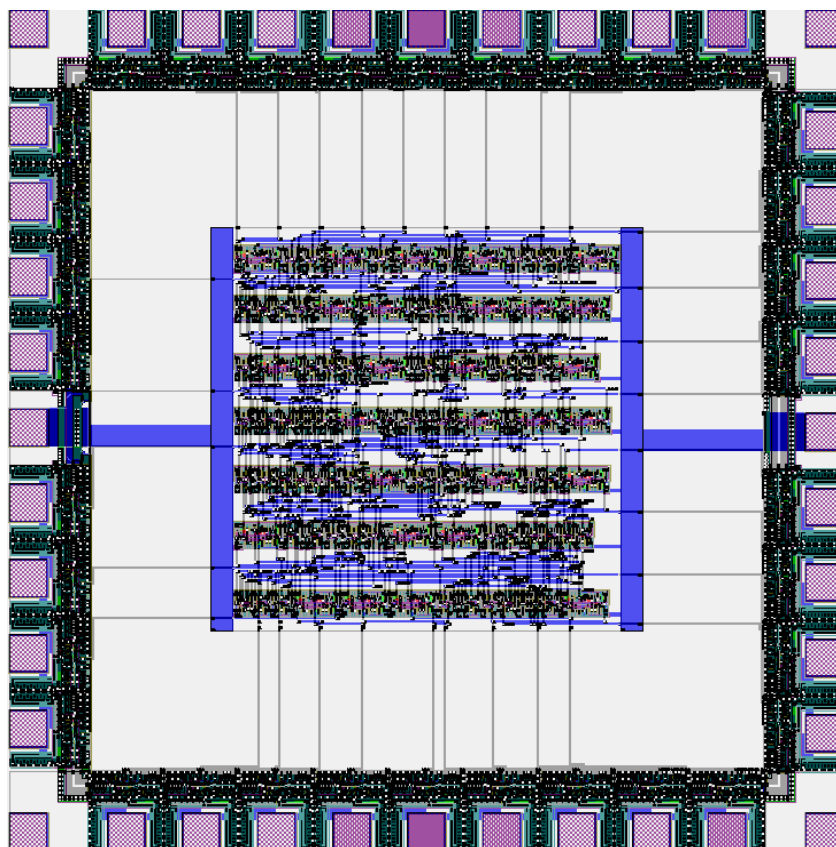




*Εργαστηριακές Ασκήσεις Σχεδίασης Ψηφιακών
Συστημάτων με χρήση Η/Υ*



Computer Aided Design

Χρ. Καβουσιανός
Καθηγητής

Φεβρουάριος 2021

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 (Classic, MAX5000-7000-9000, FLEX6000-8000-10K, Cyclone κλπ). 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 καθώς αυτή βρίσκεται στο board DE2).
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 DE2).

Λογισμικό

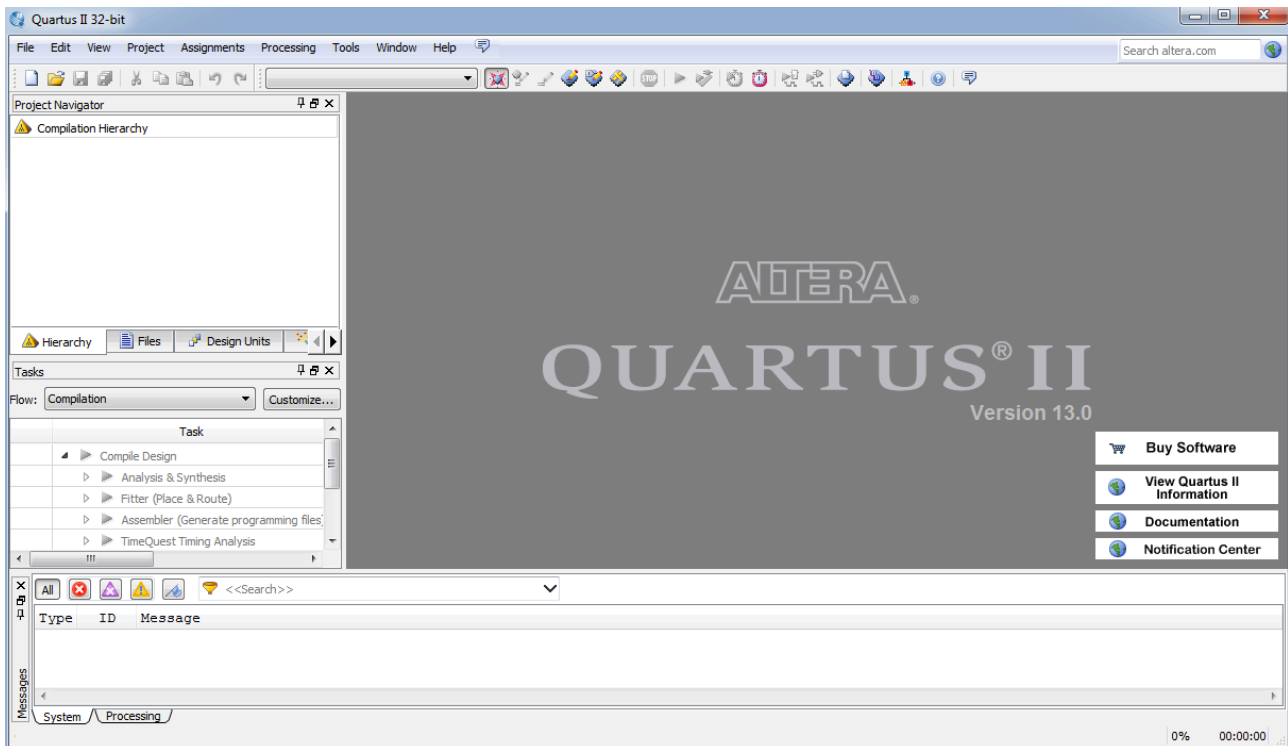
Μπορείτε να κατεβάσετε το λογισμικό (με περιορισμένες αλλά επαρκείς δυνατότητες) από την Intel (<https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html>) αφού πρώτα δημιουργήσετε έναν λογαριασμό student με τα πραγματικά στοιχεία σας και το email που διαθέτετε στο τμήμα Μηχανικών Η/Υ & Πληροφορικής. Η έκδοση του προγράμματος που θα χρησιμοποιήσουμε είναι η **Altera Quartus II, vs13.0, sp1 Web Edition** καθώς είναι η πιο πρόσφατη που υποστηρίζει το Cyclone II EP2C35F672C6 και είναι απολύτως συμβατή με τις εικόνες και τις οδηγίες που παρέχονται στην εργαστηριακή ενότητα.

Υλοποίηση Ασκήσεων

Τις παρακάτω ασκήσεις θα τις υλοποιήσετε στο εργαστήριο. Ωστόσο, για να μπορείτε να τις ολοκληρώνετε στον χρόνο του εργαστηρίου απαιτείται να τις έχετε προετοιμάσει στο σπίτι. Προτείνεται να δημιουργείτε όλα τα designs στο σπίτι και να εκτελείτε όλες τις εξομοιώσεις, και στο εργαστήριο να ελέγχετε την ορθότητα τους με την βοήθεια των επιτηρητών, στους οποίους θα πρέπει να δείξετε τους σχεδιασμούς καθώς και τα αποτελέσματά τους. Προσοχή: τα τμήματα των ασκήσεων που απαιτούν προγραμματισμό του board μπορείτε να εκτελέσετε μόνο στο εργαστήριο και εφόσον έχετε ολοκληρώσει και επιβεβαιώσει την χρονική εξομοίωση των αντίστοιχων κυκλωμάτων.

