθήκευση της νέας τιμής 0. Άρα βλέπουμε ότι το flip flop αποθηκεύει μόνο στην

ακμή (θετική στο παράδειγμα) ενώ το latch αποθηκεύει σε όλη την ενεργή περίοδο του ρολογιού (θετική στο παράδειγμα).

Ερώτημα 2ο

Σχεδιάστε σε VHDL έναν καταχωρητή 8 δυαδικών ψηφίων (οι καταχωρητές-μετρητές βασίζονται στην λογική των D-flip flop) με δυνατότητες ασύγχρονης θέσης και μηδένισης αλλάζοντας κατάλληλα τον κώδικα του D flip flop (μπορείτε να βρείτε πολλά παραδείγματα στις σημειώσεις – μην χρησιμοποιήσετε structural περιγραφή αλλά περιγραφή συμπεριφοράς). Σώστε τον καταχωρητή στο αρχείο με όνομα Reg8.vhd και δημιουργήστε σύμβολο. Με χρήση σχηματικού και αρχείου κυματομορφών (με την γνωστή διαδικασία) δείξτε ότι ο καταχωρητής που σχεδιάσατε λειτουργεί σωστά σε όλες τις λειτουργίες του. Παραθέστε τις κυματομορφές στην αναφορά σας. Εκτελέστε στατική χρονική ανάλυση και υπολογίστε την μέγιστη συχνότητα ρολογιού που μπορεί να χρησιμοποιηθεί. Δώστε τα αποτελέσματα στην αναφορά σας.

Ερώτημα 3ο

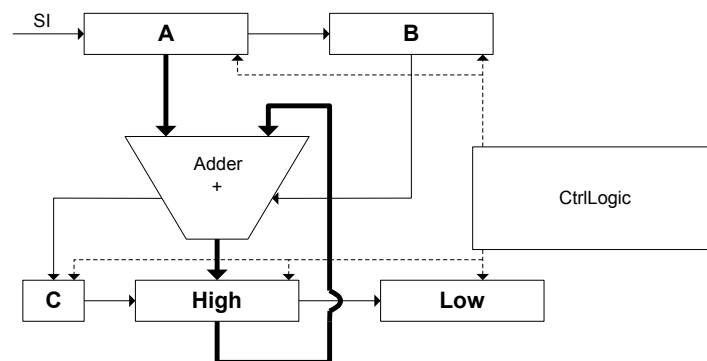
Σχεδιάστε σε VHDL έναν μετρητή των 8 δυαδικών ψηφίων με δυνατότητες μέτρησης από 0-255, ασύγχρονης θέσης και ασύγχρονης μηδένισης. Εκτελέστε εξομοίωση και αποδείξτε την σωστή λειτουργία του μετρητή. Με στατική χρονική ανάλυση βρείτε την μέγιστη ταχύτητα λειτουργίας του και αναφέρετε την στην αναφορά σας.

(στο εργαστήριο)

Υλοποιήστε τον μετρητή στο board χρησιμοποιώντας τα πλήκτρα KEY0, KEY1, KEY2 ως ασύγχρονο reset, ασύγχρονο set και ρολόι αντίστοιχα. Χρησιμοποιήστε το bit 1 από τα dip switches ως επίτρεψη μέτρησης (στην τιμή '1' μετράει και στην τιμή '0' διατηρεί σταθερή την κατάσταση του). Το περιεχόμενο του καταχωρητή τυπώστε το στην πρώτη γραμμή του LCD στις δύο πρώτες θέσεις. Δείξτε το αποτέλεσμα στον επιτηρητή σας.

Εργαστηριακή Άσκηση 5: Σχεδίαση ενός ακολουθιακού πολλαπλασιαστή σε VHDL.

Σε αυτή την ενότητα θα δούμε ένα παράδειγμα ολοκληρωμένης σχεδίασης σε VHDL. Θα σχεδιάσουμε έναν ακολουθιακό πολλαπλασιαστή βασισμένο σε διαδοχικές ολισθήσεις – προσθέσεις όπως έχουμε μάθει στην θεωρία (δείτε τις διαφάνειες με τα αριθμητικά κυκλώματα και το αντίστοιχο κεφάλαιο των σημειώσεων. Το γενικό σχηματικό του πολλαπλασιαστή φαίνεται στο ακόλουθο σχήμα:



Εικόνα 35. Γενική Δομή Πολλαπλασιαστή

Παρατηρούμε στο σχηματικό του πολλαπλασιαστή ότι υπάρχουν δύο καταχωρητές A, B με τα τελούμενα, οι οποίοι φορτώνονται σειριακά από την σειριακή είσοδο SI. Επιπλέον υπάρχουν οι καταχωρητές High, Low που περιέχουν το υψηλό και χαμηλό τμήμα του αποτελέσματος όταν ολοκληρωθεί ο πολλαπλασιασμός. Τέλος υπάρχει και ένα αθροιστής, ένα flip flop που κρατάει το κρατούμενο και μία λογική ελέγχου (CtrlLogic). Το περιεχόμενο του καταχωρητή A οδηγείται στον αθροιστή ο οποίος το προσθέτει με το μερικό άθροισμα που βρίσκεται στον καταχωρητή High κάθε φορά που το λιγότερο σημαντικό ψηφίο του B είναι 1. Εάν το ψηφίο αυτό είναι 0 τότε στην έξοδο του αθροιστή περνάει το περιεχόμενο του High (δεν γίνεται πρόσθεση). Εάν υποθέσουμε ότι ο πολλαπλασιαστής εκτελεί πολλαπλασιασμό των n δυαδικών ψηφίων τότε οι καταχωρητές A, B, High και Low έχουν μέγεθος n δυαδικά ψηφία. Η λογική ελέγχου έχει σαν στόχο να υλοποιήσει την πράξη της πρόσθεσης χρησιμοποιώντας τα αποθηκευτικά στοιχεία (καταχωρητές) και την λειτουργική μονάδα (αθροιστής). Η ακολουθία των βημάτων που απαιτούνται για τον υπολογισμό του αποτελέσματος είναι τα εξής:

1. Στους n πρώτους κύκλους φορτώνονται οι καταχωρητές A, B από την σειριακή είσοδο SI.
2. Στους $2n$ επόμενους κύκλους γίνονται τα ακόλουθα (ανά δύο κύκλους επαναλαμβάνονται τα παρακάτω):
 - α. Γίνεται πρόσθεση και το αποτέλεσμα αποθηκεύεται στον καταχωρητή High.
 - β. Γίνεται δεξιά ολίσθηση στον καταχωρητή B καθώς και στον καταχωρητή μήκους $2n+1$ που δημιουργείται από τους καταχωρητές C->High->Low.

Ας δούμε τώρα πως μπορούμε να σχεδιάσουμε μία τέτοια οντότητα χρησιμοποιώντας την VHDL.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity Reg is
  generic (n: integer:=4);
  port (
    D_IN: in std_logic_vector (n-1 downto 0);
    SI, CLK, RST, SLOAD, ENABLE: in std_logic;
    SO: out std_logic;
    D_OUT: out std_logic_vector (n-1 downto 0));
end Reg;

architecture RTL of Reg is
  signal F: std_logic_vector (n-1 downto 0);
begin
  p0: process(RST, CLK)
  begin
    if (RST='1') then F<=(n-1 downto 0 => '0');
    elsif (CLK'event and CLK='1') then
      if (ENABLE='1') then
        if (SLOAD='0') then F<=D_IN;
        else F<=SI & F(n-1 downto 1);
        end if;
      end if;
    end if;
  end process;
  D_OUT<=F;
  SO<=F(0);
end RTL;

```

Εικόνα 36. Οντότητα Καταχωρητή

Σχεδίαση Καταχωρητή

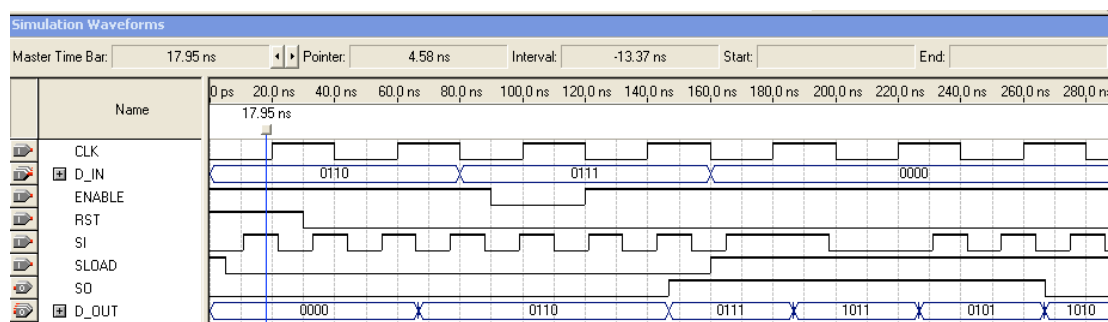
Η πρώτη οντότητα που θα σχεδιάσουμε είναι ο καταχωρητής. Προσέξτε ότι έχουμε 5 διαφορετικούς καταχωρητές. Εμείς θα σχεδιάσουμε μία οντότητα καταχωρητή με όλες τις δυνατότητες ώστε να την χρησιμοποιήσουμε για την δημιουργία και των 5 καταχωρητών. Οι δυνατότητες που θα έχει ο κάθε καταχωρητής είναι οι ακόλουθες:

1. Παράλληλη και σειριακή φόρτωση.
2. Ολίσθηση.
3. Ασύγχρονη μηδένιση.

Στην Εικόνα 36 παραθέτουμε τον κώδικα του καταχωρητή. Παρατηρήστε ότι ο καταχωρητής είναι παραμετρικός έτσι ώστε να προσαρμόζεται το μέγεθος του ανάλογα με τις ανάγκες μας κάθε φορά. Τα σήματα δι-επαφής του είναι τα ακόλουθα:

1. D_IN: θύρα παράλληλων δεδομένων
2. SI: σειριακή είσοδος
3. CLK: είσοδος ρολογιού
4. RST: είσοδος ασύγχρονου μηδενισμού
5. SLOAD: είσοδος καθορισμού ολίσθησης/παράλληλης φόρτωσης
6. ENABLE: είσοδος επίτρεψης λειτουργίας
7. SO: σειριακή έξοδος
8. D_OUT: παράλληλη θύρα εξόδου

Εξομοιώστε τον παραπάνω καταχωρητή και επιβεβαιώστε ότι όλες οι λειτουργίες του υλοποιούνται σωστά. Συγκεκριμένα εκτελέστε την ακόλουθη εξομοίωση:



Εικόνα 37

Παρατηρήστε τα ακόλουθα στην λειτουργία του καταχωρητή:

1. Το σήμα Rst αρχικοποιεί τον καταχωρητή στο 0000. Το σήμα ρολογιού κατά την αρχικοποίηση δεν παίζει κανένα ρόλο (η αρχικοποίηση γίνεται ασύγχρονα).
2. Λίγο μετά την θετική ακμή του ρολογιού την χρονική στιγμή 60ns γίνεται αποθήκευση των δεδομένων που υπάρχουν στην θύρα D_IN (παράλληλη φόρτωση). Η λειτουργία αυτή καθορίζεται από το σήμα SLOAD=0 και φυσικά από το σήμα ENABLE (επίτρεψη φόρτωσης) που είναι στο 1 (επίτρεψη λειτουργίας).
3. Στην θετική ακμή που συμβαίνει την χρονική στιγμή 100ns δεν συμβαίνει φόρτωση των δεδομένων 0111 που βρίσκονται στην θύρα D_IN καθώς το σήμα ENABLE είναι στο 0 (η επίτρεψη λειτουργίας στο 0 διατηρεί τα δεδομένα του καταχωρητή αναλοίωτα). Στην επόμενη θετική ακμή 140ns συμβαίνει φόρτωση του 0111 καθώς πλέον το σήμα ENABLE είναι στο 1 (έχει ενεργοποιηθεί).
4. Στις θετικές ακμές που έπονται το σήμα SLOAD=1 προκαλεί σειριακή φόρτωση των δεδομένων από την σειριακή είσοδο SI.

Σχεδίαση Αθροιστή

Η δεύτερη μονάδα που θα σχεδιάσουμε είναι ο αθροιστής 4 δυαδικών ψηφίων. Η σχεδίαση ενός αθροιστή μπορεί να γίνει πολύ εύκολα, και στην θεωρία έχουμε δώσει πολλά παραδείγματα σχεδίασης. Στην συγκεκριμένη περίπτωση χρειαζόμαστε έναν αθροιστή που να έχει μία επιπλέον δυνατότητα. Προσέξτε από τον αλγόριθμο του πολλαπλασιασμού ότι είτε γίνεται πρόσθεση $A + High \Rightarrow High$ είτε στην έξοδο του πολλαπλασιαστή περνάει η τιμή του High (δείτε και το μπλοκ διάγραμμα του πολλαπλασιαστή). Με αυτό το δεδομένο η περιγραφή σε VHDL του πολλαπλασιαστή φαίνεται στην Εικόνα 38. Παρατηρήστε στην περιγραφή αυτή ότι έχει προστεθεί το σήμα CTRL το οποίο καθορίζει εάν θα γίνει πρόσθεση ή αν θα περάσει η τιμή της θύρας B (εκεί πρέπει να συνδέσουμε την έξοδο του καταχωρητή High αργότερα) στην έξοδο F. Εκτελέστε στατική χρονική ανάλυση και υπολογίστε την καθυστέρηση του αθροιστή. Εκτελέστε χρονική εξομοίωση για να δείξετε όλες τις δυνατότητες του αθροιστή. Ένα παράδειγμα τέτοιας εξομοίωσης φαίνεται στην Εικόνα 39.

```

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_arith.ALL;
use ieee.std_logic_unsigned.ALL;

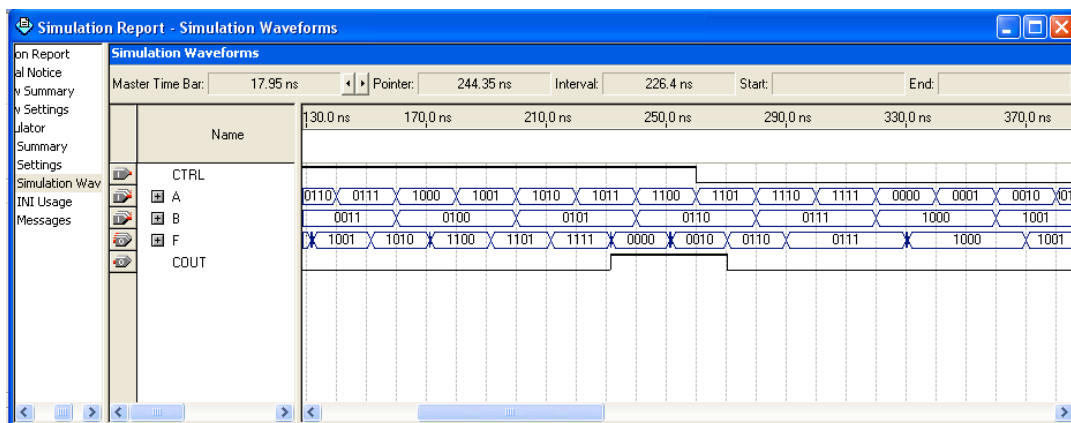
ENTITY Adder IS
generic (n: integer :=4);
PORT
(
    CTRL    : IN    STD_LOGIC;
    A, B    : IN    STD_LOGIC_VECTOR(n-1 DOWNTO 0);
    F       : OUT   STD_LOGIC_VECTOR(n-1 DOWNTO 0);
    COUT    : OUT   STD_LOGIC
);
END Adder;

ARCHITECTURE RTL OF Adder IS
    SIGNAL Interm : STD_LOGIC_VECTOR(n downto 0);
BEGIN

    p0: process(A, B, CTRL)
    begin
        if (CTRL='0') then Interm<='0' & B;
        else Interm<=('0' & A)+('0' & B);
        end if;
    end process;
    F<=Interm(n-1 DOWNTO 0);
    COUT<=Interm(n);
END RTL;

```

Εικόνα 38. Αθροιστής



Εικόνα 39

Παρατηρήστε ότι κάθε φορά που αλλάζει η τιμή του A ή του B προκαλείται μία αλλαγή της εξόδου του αθροιστή. Όταν το σήμα CTRL είναι στην μονάδα τότε στην έξοδο F περνάει το άθροισμα των A, B αλλιώς περνάει η τιμή του B.

Σχεδίαση Μονάδας Ελέγχου

Η τρίτη μονάδα που θα σχεδιάσουμε είναι και η πιο πολύπλοκη. Πρόκειται για την μονάδα ελέγχου η οποία θα συντονίζει την λειτουργία των υπόλοιπων μονάδων. Για παράδειγμα, η μονάδα αυτή θα καθορίσει πότε κάθε καταχωρητής θα φορτώνει παράλληλα, ή σειριακά, ή πότε θα κρατάει τα δεδομένα του αναλλοίωτα. Αλλά ας δούμε βήμα προς βήμα την σχεδίαση της μονάδας ελέγχου.

Η λειτουργία του πολλαπλασιαστή μπορεί να χωριστεί αρχικά σε δύο βασικές φάσεις: την φάση σειριακής φόρτωσης των καταχωρητών A και B, και την φάση του πολλαπλασιασμού των A και B. Ας υποθέσουμε ότι οι καταχωρητές δεδομένων έχουν μήκος n δυαδικά ψηφία (δηλ. εκτελείται πολλαπλασιασμός των n δυαδικών ψηφίων). Τότε η πρώτη φάση αποτελείται από 2n κύκλους, κατά τους οποίους φορτώνονται οι δύο καταχωρητές A, B. Προσέξτε από την Εικόνα 35 ότι έχουμε συνδέσει σειριακά τους δύο καταχωρητές A, B έτσι ώστε να τους φορτώσουμε σειριακά σαν να είναι ένας. Στη δεύτερη φάση εκτελείται ο πολλαπλασιασμός, οπότε απαιτούνται n ζεύγη κύκλων ρολογιού (2n κύκλοι και πάλι) όπου σε κάθε ζεύγος γίνεται μία πρόσθεση και μία ολίσθηση δεξιά των καταχωρητών C→High→Low (δείτε Εικόνα 35). Άρα καταλαβαίνουμε ότι απαιτείται μία μηχανή

κατάστασης η οποία θα καθορίζει σε ποιο σημείο βρίσκεται η εκτέλεση του πολλαπλασιασμού κάθε φορά και θα καθορίζει τις τιμές των σημάτων που θα οδηγούν τους καταχωρητές κατάλληλα.

Ας δούμε αρχικά ποια σήματα απαιτούνται στην δι-επαφή της μονάδας ελέγχου. Ακόλουθα παραθέτουμε τον κώδικα που περιγράφει την δι-επαφή:

```
entity CtrlLogic is
  generic ( n: integer := 4 );
  port (
    Rst, CLK : in std_logic;
    SL_A, SL_B, SL_H, SL_L, SL_C: out std_logic;
    EN_A, EN_B, EN_H, EN_L, EN_C: out std_logic );
end CtrlLogic;
```

Στο τμήμα σταθερών δηλώνεται η σταθερά n που αντιπροσωπεύει το μήκος των τελουμένων A, B. Τα δύο πρώτα σήματα Rst, CLK είναι τα σήματα αρχικοποίησης και ρολογιού τα οποία γνωρίζουμε καλά. Η δεύτερη σειρά αποτελείται από τα σήματα που θα οδηγούν τα σήματα SLOAD των καταχωρητών A, B, High (H), Low (L) και C (δείτε Εικόνα 35 και Εικόνα 36). Η τρίτη σειρά αποτελείται από τα σήματα που θα οδηγήσουν τα σήματα ENABLE των αντίστοιχων καταχωρητών. Με τα σήματα που αναφέραμε ελέγχεται πλήρως η λειτουργία όλων των καταχωρητών του πολλαπλασιαστή. Παρατηρούμε ότι δεν υπάρχει σήμα που να καθορίζει τι είδους πράξη θα εκτελεί ο αθροιστής (σήμα CTRL δείτε Εικόνα 38). Όμως θυμηθείτε ότι αυτό στην πραγματικότητα καθορίζεται από το λιγότερο σημαντικό δυαδικό ψηφίο του B. Άρα πράγματι δεν είναι ευθύνη της μονάδας ελέγχου να καθορίζει την λειτουργία του αθροιστή.

Ας δούμε ακόλουθα κάποιες δηλώσεις που απαιτούνται στην περιγραφή της μονάδας ελέγχου:

```
type state_type is (LOAD, MULT, HOLD);
type mult_type is (SHIFT, ADD);
signal state : state_type;
signal m_state : mult_type;
signal count : std_logic_vector (n-1 downto 0);
```

Ο τύπος state_type χρησιμοποιείται για να ονομάσουμε τις καταστάσεις που μπορεί να βρεθεί η μονάδα ελέγχου. Η κατάσταση LOAD είναι η κατάσταση φόρτωσης των καταχωρητών A, B. Η κατάσταση MULT είναι η κατάσταση πολλαπλασιασμού των A, B. Τέλος η κατάσταση HOLD είναι η τελική κατάσταση όπου ο πολλαπλασιαστής έχει ολοκληρώσει και περιμένει το σήμα Rst για να ξεκινήσει τον επόμενο πολλαπλασιασμό. Το σήμα state έχει τύπο state_type και είναι ουσιαστικά η κατάσταση του πολλαπλασιαστή σε κάθε χρονική στιγμή. Ο τύπος mult_type χρησιμοποιείται για να ξεχωρίσει τις δύο υπο-καταστάσεις της κατάστασης MULT. Όπως έχουμε ήδη αναφέρει, χρειάζονται η ζεύγη κύκλων ρολογιού για να ολοκληρωθεί η κατάσταση MULT όπου σε κάθε ζεύγος συμβαίνει μία ολίσθηση και μία πρόσθεση που αντιστοιχούν στις τιμές SHIFT, ADD του τύπου mult_type. Το σήμα m_state καθορίζει την ακριβή φάση της κατάστασης MULT στην οποία βρισκόμαστε κάθε φορά. Τέλος υπάρχει και ένας μετρητής με το όνομα count ο οποίος μετράει κάθε φορά τους απαραίτητους κύκλους που πρέπει να εκτελεστούν. Ακόλουθα παραθέτουμε τον κώδικα σε VHDL της κύριας διαδικασίας (process) της μονάδας ελέγχου.