Εργαστηριακή Άσκηση 1 (Επαναληπτική): Το περιβάλλον εργασίας και απλά ψηφιακά κυκλώματα

Στα πλαίσια της εισαγωγής θα κάνουμε συνοπτική επανάληψη του εργαλείου το οποίο έχουμε ήδη συναντήσει στην Ψηφιακή Σχεδίαση Ι. Προσοχή όμως, για τις ανάγκες της Ψηφιακής Σχεδίασης ΙΙ θα χρειαστούν πρόσθετες πληροφορίες τις οποίες θα βρείτε παρακάτω και θα πρέπει να τις τηρήσετε με προσοχή. Για περισσότερες λεπτομέρειες μπορείτε να ανατρέξετε στο εγχειρίδιο χρήσης του.



Εικόνα 1. Περιβάλλον εργασίας

Το βασικό περιβάλλον εργασίας του εργαλείου παρουσιάζεται στην Εικόνα 1. Και σε αυτήν την περίπτωση ακολουθούνται τα βασικά στοιχεία 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/Files/Design Units.** Παρέχει πρόσβαση σε όλα τα τμήματα της ιεραρχίας του σχεδιασμού, τα αρχεία και τις μονάδες σχεδίασης.
- **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.
- **Pin Planner.** Παρουσιάζει την 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 τα μενού *Tools - Processing*. Από το μενού *File* μπορούμε να ανοίξουμε ή να δημιουργήσουμε αρχεία σχεδιασμού και project. Με το μενού *Assignments* μπορούμε να καθορίσουμε διάφορες παραμέτρους του σχεδιασμού.

Γενικές Οδηγίες Δημιουργίας Αρχείων

Για την ορθή οργάνωση των ασκήσεων σας θα πρέπει όλες οι ασκήσεις σας να βρίσκονται κάτω από έναν κατάλογο με το όνομα DigitalDesignII(AM1-AM2-AM3) όπου τα AM1-AM2-AM3 είναι τα AM των φοιτητών της κάθε ομάδας. Κάτω από αυτόν τον κατάλογο η κάθε εργαστηριακή άσκηση πρέπει να έχει ξεχωριστό κατάλογο Exercise1, Exercise2, ..., Exercise8 και κάθε διαφορετικό project σε κάθε μία από αυτές τις ασκήσεις πρέπει να έχει τον δικό του υποκατάλογο με ενδεικτικό όνομα που θα παραπέμπει εύκολα στο ερώτημα που απαντάτε. Μετά το πέρας της κάθε άσκησης οι κατάλογοι θα πρέπει να είναι ενημερωμένοι ώστε οι επιτηρητές να μπορούν εύκολα να ελέγξουν τις ασκήσεις χωρίς την παρουσία των φοιτητών.

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 (θα δούμε λίγο παρακάτω την διαδικασία).

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 αντίστροφη 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.

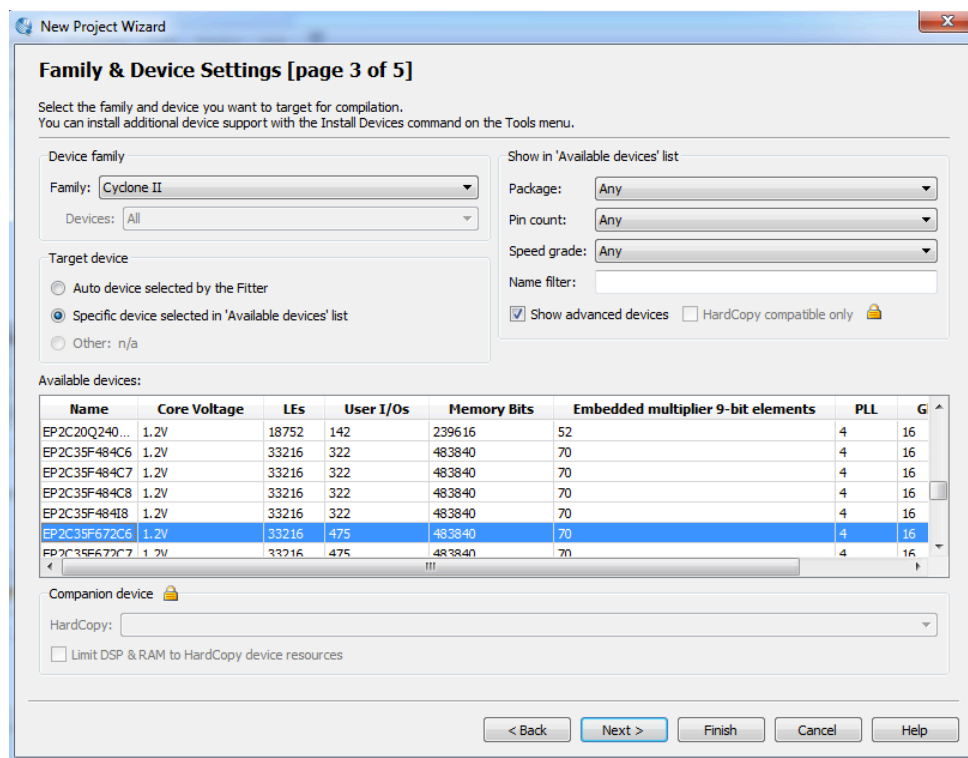
Το Ζητούμενο Κύκλωμα

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 $LED = PB1' + PB2'$.

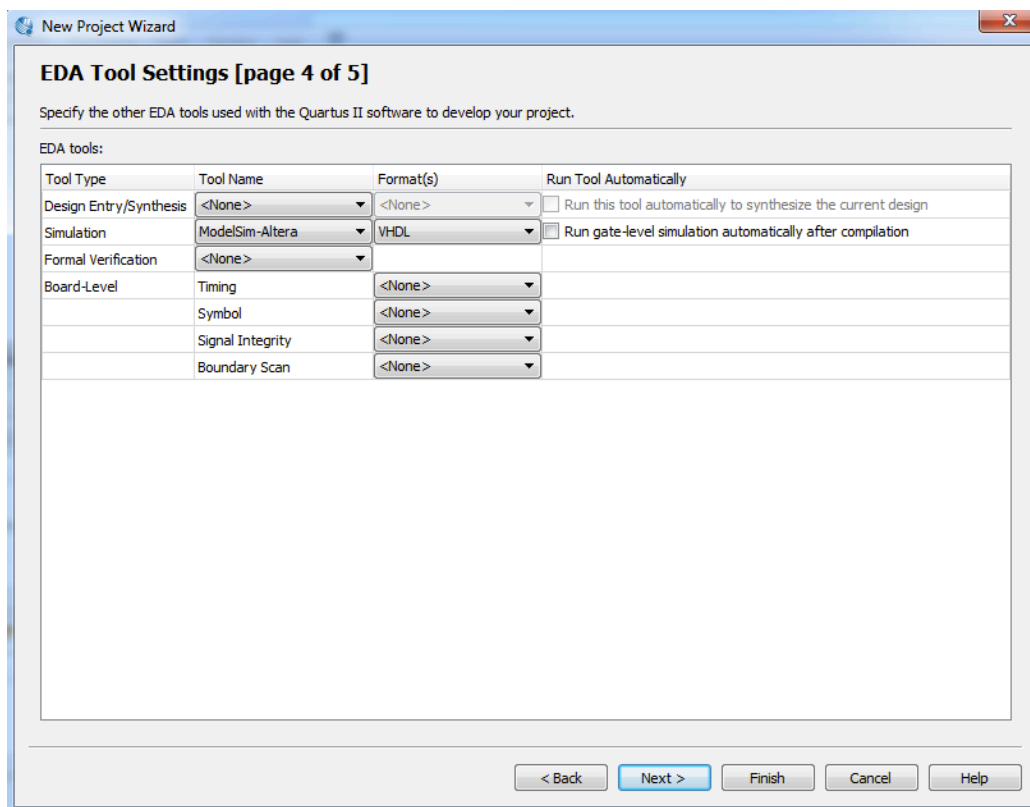
Δημιουργία Project

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. Επιπλέον **ΠΡΟΣΟΧΗ** χρειάζεται όταν προσπαθούμε να ανοίξουμε ένα παλιό project καθώς θα πρέπει να ανοίξουμε απευθείας το κεντρικό αρχείο του project με επέκταση **.qpf** (quartus project file) και όχι κάποιο αρχείο σχεδίασης ή εξομοίωσης που εντάσσεται στο project. Αν θέλουμε να ανοίξουμε ένα αρχείο σχεδίασης, τότε αυτό θα το πράξουμε **μέσω του project** και όχι απευθείας από τον δίσκο του υπολογιστή μας.