```
p0: process(Rst, CLK)
begin
  if (Rst='1') then
    state<=LOAD;
    count<=(n-1 downto 0 => '0');
  elsif (CLK'event and CLK='1') then
    case state is
      when LOAD => if (conv_integer(count)<2*n-1) then count<=count+1;
                    else count<=(n-1 downto 0 => '0');
                    state<=MULT; m_state<=ADD;
                    end if;
      when MULT => if (m_state=ADD) then m_state<=SHIFT;
                    elsif (m_state=SHIFT) then
```

```

        m_state<=ADD;
        if (conv_integer(count)<n-1) then count<=count+1;
        else state<=HOLD;
        end if;
    end if;
    when HOLD => null;
end case;
end if;
end process;

```

Παρατηρούμε ότι η διαδικασία είναι ευαίσθητη στο ρολόι και το σήμα αρχικοποίησης. Το σήμα αρχικοποίησης Rst λειτουργεί ασύγχρονα και θέτει την μονάδα στην κατάσταση LOAD ενώ ταυτόχρονα μηδενίζει τον μετρητή. Προσέξτε ότι ο μετρητής μηδενίζεται παραμετρικά έτσι ώστε η μονάδα ελέγχου να μπορεί να χρησιμοποιηθεί για οποιοδήποτε μήκος πολλαπλασιασμού.

Μετά την αρχικοποίηση περιγράφεται η λειτουργία της μονάδας ελέγχου κατά την διάρκεια της εκτέλεσης της λειτουργίας του πολλαπλασιασμού. Συγκεκριμένα σε κάθε κύκλο ρολογιού εκτελούνται μία σειρά από πράξεις που εξαρτώνται από την τρέχουσα κατάσταση της μονάδας ελέγχου. Στην κατάσταση LOAD η μονάδα παραμένει για $2n$ κύκλους. Έτσι ελέγχει την τιμή του μετρητή count τον οποίο αυξάνει μέχρι να φθάσει στην τιμή $2n-1$ (ξεκινάει από το 0). Τότε μηδενίζει πάλι τον μετρητή και θέτει σαν επόμενη κατάσταση την κατάσταση MULT ώστε να ξεκινήσει στον επόμενο κύκλο η διαδικασία του πολλαπλασιασμού. Επίσης καθορίζει σαν πρώτη ενέργεια της φάσης του πολλαπλασιασμού την πρόσθεση ($m_state \leq ADD$).

Στην κατάσταση MULT κάνουμε είτε ολίσθηση είτε πρόσθεση και εναλλάσσεται η λειτουργία για τον επόμενο κύκλο (ξεκινάμε από πρόσθεση). Ειδικότερα στην φάση που εκτελείται ολίσθηση (SHIFT) αυξάνει η τιμή του μετρητή count κατά ένα και όταν φθάσει την τιμή $n-1$ (n ζεύγη κύκλων έχουν ολοκληρωθεί) τότε αλλάζει η τρέχουσα κατάσταση σε HOLD, καθώς έχει ολοκληρωθεί η επεξεργασία των δεδομένων. Τέλος στην κατάσταση HOLD παρατηρούμε ότι δεν εκτελείται καμία ενέργεια, αλλά απλά η μονάδα παραμένει σε αυτή την κατάσταση μέχρι να πάρει σήμα αρχικοποίησης.

Όλη η παραπάνω ανάλυση αφορά την μετάβαση της μονάδας ελέγχου από τις διάφορες καταστάσεις. Κάποιος θα μπορούσε να πει ότι το μόνο που έπρεπε να γίνει είναι να μετρήσει ο μετρητής count μέχρι το $4n$ ($2n$ κύκλοι για την φόρτωση και άλλοι $2n$ για τον πολλαπλασιασμό). Ωστόσο δεν πρέπει να ξεχνάμε ότι η μετάβαση των καταστάσεων πέραν της μέτρησης έχει έναν ακόμη ρόλο. Να καθορίσει τις ενέργειες της μονάδας ελέγχου, δηλαδή την ανάθεση των σημάτων ελέγχου τα οποία θα καθορίσουν την λειτουργία των καταχωρητών. Επειδή η λειτουργία του κάθε καταχωρητή διαφοροποιείται ανάλογα με την κατάσταση της μονάδας ελέγχου, απαιτείται σαφής προσδιορισμός των καταστάσεων. Ας δούμε αναλυτικότερα τις ενέργειες της μονάδας ελέγχου που έπονται της process:

Καταχωρητής A:

(1) $EN_A \leq '1'$ **when** state=LOAD **else** '0';

(2) $SL_A \leq '1'$ **when** state=LOAD **else** '0';

Ο καταχωρητής A λειτουργεί μόνο κατά την διάρκεια της σειριακής φόρτωσης LOAD και τον υπόλοιπο χρόνο διατηρεί τα δεδομένα του (ανάθεση 1). Κατά την διάρκεια της σειριακής φόρτωσης το σήμα SL_A πρέπει να είναι στην τιμή '1' (δείτε λειτουργία καταχωρητή).

Καταχωρητής B:

(1) $EN_B \leq '1'$ **when** (state=LOAD **or** (state=MULT **and** $m_state=SHIFT$)) **else** '0';

(2) $SL_B \leq '1'$ **when** (state=LOAD **or** (state=MULT **and** $m_state=SHIFT$)) **else** '0';

Ο καταχωρητής B λειτουργεί διαφορετικά από τον A. Ενεργοποιείται και στην αρχική φόρτωση αλλά και στην διάρκεια του πολλαπλασιασμού καθώς ο καταχωρητής B ολισθαίνει δεξιά κατά την διάρκεια του πολλαπλασιασμού στις φάσεις της ολίσθησης. Η συμπεριφορά αυτή αναπαρίσταται με την ανάθεση του λογικού '1' στο σήμα EN_B . Ο καταχωρητής B ολισθαίνει πάντα οπότε το σήμα SL_B είναι όμοιο με το EN_B .

Καταχωρητής High (H):

- (1) EN_H<='1' **when** state=MULT **else** '0';
 (2) SL_H<='1' **when** m_state=SHIFT **else** '0';

Ο καταχωρητής H λειτουργεί μόνο κατά την διάρκεια του πολλαπλασιασμού είτε για να αποθηκεύσει το αποτέλεσμα της πρόσθεσης, είτε για να ολισθήσει δεξιά. Έτσι η ανάθεση (1) ενεργοποιεί τον καταχωρητή H σε όλη την κατάσταση MULT. Το σήμα SL_H που καθορίζει εάν ο καταχωρητής θα κάνει ολίσθηση (SL_H=1) ή παράλληλη φόρτωση (SL_H=0) καθορίζεται ανάλογα με την κατάσταση που βρίσκεται η μονάδα (ολίσθηση/SHIFT ή πρόσθεση/ADD αντίστοιχα).

Καταχωρητής Low (L):

- (1) EN_L<='1' **when** (state=MULT **and** m_state=SHIFT) **else** '0';
 (2) SL_L<='1' **when** (state=MULT **and** m_state=SHIFT) **else** '0';

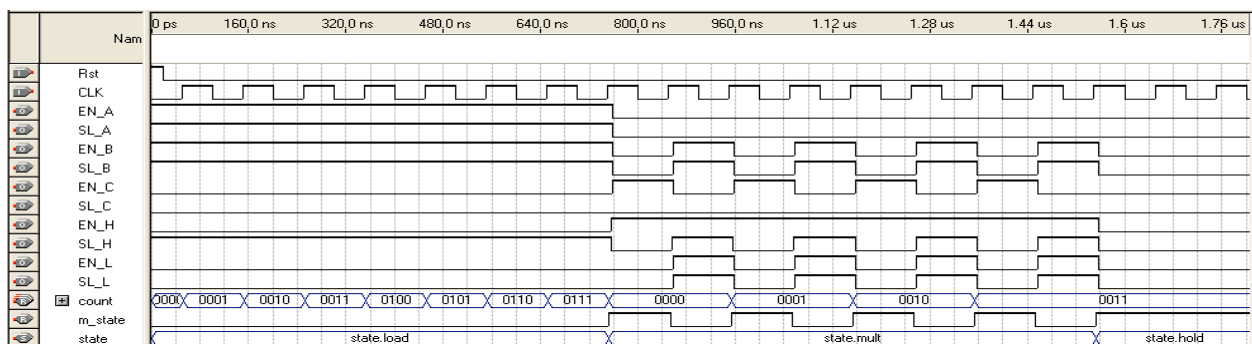
Ο καταχωρητής L κάνει μόνο ολίσθηση όταν η κατάσταση του πολλαπλασιαστή είναι η MULT και ειδικότερα η SHIFT. Σε όλες τις άλλες περιπτώσεις διατηρεί τα δεδομένα του. Έτσι οι αναθέσεις των σημάτων EN_L και SL_L είναι όμοιες και φαίνονται παραπάνω.

Καταχωρητής C:

- EN_C<='1' **when** (state=MULT **and** m_state=ADD) **else** '0';
 SL_C<='0';

Πρόκειται για ένα απλό flip flop το οποίο αποθηκεύει το κρατούμενο της πρόσθεσης. Λειτουργεί μόνο σε κάθε κύκλο που γίνεται πρόσθεση. Πάντα φορτώνει το δεδομένο από την παράλληλη θύρα οπότε δεν κάνει ποτέ ολίσθηση (ανάθεση 2).

Εισάγετε την παραπάνω περιγραφή και εκτελέστε μεταγλώττιση. Η εξομοίωση της μονάδας ελέγχου είναι δυσκολότερη από την εξομοίωση των υπόλοιπων μονάδων. Αυτό οφείλεται στην δυσκολία παρακολούθησης σημάτων που δεν είναι προφανής η χρήση και λειτουργία τους (τέτοια είναι τα σήματα ελέγχου καταχωρητών και λειτουργικών μονάδων). Ωστόσο, όπως θα δούμε, εργαλεία σχεδίασης όπως το Quartus διευκολύνουν στον έλεγχο τέτοιων μονάδων με τις δυνατότητες που προσφέρουν. Θέσετε την μονάδα ελέγχου ως την κορυφαία μονάδα σχεδίασης και εκτελέστε μεταγλώττιση. Διορθώστε λάθη εάν υπάρχουν. Δημιουργήστε ένα νέο αρχείο κυματομορφών για συνολική διάρκεια εξομοίωσης 10μsec, όπου θα ορίσετε το σήμα Rst ενεργό για 20ns και κατόπιν απενεργοποιημένο. Το σήμα ρολογιού θα έχει περίοδο 100ns και duty cycle 50%. Αυτά τα δύο σήματα αρκούν για να εκτελεστεί η εξομοίωση καθώς αποτελούν και τις μοναδικές εισόδους της μονάδας ελέγχου. Για να παρακολουθήσουμε τα αποτελέσματα της εξομοίωσης θα πρέπει να προσθέσουμε στο αρχείο κυματομορφών φυσικά όλες τις εξόδους της μονάδας ελέγχου. Ωστόσο, είναι εξαιρετικά δύσκολο να εντοπίσουμε κάποιο λάθος περιγραφής παρακολουθώντας μόνο τις εξόδους της μονάδας ελέγχου. Για τον λόγο αυτό θα προσθέσουμε και μερικά εσωτερικά σήματα τα οποία θα μας βοηθήσουν να κατανοήσουμε σε κάθε χρονική στιγμή της εξομοίωσης ποιες πρέπει να είναι οι τιμές των σημάτων που αναμένονται. Τα σήματα αυτά είναι οι καταστάσεις state, m_state καθώς και η τιμή του μετρητή count. Για να προσθέσουμε τα σήματα αυτά θα πρέπει να ψάξουμε στο *InsertNode or Bus>Node Finder>Filter: Design Entry(All names)* με την γνωστή διαδικασία. Εκτελέστε εξομοίωση, και επαληθεύστε ότι έχετε το ίδιο αποτέλεσμα με την Εικόνα 40.



Εικόνα 40. Αποτελέσματα εξομοίωσης μονάδας ελέγχου

Παρατηρήστε ότι η κατάσταση state παίρνει τις τιμές load, mult και hold ενημερώνοντας τον σχεδιαστή χρησιμοποιώντας κυριολεκτήματα που ο ίδιος έχει χρησιμοποιήσει για να κάνει

ευανάγνωστη την σχεδίαση του. Επαληθεύστε ότι η εξομοίωση είναι σωστή και αναλύστε στην αναφορά σας κάθε κύκλο ρολογιού με τις τιμές των σημάτων.

Σχεδίαση Πολλαπλασιαστή

Έχοντας σχεδιάσει τις βασικές δομικές μονάδες, απομένει πλέον να σχεδιάσουμε τον πολλαπλασιαστή διασυνδέοντας κατάλληλα τις επιμέρους μονάδες. Η δι-επαφή του πολλαπλασιαστή φαίνεται ακόλουθα:

```
entity Multiplier is  
  generic (n: integer :=4);  
  port (  
    Rst, CLK, SI      : in std_logic;  
    Low, High         : out std_logic_vector (n-1 downto 0));  
end Multiplier;
```

Αρχικά ορίζουμε την σταθερά n που δηλώνει το μήκος των τελουμένων του πολλαπλασιασμού (στην περίπτωση μας είναι 4). Στις θύρες εισόδου ανήκουν μόνο το σήμα αρχικοποίησης, ρολογιού και σειριακής εισόδου, ενώ στις θύρες εξόδου ανήκουν οι έξοδοι των καταχωρητών High και Low του αποτελέσματος.

Στο σώμα αρχιτεκτονικής θα πρέπει να δηλώσουμε αρχικά τα components που θα χρησιμοποιηθούν, όπως φαίνεται ακόλουθα:

```
component CtrlLogic  
  generic ( n: integer := 4 );  
  port (   Rst, CLK : in std_logic;  
          SL_A, SL_B, SL_H, SL_L, SL_C : out std_logic;  
          EN_A, EN_B, EN_H, EN_L, EN_C : out std_logic );  
end component;  
component Adder  
  generic (n: integer :=4);  
  port (   CTRL   : in      std_logic;  
          A, B     : in      std_logic_vector (n-1 downto 0);  
          F        : out     std_logic_vector (n-1 downto 0);  
          COUT     : out     std_logic );  
end component;  
component Reg  
  generic (n: integer:=4);  
  port (   D_IN: in std_logic_vector (n-1 downto 0);  
          SI, CLK, RST, SLOAD, ENABLE: in std_logic;  
          SO: out std_logic;  
          D_OUT: out std_logic_vector (n-1 downto 0));  
end component;
```

Ακόλουθα παραθέτουμε τον κώδικα του πολλαπλασιαστή:


```

signal SL_A, EN_A, SO_A, SL_B, EN_B, SO_B, SL_C, EN_C, SO_C : std_logic;
signal SL_H, EN_H, SO_H, SL_L, EN_L, SO_L : std_logic;
signal A,B,F_ADD, H: std_logic_vector (n-1 downto 0);
signal C, COUT : std_logic_vector (0 downto 0);

begin
uA: Reg generic map (n=>4)
port map (D_IN=>(n-1 downto 0=>'0'), SI=>SI, CLK=>CLK, RST=>RST, SLOAD=>SL_A,
ENABLE=>EN_A, SO=>SO_A, D_OUT=>A);
uB: Reg generic map (n=>4)
port map (D_IN=>(n-1 downto 0=>'0'), SI=>SO_A, CLK=>CLK, RST=>RST, SLOAD=>SL_B,
ENABLE=>EN_B, SO=>SO_B, D_OUT=>B);
uAdder: Adder generic map (n=>4)
port map (CTRL=>B(0), A=>A, B=>B, F=>F_ADD, COUT=>COUT(0));
uC: Reg generic map (n=>1)
port map (D_IN=>COUT, SI=>'0', CLK=>CLK, RST=>RST, SLOAD=>SL_C, ENABLE=>EN_C,
SO=>SO_C, D_OUT=>C);
uH: Reg generic map (n=>4)
port map (D_IN=>F_ADD, SI=>SO_C, CLK=>CLK, RST=>RST, SLOAD=>SL_H, ENABLE=>EN_H,
SO=>SO_H, D_OUT=>H);
uL: Reg generic map (n=>4)
port map (D_IN=>(n-1 downto 0=>'0'), SI=>SO_H, CLK=>CLK, RST=>RST, SLOAD=>SL_L,
ENABLE=>EN_L, SO=>SO_L, D_OUT=>Low);
uCTRL: CtrlLogic generic map (n=>4)
port map (Rst=>RST, CLK=>CLK, SL_A=>SL_A, SL_B=>SL_B, SL_H=>SL_H, SL_L=>SL_L,
SL_C=>SL_C, EN_A=>EN_A, EN_B=>EN_B, EN_H=>EN_H, EN_L=>EN_L, EN_C=>EN_C);

High<=H;

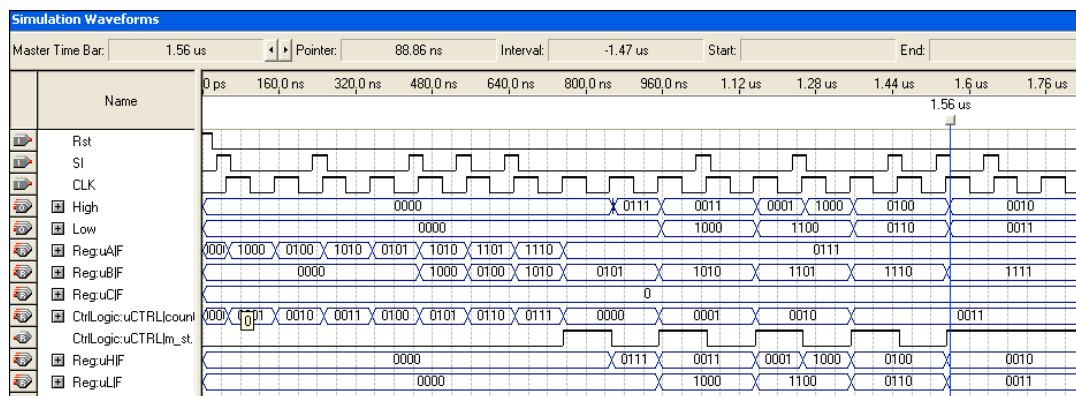
```

Ας δούμε πως διασυνδέονται οι οντότητες μεταξύ τους. Τα σήματα που έχουν δηλωθεί εξυπηρετούν μόνο τον σκοπό της διασύνδεσης των οντοτήτων οπότε η χρήση τους θα φανεί μέσα από την επεξήγηση της διασύνδεσης των οντοτήτων. Αρχικά ορίζουμε τους δύο καταχωρητές A, B μεγέθους 4 τεσσάρων δυαδικών ψηφίων ο κάθε ένας, με ετικέτες uA, uB. Τα σήμα SL_A, SL_B ελέγχουν την σειριακή φόρτωση τους. Ο καταχωρητής A δέχεται ως σειριακή είσοδο την σειριακή είσοδο του πολλαπλασιαστή SI ενώ ο καταχωρητής B δέχεται ως σειριακή είσοδο την σειριακή έξοδο του A. Έτσι ουσιαστικά διασυνδέονται οι δύο καταχωρητές και φορτώνονται σειριακά σαν να είναι ένας. Προσέξτε ότι τις θύρες παράλληλης εισόδου τους τις συνδέουμε μόνιμα στο 0 αφού δεν χρησιμοποιούνται ποτέ. Οι παράλληλες εξοδοι τους ονομάζονται A και B.

Οι καταχωρητές που αποθηκεύουν το αποτέλεσμα είναι οι uH (High), uL (Low) για το υψηλό και χαμηλό nibble του αποτελέσματος. Επίσης ο καταχωρητής uC κρατάει το κρατούμενο της πρόσθεσης. Προσέξτε από την συνδεσμολογία των τριών αυτών καταχωρητών ότι η σειριακή έξοδος του uC περνάει στην σειριακή είσοδο του uH, η σειριακή έξοδος του uH περνάει στην σειριακή είσοδο του uL. Έτσι, σε κάθε κύκλο ολίσθησης που έπεται κάθε πρόσθεση ολισθαίνουν και οι τρεις καταχωρητές μαζί (δείτε τον αλγόριθμο του πολλαπλασιασμού στην θεωρία). Παρατηρήστε ότι τα σήματα C, COUT δηλώνονται σαν διανύσματα μοναδιαίου μεγέθους (std_logic_vector(0 downto 0)) και όχι σαν απλά σήματα std_logic. Αυτό γίνεται γιατί πρέπει να διατηρηθεί η συμβατότητα με την δήλωση του καταχωρητή όπως δηλώθηκε σαν οντότητα. Θα μπορούσαμε φυσικά να ορίσουμε μία νέα οντότητα ειδικά για τον καταχωρητή κρατουμένου, αλλά προτιμήσαμε να κρατήσουμε τον αριθμό των οντοτήτων ελάχιστο.

Μια άλλη οντότητα που χρησιμοποιούμε είναι εκείνη του αθροιστή ο οποίος δέχεται ως εισόδους την έξοδο του uA και του uH και προωθεί το αποτέλεσμα της πρόσθεσης στον καταχωρητή uH, και το κρατούμενο στον καταχωρητή uC. Τέλος η τιμή του καταχωρητή uH ανατίθεται στην θύρα εξόδου High. Το ίδιο δεν συμβαίνει και με την θύρα εξόδου Low η οποία συνδέεται άμεσα στην έξοδο του καταχωρητή uL. Γιατί υπάρχει αυτή η διαφοροποίηση; Εξηγήστε στην αναφορά σας.

Εισάγετε την παραπάνω περιγραφή στο κατάλληλο αρχείο VHDL το οποίο και θα ορίσετε ως κορυφαία οντότητα της σχεδίασης σας. Εκτελέστε εξομοίωση με τον πολλαπλασιασμό $7 \times 5 = 35$. Παρακολουθήστε τα κατάλληλα σήματα όπως αυτά που φαίνονται στην Εικόνα 41. Δείτε στην αναφορά σας γιατί το αποτέλεσμα του πολλαπλασιασμού είναι σωστό. Αναλύστε κάθε κύκλο ρολογιού και δείτε ότι τα σήματα του πολλαπλασιαστή είναι σωστά και όπως τα περιγράψαμε. Εκτελέστε στατική χρονική ανάλυση και βρείτε την μέγιστη επιτρεπτή συχνότητα λειτουργίας.



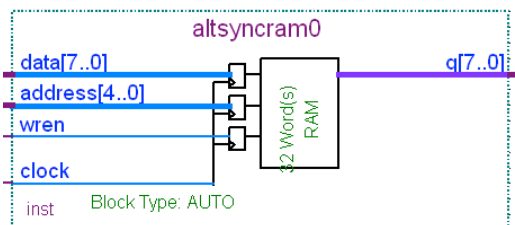
Εικόνα 41. Αποτελέσματα Πολλαπλασιασμού

Υλοποίηση Πολλαπλασιαστή (στο εργαστήριο)

Τελευταίο βήμα της άσκησης είναι η υλοποίηση του πολλαπλασιαστή στο board. Χρησιμοποιήστε ένα από τα Push-buttons για ρολόι και ένα από τα dip switches για την σειριακή είσοδο δεδομένων. Προσέξτε: όταν πιέζετε το push-button πρέπει να λαμβάνει χώρα η θετική ακμή. Στο LCD απεικονίστε τα περιεχόμενα των καταχωρητών uA, uB, uC, uH, uL. Ελέγξτε την λειτουργία του πολλαπλασιαστή δίνοντας δεδομένα από τα dip switches και διαδοχικούς παλμούς από το push button που χρησιμοποιείται ως ρολόι.

Εργαστηριακή Άσκηση 6: Σχεδίαση με VHDL και μνήμες

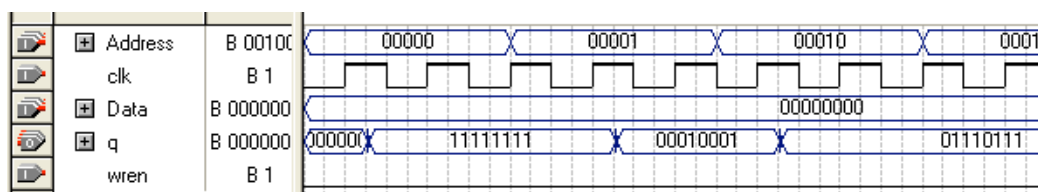
Στην 6^η εργαστηριακή άσκηση θα μάθουμε πώς να χρησιμοποιούμε μνήμη η οποία συνήθως απαιτείται σε πιο περίπλοκες σχεδιάσεις. Κατόπιν θα δούμε ένα παράδειγμα σχεδίασης με VHDL.



Εικόνα 42. RAM

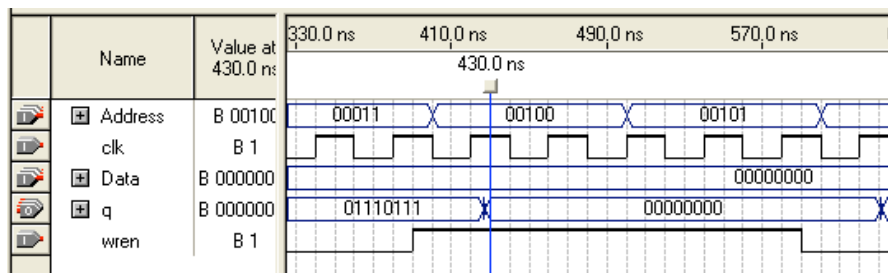
Χρησιμοποιώντας μνήμη

Μία από τις πλέον απαραίτητες μονάδες σε ένα περιβάλλον σχεδίασης είναι η χρησιμοποιούμενη μνήμη. Λέγοντας μνήμη φυσικά δεν εννοούμε καταχωρητές και flip flops τα οποία μπορεί να είναι τμήμα της σχεδίασης μας αλλά μία χωριστή μονάδα η οποία απαρτίζεται από με πολλές λέξεις συγκεκριμένου μεγέθους. Το εργαλείο σχεδίασης Quartus δίνει πολλές δυνατότητες για σχεδίαση μνήμης. Ας δούμε όμως λίγο πιο προσεκτικά πως χρησιμοποιείται η μνήμη.



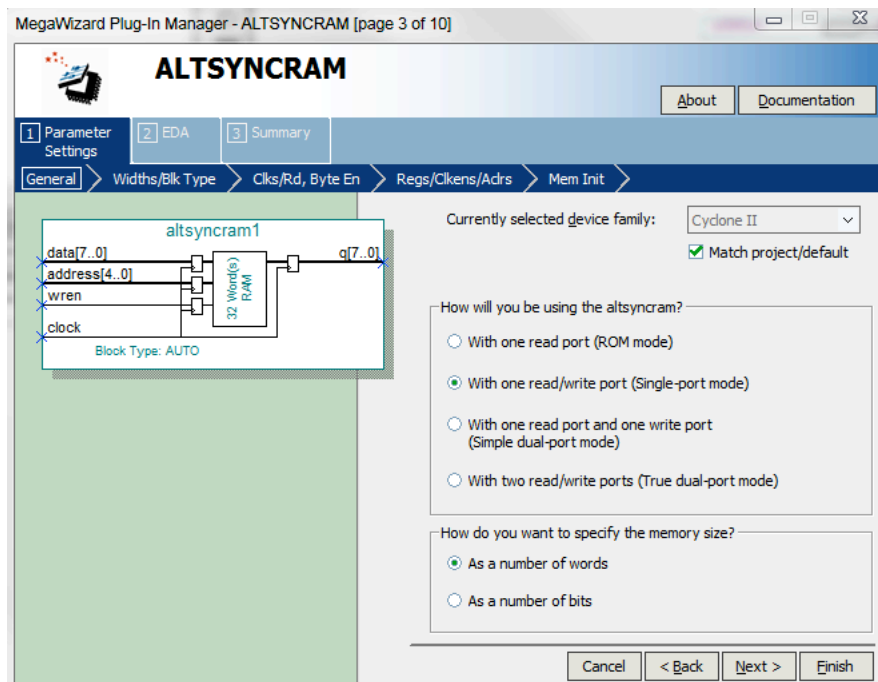
Εικόνα 43. Κύκλος ανάγνωσης

Στην Εικόνα 42 φαίνεται το interface της μνήμης RAM που διαθέτει η προγραμματιζόμενη συσκευή μας. Η συγκεκριμένη μνήμη αποτελείται από 32 λέξεις των 8 δυαδικών ψηφίων η κάθε μία. Για τον λόγο αυτό η διεύθυνση *address* αποτελείται από 5 δυαδικά ψηφία, και τα δεδομένα *data* από 8. Το σήμα *clock* χρησιμοποιείται μόνο για να κλειδώσει στους καταχωρητές *Memory Address Register* (MAR) και *Memory Data Register* (MDR) την διεύθυνση και τα δεδομένα προς εγγραφή αντίστοιχα. Η έξοδος *q[7..0]* παρέχει το περιεχόμενο της θέσης της οποίας η διεύθυνση έχει τοποθετηθεί στον MAR. Ας δούμε όμως πιο προσεκτικά το διάγραμμα χρονισμού της μνήμης. Έστω ότι στις θέσεις 0,1 και 2 έχουν αποθηκευτεί τα δεδομένα FF, 11 και 77. Για να διαβάσουμε αυτά τα περιεχόμενα της μνήμης θα πρέπει να κλειδώσουμε την διεύθυνση τους στον MAR οπότε άμεσα τα περιεχόμενα αυτά θα εμφανιστούν στην έξοδο *q*. Όπως φαίνεται και στην Εικόνα 43 τοποθετούμε την απαιτούμενη διεύθυνση στον δίαυλο *Address* και στην επόμενη άνοδο του ρολογιού η διεύθυνση αυτή κλειδώνεται στον MAR οπότε λίγο μετά εμφανίζεται το περιεχόμενο της θέσης μνήμης στην έξοδο *q*. Εάν για παράδειγμα θέλαμε να διαβάσουμε το περιεχόμενο μίας θέσης της μνήμης με κάποιο κύκλωμα τότε θα έπρεπε να τοποθετήσουμε την διεύθυνση στον δίαυλο *Address*, και να περιμένουμε την 2^η στη σειρά άνοδο του ρολογιού ώστε να θεωρήσουμε τα δεδομένα στο δίαυλο *q* διαθέσιμα (προσέξτε το αυτό γιατί θα μας χρειαστεί παρακάτω).



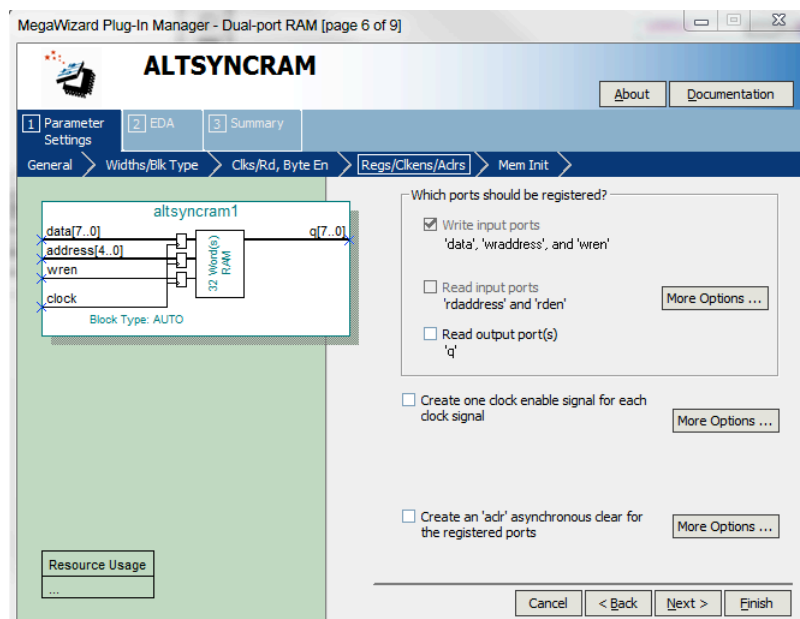
Εικόνα 44. Κύκλος εγγραφής

Για να εγγράψουμε δεδομένα σε μία θέση μνήμης θα πρέπει να ακολουθήσουμε μία ανάλογη τακτική. Αυτή την φορά θα πρέπει να τοποθετήσουμε τόσο στον δίαυλο διευθύνσεων Address όσο και στον δίαυλο δεδομένων Data την διεύθυνση και το περιεχόμενο προς εγγραφή αντίστοιχα πριν την θετική άνοδο του ρολογιού. Ταυτόχρονα ενεργοποιούμε και το σήμα wren ώστε να υποδείξουμε ότι πρόκειται για εγγραφή. Όπως παρατηρούμε και στην Εικόνα 44 όπου η τιμή 00 γράφεται στην θέση μνήμης 4 παρατηρούμε ότι αμέσως μετά την θετική άνοδο του ρολογιού τα δεδομένα εμφανίζονται στην έξοδο q πράγμα που σημαίνει ότι η εγγραφή ολοκληρώθηκε (προσέξτε ότι στην δική σας περίπτωση η καθυστέρηση μπορεί να είναι διαφορετική).



Εικόνα 45.

Ας δούμε τώρα πως μπορούμε να εισάγουμε μία τέτοια μνήμη στην σχεδίαση μας. Η μνήμη που χρησιμοποιήσαμε είναι η Megafunction Altsyncram. Δημιουργούμε ένα νέο σχηματικό στα πλαίσια ενός νέου project (ΔΕΝ ΞΕΧΝΑΜΕ ΠΟΤΕ ΟΛΕΣ ΤΙΣ ΡΥΘΜΙΣΕΙΣ ΠΟΥ ΕΧΟΥΜΕ ΑΝΑΦΕΡΕΙ ΣΤΙΣ ΠΡΟΗΓΟΥΜΕΝΕΣ ΑΣΚΗΣΕΙΣ). Εισάγουμε μία Megafunction Altsyncram χρησιμοποιώντας το Megafunction Wizard για την παραμετροποίηση. Στο πρώτο βήμα (Εικόνα 45) δηλώνουμε αριθμό και τύπο θυρών καθώς και τον τρόπο προσδιορισμού του μεγέθους της μνήμης. Επιλέγουμε *Single Port Mode* και *As a number of words* αντίστοιχα. Στο επόμενο βήμα ορίζουμε τον αριθμό των λέξεων (το μέγεθος των οποίων είναι προκαθορισμένο 8 bits) και επίσης το μέγεθος του διαύλου δεδομένων όσο και το μέγεθος της κάθε λέξης (8).



Εικόνα 46.

Στο επόμενο βήμα ορίζουμε ένα ρολόι κοινό για είσοδο-έξοδο (ωστόσο στην έξοδο δεν θα μας χρειαστεί ρολόι). Ο τρόπος χρονισμού της μνήμης καθορίζεται και στο επόμενο βήμα το οποίο φαίνεται στην Εικόνα 46. Στο πεδίο «which port should be registered» καθορίζουμε εάν θα έχουμε καταχωρητή στην διεύθυνση ή/και στα αποτελέσματα (MAR/MDR). Όπως παρατηρούμε ο καταχωρητής στην διεύθυνση είναι υποχρεωτικός ενώ εκείνος στην έξοδο δεν είναι. Η διαφορά είναι ότι εάν βάλουμε και στην έξοδο καταχωρητή, τότε τα δεδομένα θα κλειδώνονται πριν βγουν στην έξοδο αλλά θα καθυστερούν έναν κύκλο ρολογιού παραπάνω. Καθώς δεν θέλουμε να συμβαίνει κάτι τέτοιο, απενεργοποιούμε αυτή την δυνατότητα.

Στο επόμενο βήμα καθορίζουμε εάν θέλουμε να χρησιμοποιήσουμε αρχείο αρχικοποίησης της μνήμης. Το αρχείο αυτό έχει κατάληξη .mif. Τα περιεχόμενα του φαίνονται ακόλουθα:

```
DEPTH = 32;
WIDTH = 8;
CONTENT
BEGIN
    00      : FF;
    01      : 11;
    [02..1F] : 77;
END;
```

Στην συγκεκριμένη περίπτωση αρχικοποιήσαμε την θέση 00 με την τιμή FF, την θέση 01 με την τιμή 11 και τις υπόλοιπες με την τιμή 77.

Εκτελέστε χρονική εξομοίωση και επιβεβαιώστε την λειτουργία της μνήμης που δημιουργήσατε. Κατόπιν σχεδιάστε το κατάλληλο κύκλωμα το οποίο θα χρησιμοποιεί το pushbutton KEY0 ως ρολόι και τα DipSwitches SW0-SW3 ως 4-bit διεύθυνση και θα εμφανίζει στο LCD Display τα περιεχόμενα της θέσης μνήμης της οποίας η διεύθυνση έχει τοποθετηθεί στα DipSwitches (το πιο σημαντικό bit συνδέεται μόνιμα στο 0). Η μνήμη αρχικοποιείται στις τιμές FF, EE, DD, CC, BB, AA, 99, 88, 77, 66, 55, 44, 33, 22, 11, 00, 00, 00, ... με την χρήση του κατάλληλου αρχείου. Δείτε τα αποτελέσματα στον επιτηρητή σας.

Σχεδίαση με VHDL

Ένα από τα σημαντικότερα πλεονεκτήματα των σύγχρονων εργαλείων σχεδίασης είναι οι δυνατότητες που προσφέρουν για σχεδίαση πολύπλοκων συστημάτων. Η πολυπλοκότητα των σύγχρονων σχεδιασμών μπορεί να αντιμετωπιστεί μόνο με χρήση γλωσσών περιγραφής υψηλού

επιπέδου. Σε αυτή την ενότητα θα δούμε πως μπορούμε να σχεδιάσουμε ένα κύκλωμα χρησιμοποιώντας την γλώσσα περιγραφής VHDL.

Θα σχεδιάσουμε ένα κύκλωμα το οποίο θα διατάσσει σε αύξουσα σειρά τα περιεχόμενα μονοδιάστατου διανύσματος το οποίο βρίσκεται αποθηκευμένο σε μνήμη RAM 256x8 χρησιμοποιώντας τον αλγόριθμο διάταξης φυσαλίδας (BubbleSort). Η μνήμη RAM θα δημιουργηθεί με την διαδικασία που περιγράψαμε στην προηγούμενη ενότητα ενώ το κύκλωμα που εκτελεί την διάταξη θα περιγραφεί στην γλώσσα VHDL. Πριν ξεκινήσουμε την περιγραφή θα πρέπει να σκιαγραφήσουμε την αρχιτεκτονική του συστήματος που θέλουμε να σχεδιάσουμε. Με αυτόν τον τρόπο θα μπορούσαμε να σκεφτούμε σαν σχεδιαστές συστημάτων υλικού και όχι ως συγγραφείς λογισμικού.

Ας δούμε τώρα τι πρέπει να κάνει το σύστημα που θα σχεδιάσουμε. Αρχικά έχουμε ως δεδομένο ότι υπάρχει μνήμη που περιέχει τα δεδομένα προς διάταξη. Το κύκλωμα ελέγχου θα πρέπει να διαβάσει σε διαδοχικά ζεύγη τα δεδομένα αυτά και να τα διατάσει σύμφωνα με τον παρακάτω αλγόριθμο:

```
do
    flag=0;
    for i = 0 to 254
        read a(i), read a(i+1)
        if (a[i]>a[i+1]) then a[i]↔a[i+1], flag=1;
    end for
while flag=1;
```

Για να μπορέσουμε να περιγράψουμε σε hardware αυτόν τον αλγόριθμο θα πρέπει να σκεφτούμε με λίγο διαφορετικό τρόπο από αυτόν που σκεφτόμαστε όταν προγραμματίζουμε. Το πρώτο πρόβλημα που έχουμε είναι η διαχείριση της μνήμης η οποία όπως είδαμε στην προηγούμενη ενότητα έχει συγκεκριμένο πρωτόκολλο επικοινωνίας. Για να μπορέσουμε να συγκρίνουμε τις θέσεις $a[i]$, $a[i+1]$ πρέπει καταρχήν να φέρουμε τα δεδομένα από την μνήμη, να τα αποθηκεύσουμε σε δύο καταχωρητές (dataA, dataB) και κατόπιν να τα συγκρίνουμε. Άρα θα πρέπει να εκτελέσουμε ανάγνωση από την μνήμη δύο φορές. Σε αυτό το σημείο θα πρέπει να θυμηθούμε ότι υπάρχει το ρολόι που συντονίζει όλη την διαδικασία και με βάση αυτό γίνεται η προσπέλαση. Επομένως η κάθε ανάγνωση διαρκεί κάποιους κύκλους ρολογιού τους οποίους εμείς θα πρέπει να λάβουμε υπόψη μας. Έστω λοιπόν ότι ένας καταχωρητής με το όνομα *count* περιέχει την τιμή του i . Για να γίνει μία ανάγνωση από την μνήμη θα πρέπει να αποσταλεί στον δίαυλο διευθύνσεων η τιμή που βρίσκεται στον καταχωρητή *count*, οπότε με την ερχόμενη άνοδο του ρολογιού θα κλειδωθεί στον καταχωρητή *MAR* και τα δεδομένα θα εμφανιστούν στην έξοδο της μνήμης RAM. Έτσι μπορούμε να αποθηκεύσουμε τα δεδομένα αυτά στην μεθ-επόμενη θετική ακμή του ρολογιού σε κάποιον εσωτερικό καταχωρητή. Παρατηρούμε λοιπόν ότι υπάρχει μία ακολουθία κινήσεων και καταστάσεων που συμπληρώνουν την ανάγνωση από την μνήμη. Έστω λοιπόν οι καταστάσεις *SendAddressA_r* και *GetA* κατά τις οποίες μεταφέρουμε το περιεχόμενο της θέσης μνήμης $a[i]$ στον καταχωρητή *dataA* ($a[i] \rightarrow \text{dataA}$). Για να ξεκινήσουμε την ανάγνωση μεταβαίνουμε στην κατάσταση *SendAddressA_r* κατά την οποία στέλνουμε την τιμή του καταχωρητή *count* στον δίαυλο διευθύνσεων *Address*. Η επόμενη κατάσταση είναι η *GetA* κατά την οποία περιμένουμε τα δεδομένα να έρθουν από την μνήμη. Η μετάβαση από κατάσταση σε κατάσταση γίνεται πάντα στην θετική ακμή του ρολογιού. Στην επόμενη λοιπόν θετική ακμή του ρολογιού η νέα κατάσταση είναι η *GetA*. Προσέξτε τώρα ότι ο *MAR* έχει κλειδώσει την διεύθυνση και πριν έρθει η επόμενη θετική ακμή τα δεδομένα της αντίστοιχης θέσης μνήμης θα εμφανιστούν στον έξοδο της RAM. Άρα αυτός ο κύκλος δεν μπορεί να αξιοποιηθεί και απλά θα πρέπει να περιμένουμε την επόμενη θετική ακμή για να αποθηκεύσουμε τα δεδομένα στον καταχωρητή *dataA*. Έτσι χρησιμοποιούμε ένα μετρητή 1-bit για να παραμείνουμε σε αυτήν την κατάσταση για άλλον έναν κύκλο. Ο μετρητής αυτός (*delay*) έχει αρχικά την τιμή 0 και μόλις περάσουμε στην κατάσταση *GetA* παίρνει την τιμή 1. Έτσι για να μεταβούμε στην επόμενη κατάσταση θα πρέπει η τιμή του *delay* να είναι 1. Αυτό δεν συμβαίνει την πρώτη φορά που μπαίνουμε στην κατάσταση *GetA* αλλά στον επόμενο κύκλο. Όταν πλέον το *delay* είναι 1 και βρισκόμαστε στην κατάσταση *GetA* αποθηκεύουμε τα δεδομένα που διαβάστηκαν από την RAM, αυξάνουμε την τιμή του μετρητή *count* ώστε να μπορέσουμε να διαβάσουμε και την θέση $a[i+1]$ και προχωράμε στην επόμενη

κατάσταση η οποία είναι η `SendAddressB_r`. Με τον ίδιο τρόπο διαβάζουμε τα δεδομένα από την `a[i+1]` θέση χρησιμοποιώντας τις καταστάσεις `SendAddressB_r`, `GetB`.

Έχοντας πλέον διαβάσει από την μνήμη τα δεδομένα `a[i]→dataA`, `a[i+1]→dataB` τα συγκρίνουμε στην κατάσταση `Compare` και εάν πρέπει να εναλλαχθούν θέτουμε την σημαία `flag=1` σύμφωνα με τον αλγόριθμο, μειώνουμε την τιμή του μετρητή `count` κατά 1 ώστε να δεικτοδοτήσουμε την θέση μνήμης `a[i]` και μεταβαίνουμε στην κατάσταση `SendAddrB_w` κατά την οποία θα στείλουμε τα περιεχόμενα του καταχωρητή `dataB` στην θέση μνήμης `a[i]` για να επιτύχουμε την εναλλαγή. Η εγγραφή γίνεται ανάλογα με την ανάγνωση, με την διαφορά όμως ότι πρέπει να ενεργοποιήσουμε το σήμα εγγραφής `WR`. Επιπλέον δεν χρειάζεται να καθυστερήσουμε έναν κύκλο αφού δεν χρειάζεται να περιμένουμε δεδομένα από την μνήμη. Έτσι έχουμε δύο καταστάσεις: α) `SendAddrB_w` κατά την οποία τοποθετούμε την τιμή του μετρητή `count` στον δίαυλο διευθύνσεων (προκειμένου να κλειδωθεί στον `MAR` στην επόμενη θετική ακμή) και το περιεχόμενο του καταχωρητή `dataB` στον δίαυλο δεδομένων και μεταβαίνουμε στην επομένη κατάσταση η οποία είναι η β) `WriteB` κατά την οποία ο `MAR` αποθηκεύει την διεύθυνση και ενεργοποιείται το σήμα `WR` έτσι ώστε στην επόμενη θετική ακμή να αποθηκευτεί στην μνήμη το περιεχόμενο του `dataB` το οποίο βρίσκεται στον δίαυλο δεδομένων. Παράλληλα στην κατάσταση `WriteB` αυξάνουμε το περιεχόμενο του μετρητή `count` κατά 1 προκειμένου να δεικτοδοτήσουμε την επόμενη θέση μνήμης στην οποία θα τοποθετηθεί το περιεχόμενο του καταχωρητή `dataA` χρησιμοποιώντας δύο ανάλογες καταστάσεις οι οποίες είναι οι `SendAddrA_w`, `WriteA`.

Ολοκληρώνοντας με την εναλλαγή θα πρέπει να ελέγξουμε εάν έχει ολοκληρωθεί η τρέχουσα επανάληψη, δηλαδή εάν ο μετρητής `count` έχει φθάσει στην μέγιστη τιμή του και σε τέτοια περίπτωση θα μεταβούμε στην κατάσταση `CheckFlag` αλλιώς επανερχόμαστε στην αρχική κατάσταση `SendAddrA_r`. Στην κατάσταση `CheckFlag` ελέγχουμε εάν έχει τεθεί η σημαία οπότε μηδενίζουμε τον μετρητή και μεταβαίνουμε στην κατάσταση `SendAddrA_r` αλλιώς τίθεται το σήμα `Complete` που δηλώνει την ολοκλήρωση της διαδικασίας.

Από τα παραπάνω καταλαβαίνουμε ότι ουσιαστικά πρόκειται για μία μηχανή καταστάσεων η οποία έχει σαν βάση το ρολόι και με την βοήθεια του φροντίζει να συντονίζει την σειρά των ενεργειών. Με αυτόν τον τρόπο θα σχεδιάζουμε οποιοδήποτε σύστημα απαιτεί ακολουθία ενεργειών όπως το παραπάνω. Ας δούμε όμως και την υλοποίηση από την πλευρά του εργαλείου Quartus.

Δημιουργούμε ένα νέο project και ένα νέο αρχείο με κατάληξη `.vhd`. Όταν ανοίξει ο κειμενογράφος εισάγουμε τον κώδικα που επιθυμούμε. Το Quartus επιταχύνει την διαδικασία σχεδίασης παρέχοντας πρότυπα VHDL. Με δεξί κλικ ποντικιού επιλέγουμε *Insert Template* και κατόπιν επιλέγουμε από το συντακτικό της VHDL την επιλογή *Entity Declaration*. Τότε εμφανίζεται στον κειμενογράφο το συντακτικό της δήλωσης οντότητας από την οποία λείπουν τα συγκεκριμένα στοιχεία της οντότητας που θέλουμε να εισάγουμε. Με όμοιο τρόπο μπορούμε να εισάγουμε και σώμα αρχιτεκτονικής. Ο κώδικας για την περιγραφή του παραπάνω συστήματος φαίνεται στην Εικόνα 47. Αρχικά ορίζουμε τον καταχωρητή κατάστασης `state` ο οποίος παίρνει τιμές τύπου απαρίθμησης, τον μετρητή `count`, τους καταχωρητές δεδομένων `dataA`, `dataB`, το σήμα σημαίας και διάφορα άλλα βοηθητικά σήματα. Η αρχιτεκτονική αποτελείται από δύο process οι οποίες έχουν την ευθύνη α) της μετάβασης από κατάσταση σε κατάσταση και β) της εκτέλεσης των βημάτων ελέγχου αντίστοιχα. Φροντίζουμε να υπάρχει σήμα αρχικοποίησης `Reset` ασύγχρονο (έξω από το ρολόι) καθώς επίσης σήμα εκκίνησης ταξινόμησης `launch`. Επίσης παρατηρούμε ότι όσο το κύκλωμα ταξινομεί τον πίνακα, το σήμα `Complete` είναι στο '1' ενώ όσο βρίσκεται σε αναμονή στην κατάσταση `Waiting` το σήμα `Complete` είναι στο '0'. Τέλος, εκτός από τις δύο process υπάρχουν και αναθέσεις ροής δεδομένων (συνδυαστική λογική) για διάφορα σήματα που είναι απαραίτητα. Προσέξτε ότι εσκεμμένα έχει γίνει μία παραβίαση του αλγορίθμου, και η τελευταία τιμή του μετρητή δεν είναι η "11111111" όπως λέει η εκφώνηση αλλά η "00000111" έτσι ώστε να μπορούμε να κάνουμε εξομοίωση για ταξινόμηση 8 αριθμών και όχι 256. Όταν αργότερα βεβαιωθούμε για την σωστή λειτουργία θα αλλάξουμε την τιμή αυτή για να προγραμματίσουμε την συσκευή FPGA στο board.

Εκτελούμε compilation, επιβεβαιώνουμε ότι δεν υπάρχουν λάθη και δημιουργούμε ένα σύμβολο. Το επόμενο βήμα είναι η προσθήκη μνήμης. Δημιουργούμε ένα νέο σχηματικό και εισάγουμε το σύμβολο του κυκλώματος ταξινόμησης καθώς και το σύμβολο της μνήμης 256x8 όπως έχουμε περιγράψει. Η διασύνδεση γίνεται με τον τρόπο που φαίνεται στην Εικόνα 48. Αρχικοποιούμε τις 8

πρώτες θέσεις της μνήμης 00-07 ως εξής: FF, FE, FD, FC, FB, FA, F9, F8 αντιστοίχως. Εκτελούμε εξομοίωση χρησιμοποιώντας περίοδο ρολογιού 40ns (θέτουμε αρχικά το σήμα Reset στο '1' για 10ns και κατόπιν το αφήνουμε μόνιμα στο '0'). Ξεκινούμε την ταξινόμηση με χρήση του σήματος launch το οποίο πρέπει να τοποθετηθεί για λίγο στο '1'. Μπορούμε να παρακολουθούμε οποιοδήποτε εσωτερικό σήμα ή καταχωρητή θέλουμε. Για την συγκεκριμένη περίπτωση θα παρακολουθήσουμε τα σήματα/καταχωρητές: *Reset, clk, launch, address, data, count, dataA, dataB, Flag, state, Complete*.

Η εξομοίωση μπορεί να βοηθήσει στην διόρθωση λαθών σχεδίασης αλλά συνήθως είναι δύσκολο να την παρακολουθήσει κανείς από την αρχή ως το τέλος για να βεβαιωθεί ότι το κύκλωμα λειτουργεί σωστά. Ένας γρήγορος τρόπος είναι να περιμένουμε έως ότου ολοκληρωθεί η ταξινόμηση (Complete=1) και κατόπιν να ελέγξουμε την μνήμη ώστε να δούμε εάν έγινε σωστά η ταξινόμηση. Σε αυτή την περίπτωση πρέπει να ακολουθήσουμε την διαδικασία της εκσφαλμάτωσης (Debug). Θα ορίσουμε αρχικά ένα breakpoint στο σημείο το οποίο γίνεται το σήμα Complete ίσο με 1. Για αυτό εκτελούμε *Processing>Simulation Debug>Breakpoints*.