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 ως όνομα το *MyFirstProject*. Παρατηρήστε ότι αυτόματα συμπληρώνεται με το ίδιο όνομα και η υψηλότερη οντότητα του σχεδιασμού μας (αυτό είναι απαίτηση του Quartus, αν και παρακάτω θα μάθουμε πως μπορούμε να το αναιρέσουμε). Πιέζουμε *Next* και πάλι *Next* ώστε να παρακάμψουμε το επόμενο παράθυρο το οποίο προσθέτει αρχεία στο Project (ακόμη δεν έχουμε δημιουργήσει τέτοια αρχεία). Έτσι εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 2. Εδώ πρέπει να ορίσουμε ακριβώς την προγραμματιζόμενη συσκευή που θα χρησιμοποιήσουμε (Cyclone II, EP2C35F672C6). Κάνουμε ακριβώς τις ρυθμίσεις που φαίνονται στην Εικόνα 2. Εάν επιλέξουμε άλλη συσκευή τότε δεν θα μπορέσουμε να εκτελέσουμε την άσκηση στο board.



Εικόνα 3

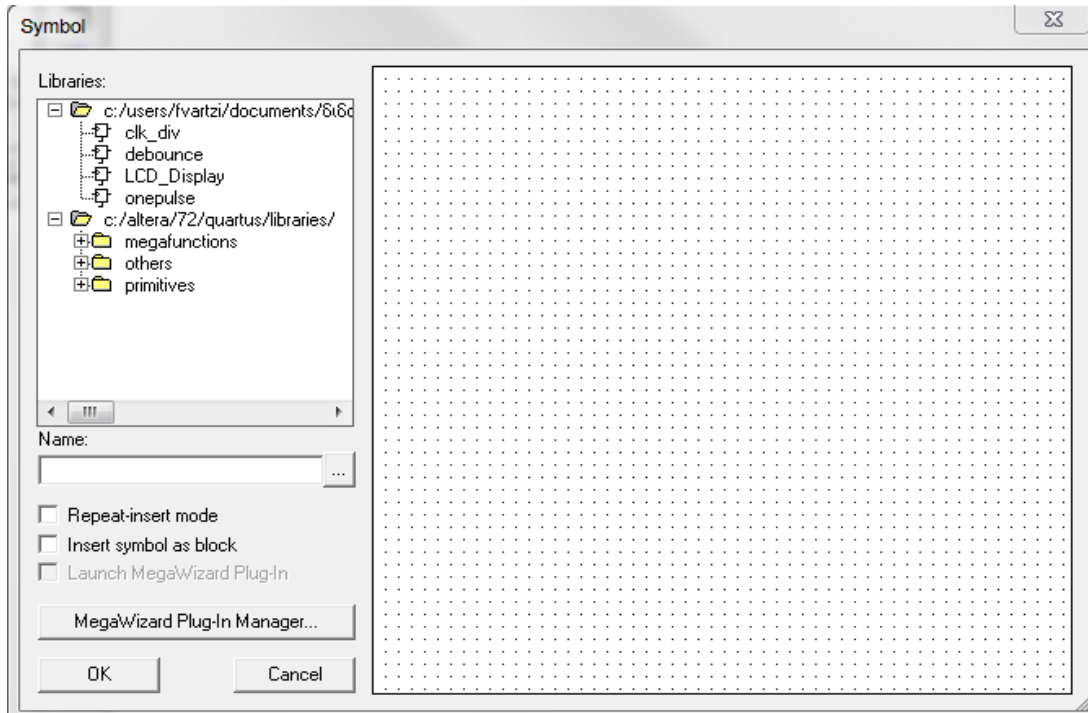
Πιέζουμε *Next* ώστε να περάσουμε στις ρυθμίσεις των εργαλείων EDA. Σε αυτό το πεδίο πρέπει να ορίσουμε ποια εργαλεία θα χρησιμοποιήσουμε σε συνδυασμό με το Quartus. Στην περίπτωσή μας πρέπει να ορίσουμε το πρόγραμμα εξομίωσης το οποίο είναι το *ModelSim-Altera* όπως φαίνεται στην Εικόνα 3. Κατόπιν πατάμε *Next* για άλλη μία φορά οπότε εμφανίζεται η περίληψη των βασικών ρυθμίσεων του Project. Πατάμε *Finish* και έχουμε πλέον δημιουργήσει το πρώτο μας project.

Η προγραμματιζόμενη συσκευή της Cyclone παρέχει έναν αριθμό από ακροδέκτες προκειμένου να μπορέσουμε να διασυνδέσουμε περιφερειακές συσκευές πάνω σε αυτήν. Στα πλαίσια των εργαστηριακών ασκήσεων μόνο ένα μικρό μέρος από αυτούς τους ακροδέκτες θα χρησιμοποιήσουμε. Όλοι οι ακροδέκτες έχουν διασυνδεθεί με 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 χρησιμοποιώντας έναν ασύνδετο ακροδέκτη με λάθος τρόπο. 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 όπου έχουμε τοποθετήσει την βιβλιοθήκη CoreLibrary η οποία βρίσκεται αναρτημένη στο ecourse, και επιλέγουμε "*Add*". 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 4

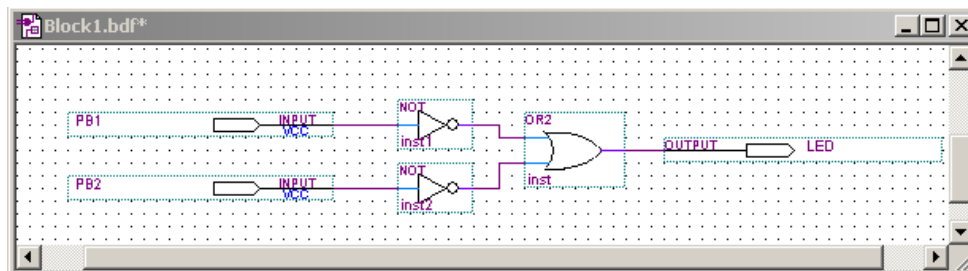
Για να τοποθετήσουμε κάποιο σύμβολο του κυκλώματος πρέπει αρχικά να τοποθετήσουμε τον ειδικό κέρσορα σε οποιοδήποτε σημείο του σχηματικού επιθυμούμε (απλό αριστερό κλικ στο ποντίκι) και κατόπιν να εκτελέσουμε την εντολή *Insert>Symbol* (με διπλό κλικ αριστερό ή κλικ δεξιό και επιλογή *Insert>Symbol*). Τότε ανοίγει το παράθυρο που φαίνεται στην Εικόνα 4. 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/13.0sp1/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 KAI, Η κλπ)
 - *Others*.
2. *altera/CoreLibrary*. Επιλεγμένοι 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.