```

LIBRARY IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

ENTITY Sort IS
    PORT
    (
        clk, launch, reset : IN    STD_LOGIC;
        DataIn : IN    STD_LOGIC_VECTOR(7 DOWNTO 0);
        Address, DataOut : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0);
        Complete, WR : OUT    STD_LOGIC
    );
END Sort;

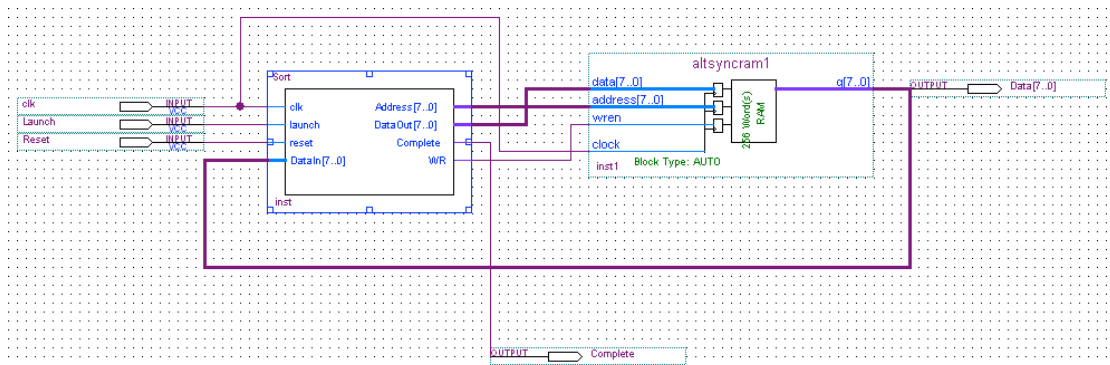
ARCHITECTURE RTL OF Sort IS
    type state_type is (Waiting, SendAddrA_r, GetA, SendAddrB_r, GetB, Compare, SendAddrA_w, WriteA, SendAddrB_w, WriteB, CheckLoop, CheckFlag);
    SIGNAL state : state_type:=Waiting;
    signal count : std_logic_vector(7 DOWNTO 0);
    signal delay : std_logic;
    signal dataA, dataB: std_logic_vector(7 DOWNTO 0);
    signal Flag, Swap, CountEnd : std_logic;
BEGIN
    process (clk, reset)
    begin
        if (reset='1') then
            state<=Waiting;
        elsif (clk'event and clk='1') then
            case state is
                when Waiting => if (launch='1') then state<=SendAddrA_r; end if;
                when SendAddrA_r => state<=GetA; delay<='0';
                when GetA => if (delay='1') then state<=SendAddrB_r; else delay<='1'; end if;
                when SendAddrB_r => state<=GetB; delay<='0';
                when GetB => if (delay='1') then state<=Compare; else delay<='1'; end if;
                when Compare => if Swap='1' then state<=SendAddrB_w; else state<=CheckLoop; end if;
                when SendAddrB_w => state<=WriteB;
                when WriteB => state<=SendAddrA_w;
                when SendAddrA_w => state<=WriteA;
                when WriteA => state<=CheckLoop;
                when CheckLoop => if (CountEnd='1') then state<=CheckFlag; else state<=SendAddrA_r; end if;
                when CheckFlag => if (Flag='0') then state<=Waiting; else state<=SendAddrA_r; end if;
            end case;
        end if;
    end process;

    process (clk)
    begin
        if (clk'event and clk='1') then
            case state is
                when Waiting => count<="00000000"; Flag<='0';
                when SendAddrA_r => Address<=count;
                when GetA => if (delay='1') then dataA<=DataIn; count<=count+1; end if;
                when SendAddrB_r => Address<=count;
                when GetB => if (delay='1') then dataB<=DataIn; end if;
                when Compare => if Swap='1' then count<=count-1; Flag<='1'; end if;
                when SendAddrB_w => Address<=count; DataOut<=dataB;
                when WriteB => count<=count+1;
                when SendAddrA_w => Address<=count; DataOut<=dataA;
                when WriteA => null;
                when CheckLoop => null;
                when CheckFlag => if (Flag='1') then Flag<='0'; count<="00000000"; end if;
            end case;
        end if;
    end process;