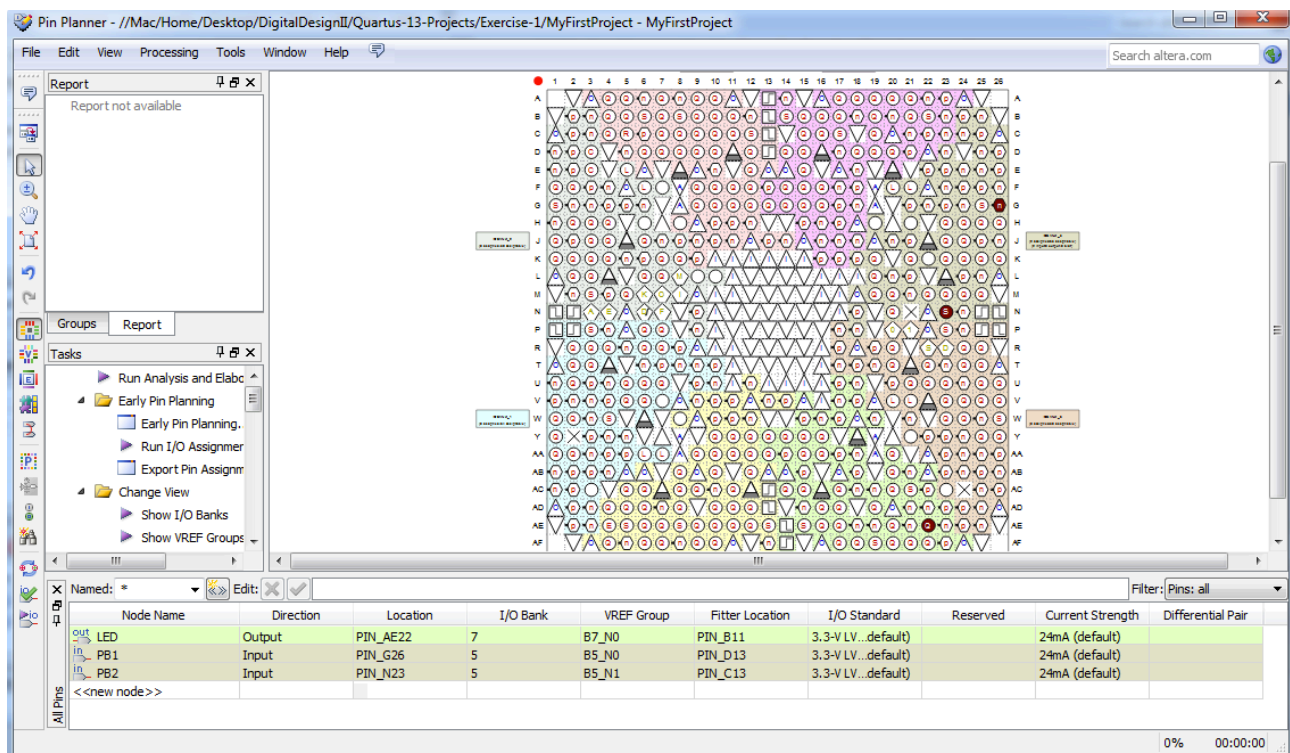
Το επόμενο βήμα είναι να συνδέσουμε τις πύλες μεταξύ τους. Συγκεκριμένα οι έξοδοι των δύο αντιστροφέων θα συνδεθούν στις εισόδους της πύλης Or. Εάν τοποθετήσουμε τον δείκτη του ποντικιού πάνω στο άκρο της εξόδου του αντιστροφέα παρατηρούμε ότι ο δείκτης αλλάζει σε σταυρό. Πατώντας το αριστερό πλήκτρο 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 5.

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 5.

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 6

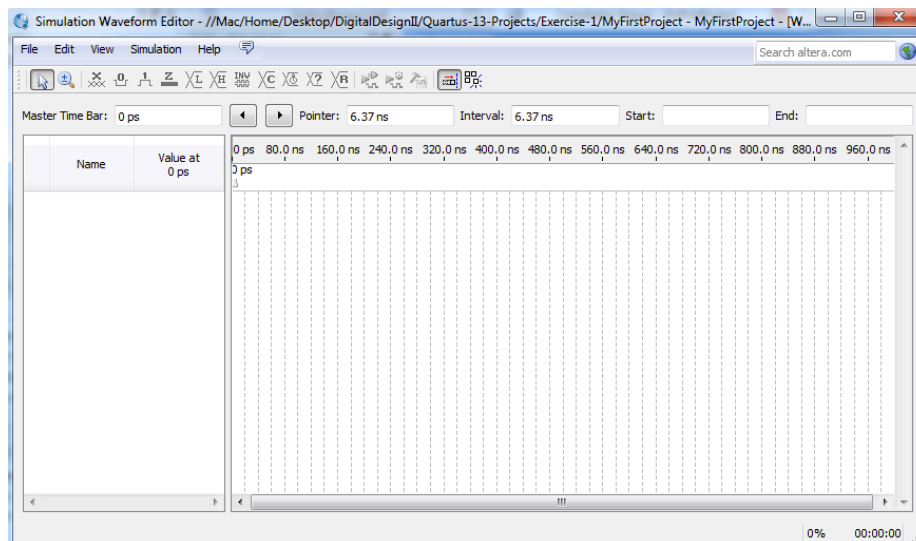
Για να αντιστοιχήσουμε τις θύρες εισόδου εξόδου σε συγκεκριμένους ακροδέκτες εκτελούμε την ακόλουθη διαδικασία: αρχικά εκτελούμε την διαδικασία μεταγλώττισης (compile) ώστε να μεταγλωττιστεί το σχηματικό διάγραμμα που δημιουργήσαμε. Εκτελούμε Processing > Start Compilation. Ακόλουθα εκτελούμε την εντολή Assignments>Pin Planner οπότε εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 6. 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.

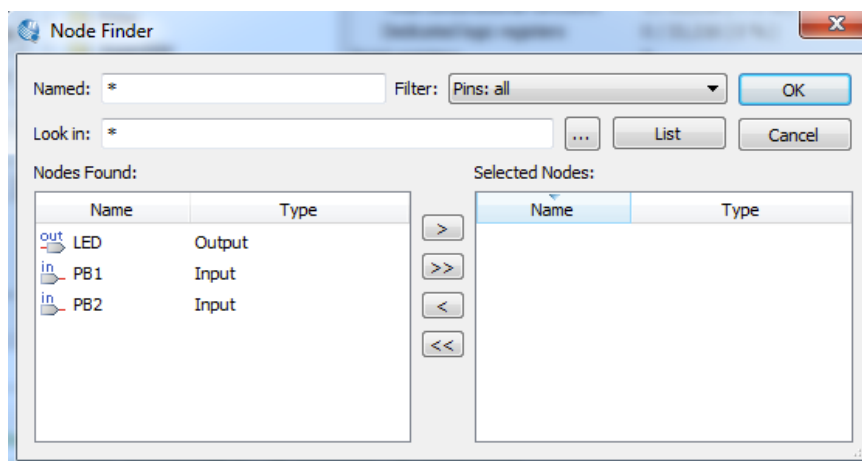
Επιβεβαίωση της Σχεδίασης

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 (XX warnings)" οπότε διαπιστώνουμε ότι δεν έχουμε κάνει κάποιο λάθος, αν και είναι πολύ πιθανόν να έχουμε κάποια warnings τα οποία πρέπει να αξιολογήσουμε (τα περισσότερα από αυτά δεν μας ανησυχούν). 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 7. Δημιουργία αρχείου κυματομορφών



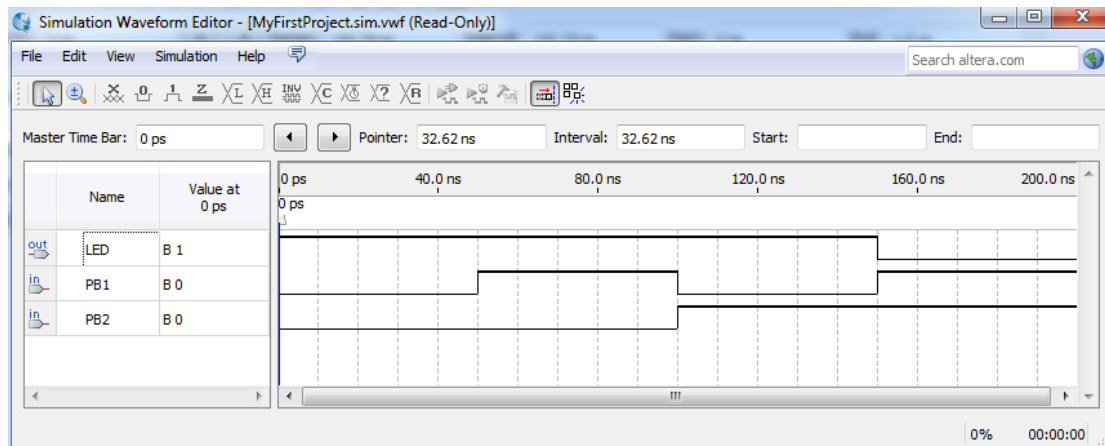
Εικόνα 8. Επιλογή σημάτων

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 *Verification/Debugging Files* και συγκεκριμένα την επιλογή *University Program VWF*. Έτσι εμφανίζεται το παράθυρο καθορισμού των κυματομορφών που είναι αρχικά κενό (Εικόνα 7). 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 8. Επιλέγοντας στο πεδίο *Filter* την επιλογή *Pins: all* και κατόπιν πιέζοντας το *List* εμφανίζονται όλες οι θύρες. Με το πλήκτρο *>>* επιλέγονται όλες οι θύρες μαζί και τοποθετούνται στην περιοχή *Selected Nodes* (Εικόνα 8). 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. Προσοχή όμως χρειάζεται στον εντοπισμό των αποτελεσμάτων της εξομοίωσης, καθώς το παράθυρο που δουλεύουμε τώρα αφορά τις εισόδους και όχι τα αποτελέσματα.

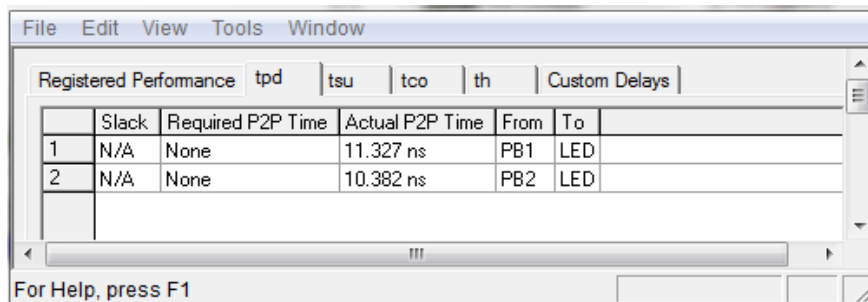
Αφού επιλέξουμε το σήμα PB1 εκτελούμε δεξιό κλικ πάνω στο PB1 στην περιοχή της κυματομορφής με το πλέγμα και επιλέγουμε *Value>Count Value*. Με την επιλογή αυτή θα δώσουμε αυτόματα τιμές μέτρησης (Count) στην είσοδο PB1 για όσο διάστημα έχουμε ορίσει ότι θα εκτελεστεί η εξομοίωση. Επιλέγουμε την καρτέλα *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 9.

Αποθηκεύουμε το αρχείο εισόδου της εξομοίωσης (*File>Save*) δίνοντας το ίδιο όνομα με τον σχεδιασμό που θα εξομοιώσουμε (στην περίπτωση μας είναι το όνομα του project) και εκτελούμε την εξομοίωση με την εντολή *Simulation>Run Functional Simulation*. Όταν ολοκληρωθεί η εξομοίωση εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 9. Μπορούμε να κάνουμε View Zoom In/View Zoom Out για να εστιάσουμε στο σημείο που επιθυμούμε. Παρατηρούμε ότι κάθε αλλαγή στην έξοδο συμβαίνει ταυτόχρονα με τις αλλαγές στις εισόδους χωρίς καμία καθυστέρηση. Φυσικά αυτό οφείλεται στο είδος της εξομοίωσης που εκτελέσαμε η οποία είναι λειτουργική και όχι χρονική. Αυτή η εξομοίωση είναι απαραίτητη όταν θέλουμε σε πρωταρχικά στάδια να ελέγξουμε την ορθότητα του σχεδιασμού μας γρήγορα, για να εντοπίσουμε τα πιο σημαντικά λάθη της σχεδίασης μας.

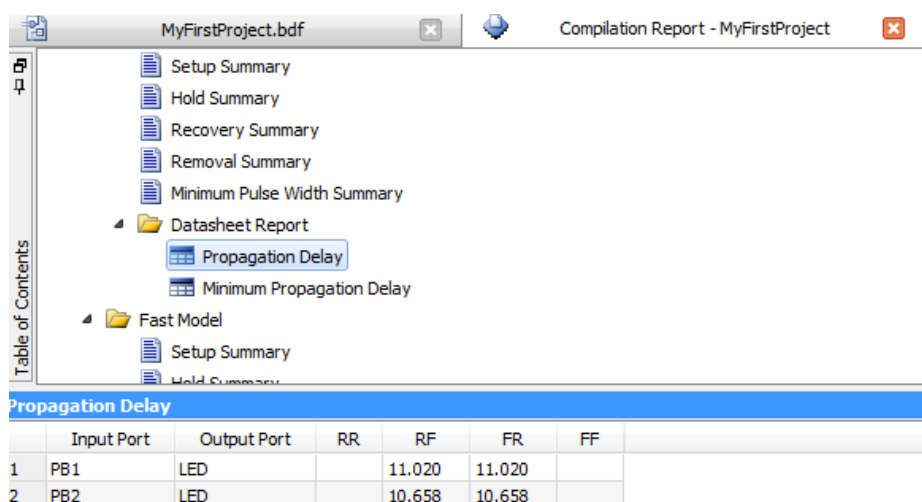
Επαναλαμβάνουμε την εξομοίωση επιλέγοντας *Simulation>Run Timing Simulation* αυτή την φορά αλλάζοντας τον χρόνο εξομοίωσης σε 300ns. Παρατηρούμε ότι από την αλλαγή της εισόδου PB1 0→1 περνάει χρόνος περίπου 10.63 ns (τον δείκτη χρόνου μπορούμε να τον εισάγουμε με διπλό κλικ στην περιοχή της κυματομορφής). 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.



	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 10

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 ns όπως προέκυψε από την χρονική εξομοίωση). 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 θα πρέπει να εκτελέσουμε στατική ανάλυση. Η στατική ανάλυση εκτελείται μέσω του TimeQuest Timing Analyzer, και σε επόμενη άσκηση θα μιλήσουμε αναλυτικά για την χρήση του κυρίως στην εισαγωγή χρονικών περιορισμών στην λειτουργία του κυκλώματος.



	Input Port	Output Port	RR	RF	FR	FF
1	PB1	LED		11.020	11.020	
2	PB2	LED		10.658	10.658	