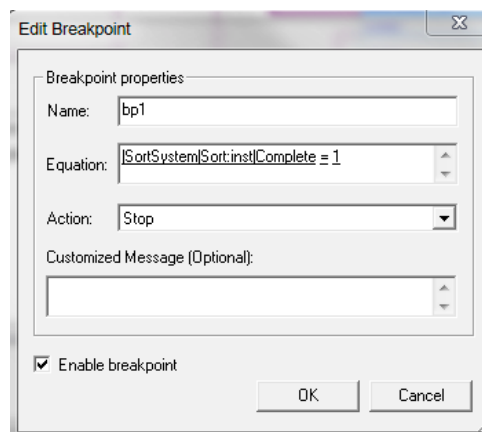
    WR<='1' when state=WriteB or state=WriteA else '0';
    Swap<='1' when dataA>dataB else '0';
    CountEnd<='1' when count="00000111" else '0';
    Complete<='1' when state=Waiting else '0';
END RTL;

```

Εικόνα 47. Περιγραφή κυκλώματος ταξινόμησης



Εικόνα 48.



Εικόνα 49. Breakpoints

Αφού επιλέξουμε την επιλογή "New" ανοίγει το παράθυρο που φαίνεται στην Εικόνα 49. Επιλέγουμε το "Condition" στο πεδίο *Equation* και εισάγουμε την συνθήκη μας. Επιλέγουμε ως *node* το σήμα *Complete*, ως *operator* το '=' και ως τιμή το '1'. Πριν εκτελέσουμε εξομοίωση πρέπει να αυξήσουμε τον χρόνο εξομοίωσης στο 1ms (*Edit>End Time*).

SortSystem altsyncram1:inst1 altsy...								
Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	248	249	250	251	252	253	254	255
8	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0

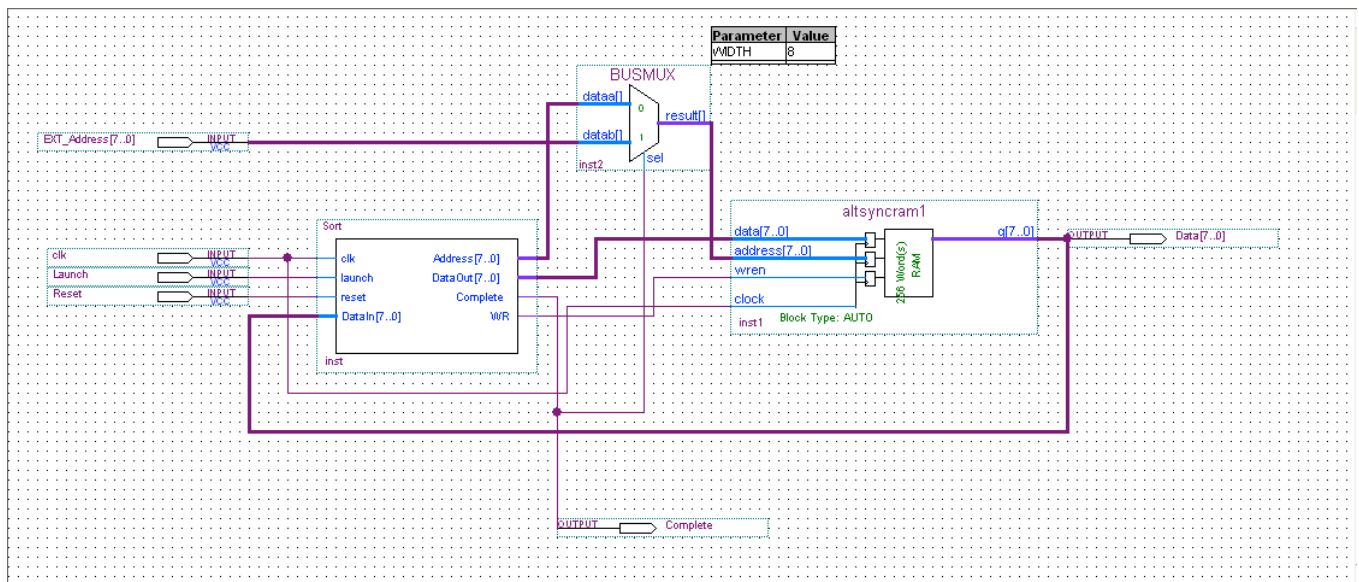
Εικόνα 50. Περιεχόμενα RAM

Εκτελούμε εξομοίωση και το εργαλείο σταματά όταν το σήμα *Complete* γίνεται 1. Τότε εκτελούμε την εντολή *Processing>Simulation Debug>Embedded Memory* και ανοίγει το παράθυρο που φαίνεται στην Εικόνα 50 το οποίο δείχνει τα περιεχόμενα της μνήμης. Παρατηρούμε ότι τα περιεχόμενα των θέσεων 0-7 έχουν ταξινομηθεί όπως περιμέναμε.

Υλοποίηση στο board

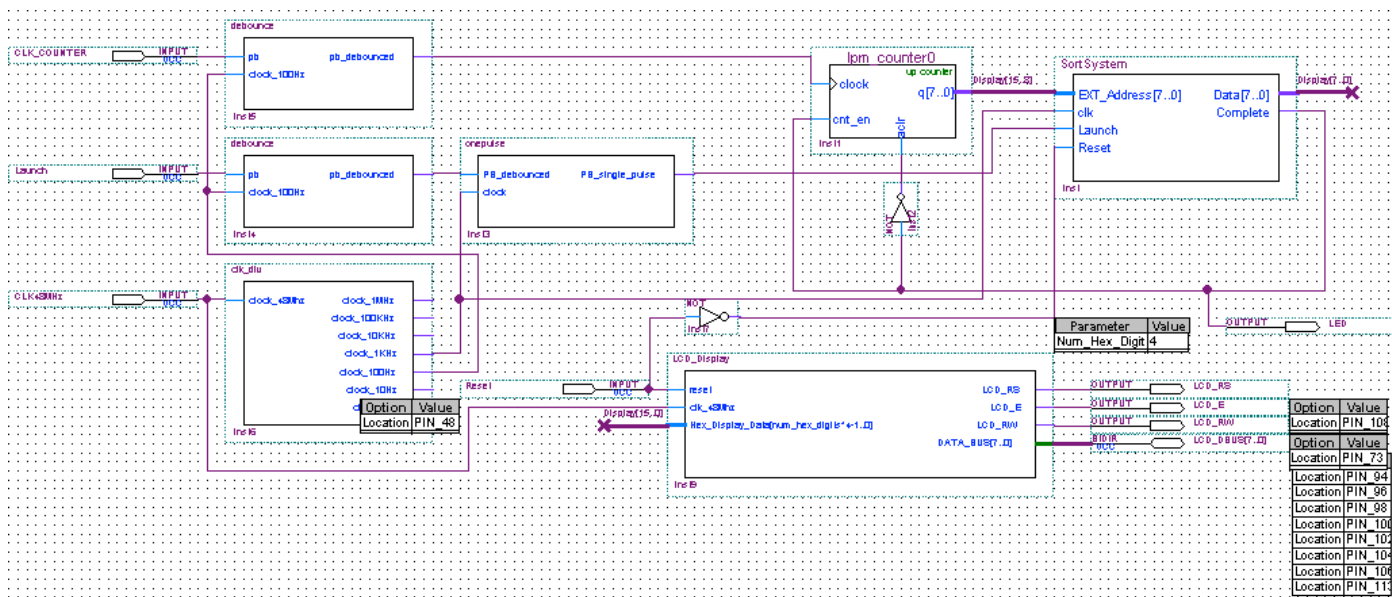
Πριν ελέγξουμε την λειτουργία του κυκλώματος στο board πρέπει να επαναφέρουμε την εντολή $CountEnd \leq '1' \text{ when } count = "00000111" \text{ else } '0'$ στην σωστή της μορφή η οποία είναι $CountEnd \leq '1' \text{ when } count = "11111111" \text{ else } '0'$ έτσι ώστε το μέγεθος του πίνακα να θεωρηθεί ως 256×8 . Επιπλέον θα πρέπει να δώσουμε την δυνατότητα οπτικού ελέγχου του αποτελέσματος όταν χρησιμοποιούμε το board. Αυτό θα γίνει με την βοήθεια μετρητή ο οποίος θα δεικτοδοτεί όλη την μνήμη και θα προκαλεί ανάγνωση των περιεχομένων της μετά την ολοκλήρωση της ταξινόμησης. Για τον λόγο αυτό χρειαζόμαστε έναν τρόπο να παρέχουμε διευθύνσεις στην μνήμη από τον μετρητή και όχι από το κύκλωμα Sort. Αυτό επιτυγχάνεται με την χρήση ενός πολυπλέκτη ο οποίος θα επιλέγει εκείνη την διεύθυνση όταν δεν ταξινομεί το σύστημα ($Complete = '1'$) ή αλλιώς την διεύθυνση που παράγεται από την μονάδα Sort όταν ταξινομεί το σύστημα ($Complete = '0'$). Ο πολυπλέκτης αυτός είναι ο BusMux (Megafunction) όπως φαίνεται στην Εικόνα 51. Εκτελούμε πάλι σύνθεση και δημιουργούμε νέο σύμβολο για τον σχεδιασμό.

Στο επόμενο βήμα δημιουργούμε ένα νέο σχηματικό στο οποίο τοποθετούμε το σύμβολο που δημιουργήσαμε στο προηγούμενο βήμα. Σχεδιάζουμε έναν μετρητή των 8 bits χρησιμοποιώντας την Megafunction lpm_counter. Επιπλέον ο μετρητής θα έχει είσοδο ασύγχρονης μηδένισης καθώς και είσοδο επίτρειψης μέτρησης (enable).



Εικόνα 51.

Συνδέουμε την έξοδο Complete στην είσοδο επίτρειψης έτσι ώστε ο μετρητής να μετρά μόνο όταν έχει ολοκληρωθεί η ταξινόμηση ($Complete=1$). Επιπλέον για να ξεκινά πάντα από την διεύθυνση 00 της μνήμης η εμφάνιση των αποτελεσμάτων συνδέουμε το ανεστραμμένο σήμα Complete στην είσοδο ασύγχρονης μηδένισης. Το ρολόι που θα ενεργοποιεί τον μετρητή θα προέρχεται από το KEY0 και το σήμα εκκίνησης launch από το KEY1.



Εικόνα 52.

Κάνουμε τις απαραίτητες συνδέσεις στο LCD Display ώστε να δίνει το ακόλουθο μήνυμα που φαίνεται στον παρακάτω πίνακα, όπου XX, YY οι διευθύνσεις-περιεχόμενα της μνήμης. Προσοχή, το σήμα Reset συνδέεται απευθείας στο LCD και ανεστραμμένο στο κύκλωμα ταξινόμησης. Επίσης συνδέουμε το σήμα *Complete* στο LED D3 για να γνωρίζουμε πότε έχει ολοκληρωθεί η ταξινόμηση. Το τελικό κύκλωμα φαίνεται στην Εικόνα 52 (ΠΡΟΣΟΧΗ: ο πυρήνας του LCD είναι διαφορετικός από εκείνον που φαίνεται στην Εικόνα 52 – αναζητήστε την επικαιροποιημένη περιγραφή του από προηγούμενη εργαστηριακή άσκηση). Αρχικοποιούμε την μνήμη RAM στις τιμές FF, FE, FD, ..., 00 και προγραμματίζουμε το FPGA (στο εργαστήριο)

A	D	D	R	E	S	S	D	A	T	A
X	X						Y	Y		

Ερώτημα (στο εργαστήριο)

Με ανάλογο τρόπο σχεδιάστε ένα κύκλωμα το οποίο θα υπολογίζει τον μέσο όρο δύο διανυσμάτων A, B 256 bytes το κάθε ένα. Τα δύο διανύσματα βρίσκονται στην ίδια μνήμη 1K σε διαδοχικές θέσεις (A:000-0FF, B:100-1FF) και το διάνυσμα μέσου όρου στις θέσεις C:200-2FF. Η πράξη του μέσου όρου θα γίνεται ως εξής:

$$C[0]=(A[0]+B[0])/2, C[1]=(A[1]+B[1])/2 \dots$$

Σε κάθε πράξη θα γίνεται αποκοπή των δεκαδικών ψηφίων, αλλά τελικά θα αποθηκεύεται στην θέση μνήμης 000 το άθροισμα όλων των δεκαδικών ψηφίων που χάθηκαν. Η άσκηση αυτή θα υλοποιηθεί στο board ανάλογα με την προηγούμενη. Δώστε την δυνατότητα οπτικού ελέγχου των αποτελεσμάτων στο board.

Παράρτημα Α – Ακροδέκτες

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]
SW[2]	PIN_P25	Toggle Switch[2]
SW[3]	PIN_AE14	Toggle Switch[3]
SW[4]	PIN_AF14	Toggle Switch[4]
SW[5]	PIN_AD13	Toggle Switch[5]
SW[6]	PIN_AC13	Toggle Switch[6]
SW[7]	PIN_C13	Toggle Switch[7]
SW[8]	PIN_B13	Toggle Switch[8]
SW[9]	PIN_A13	Toggle Switch[9]
SW[10]	PIN_N1	Toggle Switch[10]
SW[11]	PIN_P1	Toggle Switch[11]
SW[12]	PIN_P2	Toggle Switch[12]
SW[13]	PIN_T7	Toggle Switch[13]
SW[14]	PIN_U3	Toggle Switch[14]
SW[15]	PIN_U4	Toggle Switch[15]
SW[16]	PIN_V1	Toggle Switch[16]
SW[17]	PIN_V2	Toggle Switch[17]

Πίνακας 1. Dip Switches

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]

Πίνακας 2. PushButtons

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]
LEDR[2]	PIN_AB21	LED Red[2]
LEDR[3]	PIN_AC22	LED Red[3]
LEDR[4]	PIN_AD22	LED Red[4]
LEDR[5]	PIN_AD23	LED Red[5]
LEDR[6]	PIN_AD21	LED Red[6]
LEDR[7]	PIN_AC21	LED Red[7]
LEDR[8]	PIN_AA14	LED Red[8]
LEDR[9]	PIN_Y13	LED Red[9]
LEDR[10]	PIN_AA13	LED Red[10]
LEDR[11]	PIN_AC14	LED Red[11]
LEDR[12]	PIN_AD15	LED Red[12]
LEDR[13]	PIN_AE15	LED Red[13]
LEDR[14]	PIN_AF13	LED Red[14]
LEDR[15]	PIN_AE13	LED Red[15]
LEDR[16]	PIN_AE12	LED Red[16]
LEDR[17]	PIN_AD12	LED Red[17]
LEDG[0]	PIN_AE22	LED Green[0]
LEDG[1]	PIN_AF22	LED Green[1]
LEDG[2]	PIN_W19	LED Green[2]
LEDG[3]	PIN_V18	LED Green[3]
LEDG[4]	PIN_U18	LED Green[4]
LEDG[5]	PIN_U17	LED Green[5]
LEDG[6]	PIN_AA20	LED Green[6]
LEDG[7]	PIN_Y18	LED Green[7]
LEDG[8]	PIN_Y12	LED Green[8]

Πίνακας 3. LED

Signal Name	FPGA Pin No.	Description
HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]
HEX1[0]	PIN_V20	Seven Segment Digit 1[0]
HEX1[1]	PIN_V21	Seven Segment Digit 1[1]
HEX1[2]	PIN_W21	Seven Segment Digit 1[2]
HEX1[3]	PIN_Y22	Seven Segment Digit 1[3]
HEX1[4]	PIN_AA24	Seven Segment Digit 1[4]
HEX1[5]	PIN_AA23	Seven Segment Digit 1[5]
HEX1[6]	PIN_AB24	Seven Segment Digit 1[6]
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]
HEX2[1]	PIN_V22	Seven Segment Digit 2[1]
HEX2[2]	PIN_AC25	Seven Segment Digit 2[2]
HEX2[3]	PIN_AC26	Seven Segment Digit 2[3]
HEX2[4]	PIN_AB26	Seven Segment Digit 2[4]
HEX2[5]	PIN_AB25	Seven Segment Digit 2[5]
HEX2[6]	PIN_Y24	Seven Segment Digit 2[6]
HEX3[0]	PIN_Y23	Seven Segment Digit 3[0]
HEX3[1]	PIN_AA25	Seven Segment Digit 3[1]
HEX3[2]	PIN_AA26	Seven Segment Digit 3[2]
HEX3[3]	PIN_Y26	Seven Segment Digit 3[3]
HEX3[4]	PIN_Y25	Seven Segment Digit 3[4]
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]

Πίνακας 4. 7-segment displays HEX0 – HEX3

Signal Name	FPGA Pin No.	Description
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]
HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]
HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]

Πίνακας 5. 7-segment displays HEX4 – HEX7

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	27 MHz clock input
CLOCK_50	PIN_N2	50 MHz clock input
EXT_CLOCK	PIN_P26	External (SMA) clock input

Πίνακας 6. Clock

Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

Πίνακας 7. LCD Module