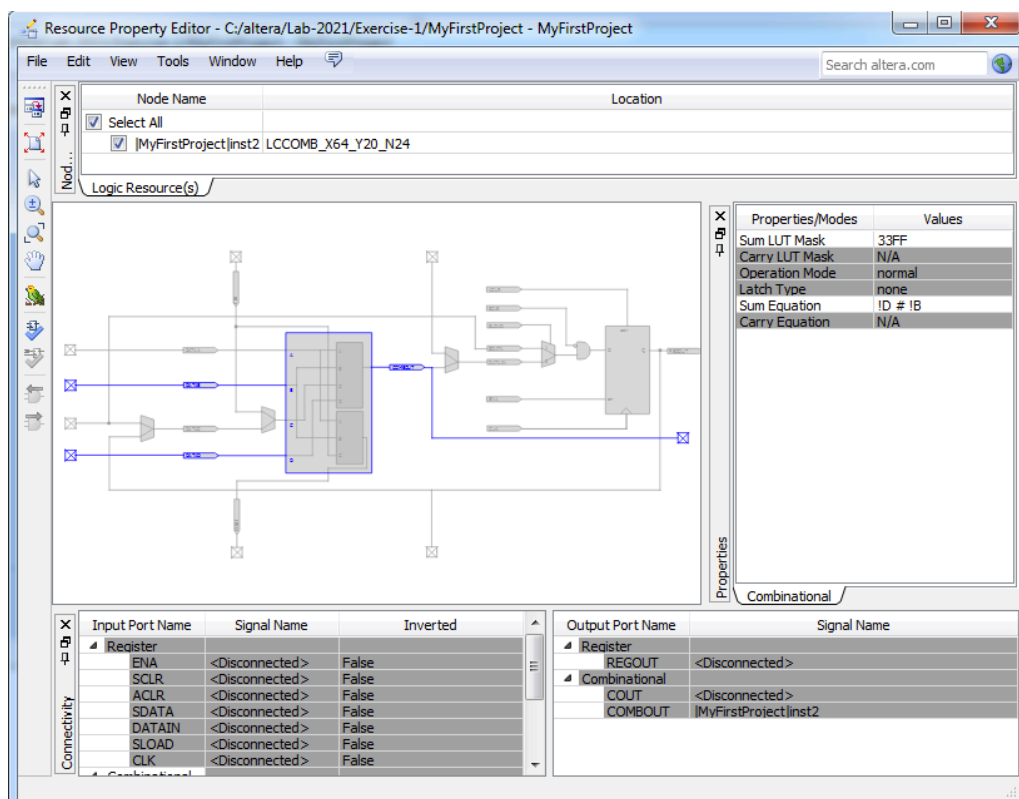
Εικόνα 11. Χρονικές Καθυστερήσεις Εισόδων/Εξόδων

Η στατική ανάλυση εκτελείται κατά την διαδικασία της σύνθεσης (compilation), και τα αποτελέσματα της μπορούμε να τα αναζητήσουμε στο tab Compilation Report. Καθώς το κύκλωμα μας είναι τετριμμένο και δεν έχει ακολουθιακά στοιχεία (συνεπώς δεν έχει σήμα ρολογιού) μπορούμε

να δούμε μόνο καθυστερήσεις από εισόδους σε εξόδους. Ανοίγουμε λοιπόν το Tab Compilation Report και κάτω από τον *TimeQuestAnalyzer* επιλέγουμε *Slow Model/Datasheet Report/Propagation Delay*. Αν η σύνθεση έχει γίνει σωστά τότε πρέπει να βλέπουμε τα ίδια αποτελέσματα με την Εικόνα 11. Αναλυτικότερα εμφανίζονται οι χρόνοι RR, RF, FR, FF για τα μονοπάτια PB1 → LED και PB2 → LED (προσέξτε ότι είναι τα μοναδικά μονοπάτια στο κύκλωμα μας, αλλά σε πιο συνηθισμένα κυκλώματα υπάρχουν χιλιάδες διαφορετικά μονοπάτια). Η τιμή RF = 11.02 αντιστοιχεί στον χρόνο που απαιτείται για να πάει στο 0 (F) η έξοδος LED όταν η είσοδος PB1 πάει στην μονάδα, υπό την προϋπόθεση ότι η δεύτερη είσοδος της πύλης OR επιτρέπει την διάδοση του σήματος (είναι ίση με 0 άρα PB2=1). Ίδια καθυστέρηση αντιστοιχεί και στην μετάβαση FR (PB1 → 0, LED → 1). Προσέξτε ότι δεν υπάρχουν οι καθυστερήσεις RR και FF καθώς δεν υπάρχει τρόπος μία τιμή στην είσοδο PB1 = 0,1 όταν PB2 = 1 να δώσει τιμή στην έξοδο αντίστοιχα LED = 0,1 (το ίδιο ισχύει και για την PB2 όταν PB1 = 1). Παρατηρήστε ότι οι καθυστερήσεις από PB1 → LED και από PB2 → LED είναι διαφορετικές. Αντίστοιχες τιμές μπορούμε να δούμε και στο *Fast Model/Datasheet Report/Propagation Delay*, μόνο που είναι πολύ μικρότερες από τις προηγούμενες. Αυτό οφείλεται στα διαφορετικά μοντέλα καθυστερήσεων που χρησιμοποιούνται (τα δύο ακραία μοντέλα Slow και Fast εξαρτώνται από τις παραμέτρους PVC - Process variation, Voltage, Temperature). Εμείς θα χρησιμοποιούμε το Slow Model για τις ασκήσεις μας.

Floorplan Editor

Έχοντας πλέον ελέγξει και την ορθότητα του σχεδιασμού μας μπορούμε να δούμε οπτικά πως τοποθετείται 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. Εμείς στο εργαστήριο θα δούμε αυτή την δυνατότητα του Quartus αλλά μόνο επιγραμματικά.



Εικόνα 12. Resource Project Editor

Εκτελούμε *Assignments>Back Annotate Assignments*, πιέζουμε OK και κατόπιν επιλέγουμε την πύλη OR στον σχεδιασμό μας. Με δεξί κλικ του ποντικιού επιλέγουμε *Locate in Chip Planner*. Στο παράθυρο που ανοίγει αναζητούμε το μπλε Block και αφού τοποθετήσουμε τον κέρσορα πάνω του κάνουμε διπλό κλικ οπότε ανοίγει το παράθυρο *Resource Project Editor* όπως φαίνεται στην Εικόνα 12. Εδώ μπορεί κάποιος να δει τις ακριβείς συνδέσεις που υλοποιούν την πύλη OR μέσα στο επιλεγμένο LUT (Look-Up Table – θα μιλήσουμε για αυτό στην θεωρία). Με παρόμοιο τρόπο μπορεί να εντοπίσει τις εισόδους/εξόδους του κυκλώματος και τα αντίστοιχα pins του FPGA που χρησιμοποιήθηκαν.

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 και πιέζουμε 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 (αν όχι το εντοπίζετε και το προσθέτετε με την επιλογή *Add File*). 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. Προσέξτε ότι η πρώτη εργαστηριακή άσκηση είναι σχετικά εύκολη, αλλά οι επόμενες ασκήσεις είναι πιο απαιτητικές και χρειάζονται προετοιμασία από το σπίτι.

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 και βρείτε την μέγιστη καθυστέρηση του αποκωδικοποιητή. Συγκρίνετε την με αυτή που θα σας δώσει η στατική ανάλυση (δείτε το 1^ο Μέρος της άσκησης). 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.

A_1	A_0	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.

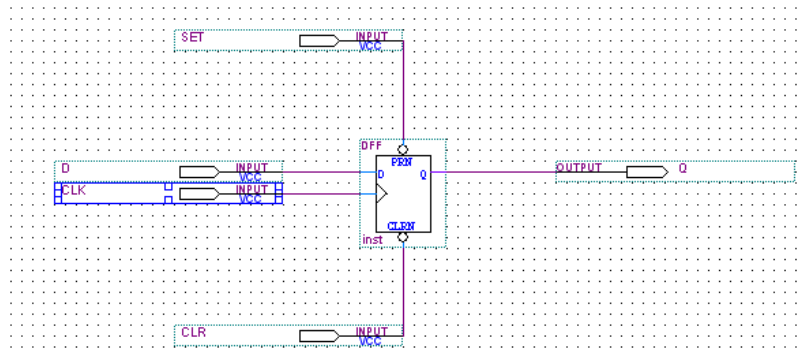
I_2	I_1	I_0	F_1	F_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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).

Βασικά Ακολουθιακά Κυκλώματα

Ερώτημα 1ο

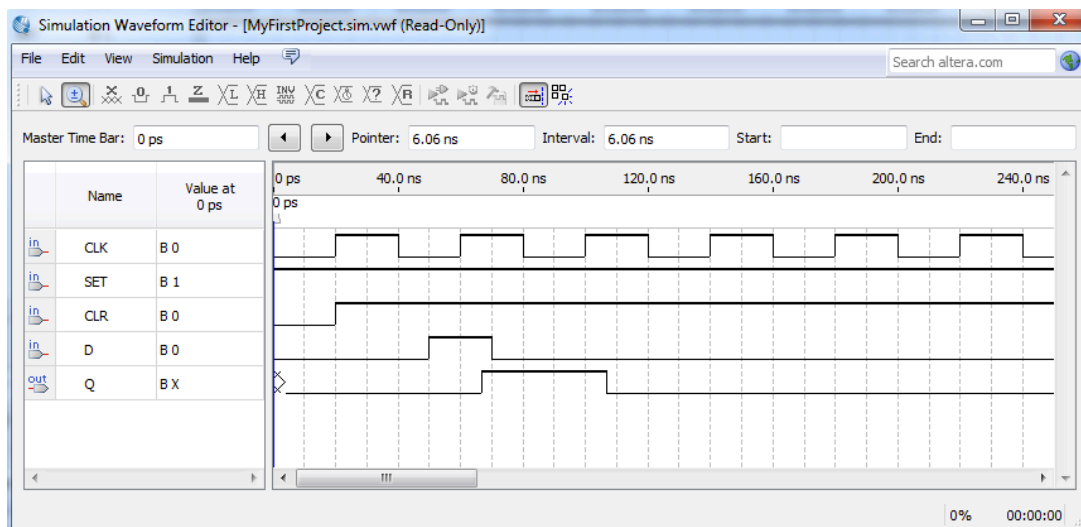
Στο πρώτο τμήμα της άσκησης θα σχεδιάσουμε τα βασικά flip flop. Με την γνωστή διαδικασία σχεδίασης δημιουργήστε ένα νέο σχηματικό στο οποίο θα τοποθετήσετε ένα D flip flop. Το D-flip flop θα το βρείτε στην βιβλιοθήκη libraries>Primitives>Dff και θα το τοποθετήσετε στο σχηματικό με τον γνωστό τρόπο. Συγκεκριμένα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



Εικόνα 13. Σχηματικό με D flip-flop

Παρατηρήστε στην Εικόνα 13 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 14

Παρατηρήστε τις κυματομορφές στην Εικόνα 14. 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 μηδένισης (CLR) το οποίο ενεργοποιείται από την αρχή μέχρι την χρονική στιγμή 20 ns και μετά απενεργοποιείται. Για να το επιτύχουμε αυτό ακολουθούμε τα εξής βήματα:

1. Θέτουμε το σήμα CLR στην μονάδα (με δεξί κλικ πάνω στο σήμα επιλέγουμε Value>Forcing High 1).

2. Με το ποντίκι επιλέγουμε την περιοχή του σήματος (0ns-20ns) και την θέτουμε με αντίστοιχο τρόπο στο μηδέν.

Με όμοιο τρόπο θέτουμε και τα σήματα SET, D στις τιμές που φαίνονται στο παραπάνω σχήμα. Κατόπιν εκτελούμε εξομοίωση και παίρνουμε τα αποτελέσματα που φαίνονται στο παραπάνω σχήμα (για τα πρώτα 250ns). 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 περίπου 7ns τα οποία είναι ουσιαστικά η καθυστέρηση που εισάγει το flip flop. Παρατηρήστε ότι το σήμα D πάει στο λογικό 1 αρκετά πιο πριν (στα 50ns) 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.

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 , μία σειριακή είσοδο δεδομένων 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	Παράλληλη φόρτωση $I_{3...0}$
0	1	Αριστερή Ολίσθηση
1	0	Δεξιά Ολίσθηση
1	1	Συμπλήρωση $(Q_{3...0})' \rightarrow Q_{3...0}$

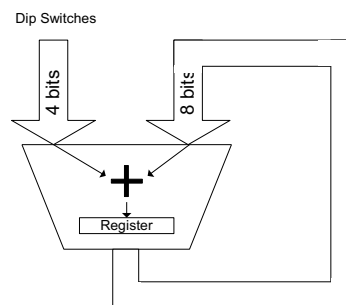
Χρησιμοποιήστε τέσσερις πολυπλέκτες 4 σε 1 σύμφωνα με τις μεθόδους σχεδίασης που ακολουθήσαμε στην Ψηφιακή Σχεδίαση. Σχεδιάστε μόνοι σας έναν πολυπλέκτη 4 σε 1 και αντιγράψτε τον με copy-paste τέσσερις φορές. Παρακάτω θα μάθουμε πιο αποδοτικούς τρόπους να εισάγουμε επαναλαμβανόμενες δομές και θα δούμε 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.

Παραδοτέα 1^{ης} Εργαστηριακής Άσκησης

1. Σχεδιάστε όλα τα ζητούμενα κυκλώματα στο Μέρος 2 και εκτελέστε τις απαραίτητες εξομοιώσεις.
2. Παρουσιάστε τα αποτελέσματα στους επιτηρητές σας.
3. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

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).



Εικόνα 15. Συσσωρευτής

Κύκλωμα Συσσωρευτή

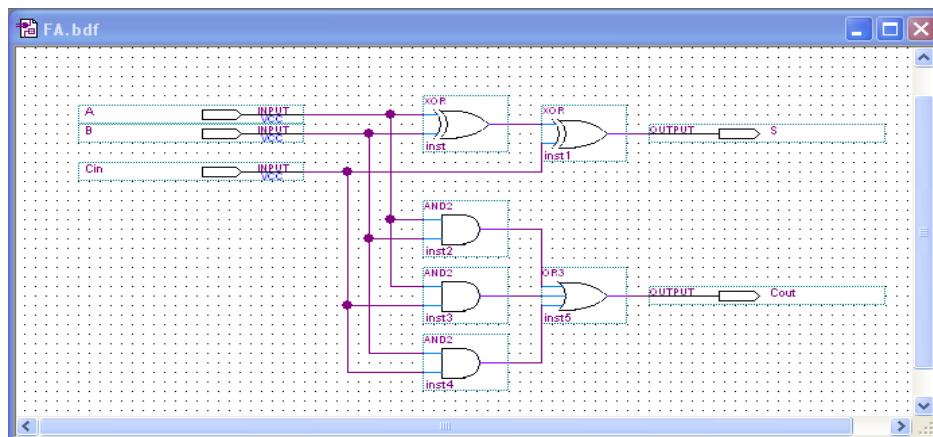
Θα σχεδιάσουμε έναν συσσωρευτή των 8 δυαδικών ψηφίων όπως φαίνεται στην Εικόνα 15. Ο ζητούμενος συσσωρευτής αποτελείται από έναν αθροιστή και έναν καταχωρητή των 8 δυαδικών ψηφίων. Παρατηρούμε ότι οι δύο εισοδοί του αθροιστή δεν έχουν το ίδιο μέγεθος. Η μία είσοδος αποτελείται από 4 δυαδικά ψηφία και 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 για τις ανάγκες 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.

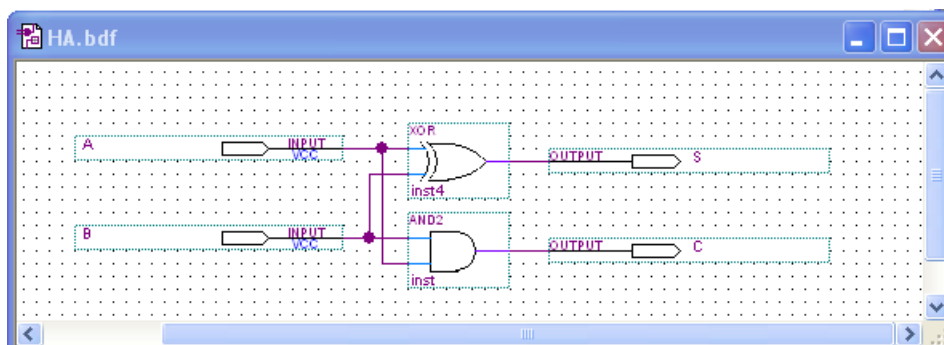
Ωστόσο για εκπαιδευτικούς λόγους θα σχεδιάσουμε μόνοι μας τις βασικές δομικές μονάδες ώστε να κατανοήσουμε την διαδικασία της ιεραρχικής σχεδίασης.



Εικόνα 16. 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 16. 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 απαιτεί από την 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. Αρχικά ελέγχουμε την μέγιστη καθυστέρηση που μας δίνει η χρονική ανάλυση (από το tab *Compilation Report* επιλέγουμε *Time Quest Timing Analyzer > Slow Model > Data Sheet Report > Propagation Delay*) και παρατηρούμε ότι είναι λίγο κάτω από 10 ns. Δίνουμε όλες τις δυνατές τιμές στον αθροιστή με ελάχιστη περίοδο σήματος εισόδου 15 ns (χρησιμοποιήστε το Count Value με περίοδο 15ns, 30ns, 60ns για τα σήματα A, B, Cin αντίστοιχα). Κατά την επιλογή των σημάτων ενδεχομένως να μην εμφανίζονται *Nodes* στην Εικόνα 8. 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 χρονική εξομοίωση με τον τρόπο που δείξαμε στην άσκηση 1. Δείτε τα αποτελέσματα της εξομοίωσης στον επιτηρητή σας.

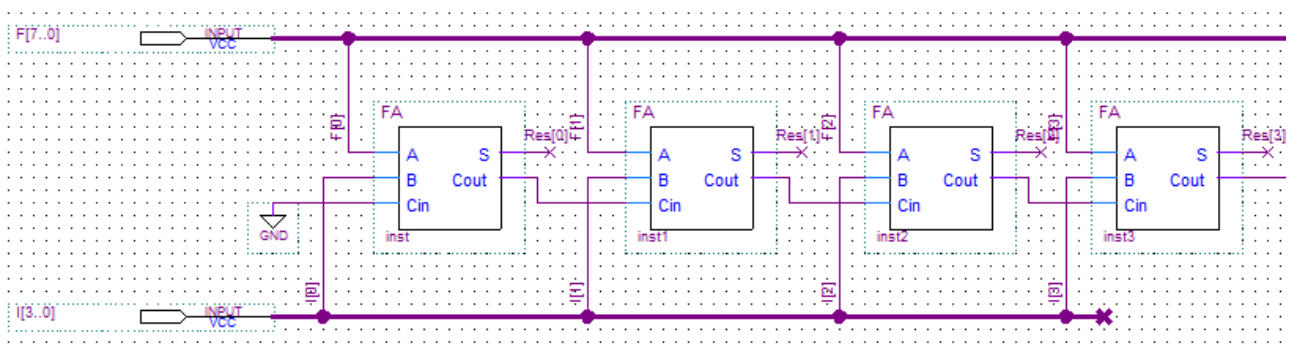


Εικόνα 17. Ημιαθροιστής

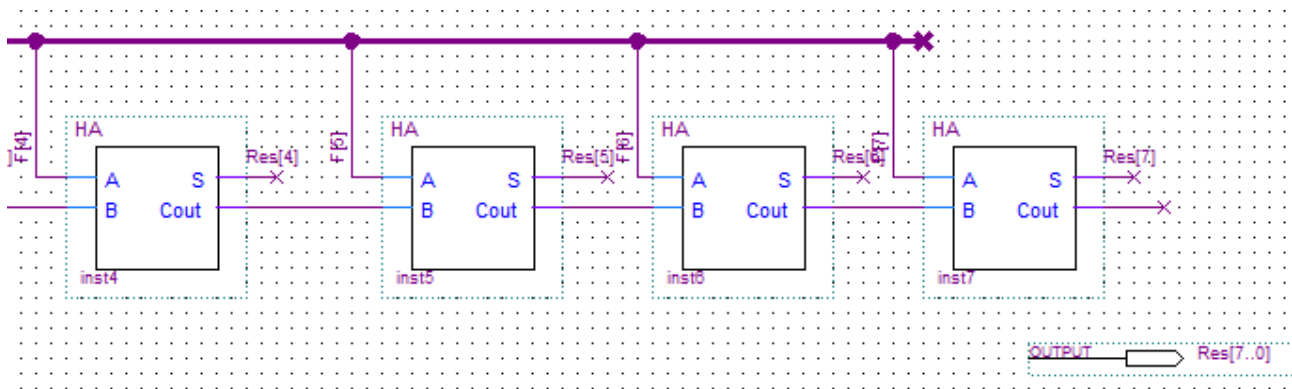
Έχοντας πλέον επιβεβαιώσει την σωστή λειτουργία του πλήρη αθροιστή μπορούμε να δημιουργήσουμε το σύμβολο του για να μπορέσουμε να τον χρησιμοποιήσουμε ιεραρχικά. Εκτελούμε την εντολή *File>Create/Update>Create Symbol Files for Current File*. Ακολουθούμε την ίδια διαδικασία για τον ημιαθροιστή που φαίνεται στην Εικόνα 17. 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 4) εμφανίζεται επιπλέον και η βιβλιοθήκη *Project* και αν την ανοίξουμε 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 *B* κάθε πλήρη αθροιστή πάνω στον δίαυλο και δίνουμε τα ονόματα *I[0]*, *I[1]*, *I[2]*, *I[3]* στην κάθε γραμμή με δεξιό 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 *A* 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 18 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 18 φαίνεται το κύκλωμα που υπολογίζει τα 4 λιγότερο σημαντικά δυαδικά ψηφία του αθροίσματος, ενώ στην Εικόνα 19 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.

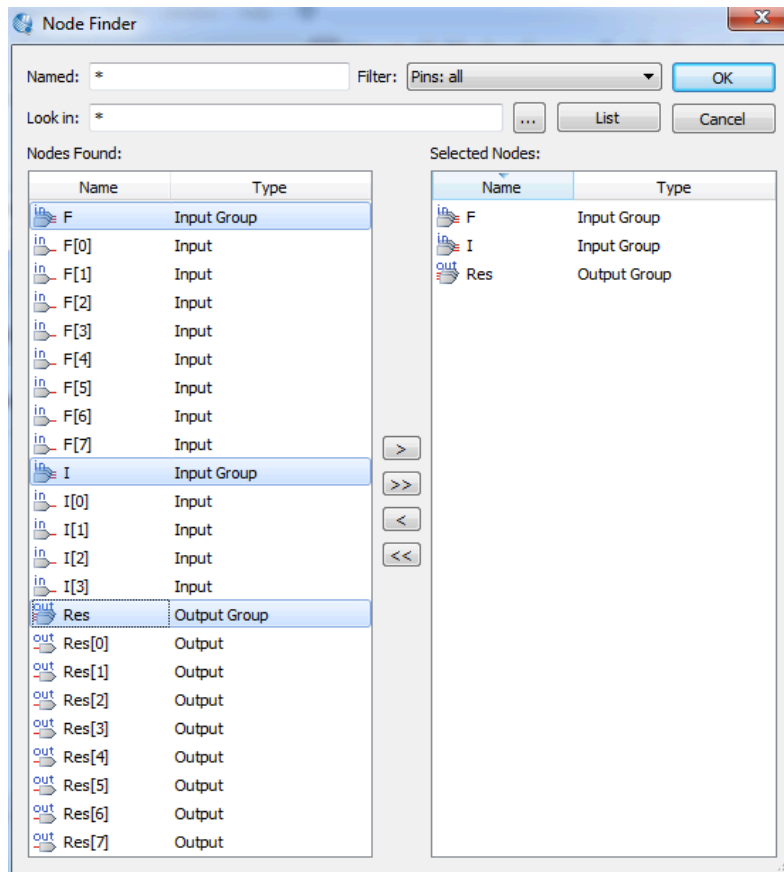


Εικόνα 18. Αθροιστής 8 bits (α)



Εικόνα 19. Αθροιστής 8 bits (β)

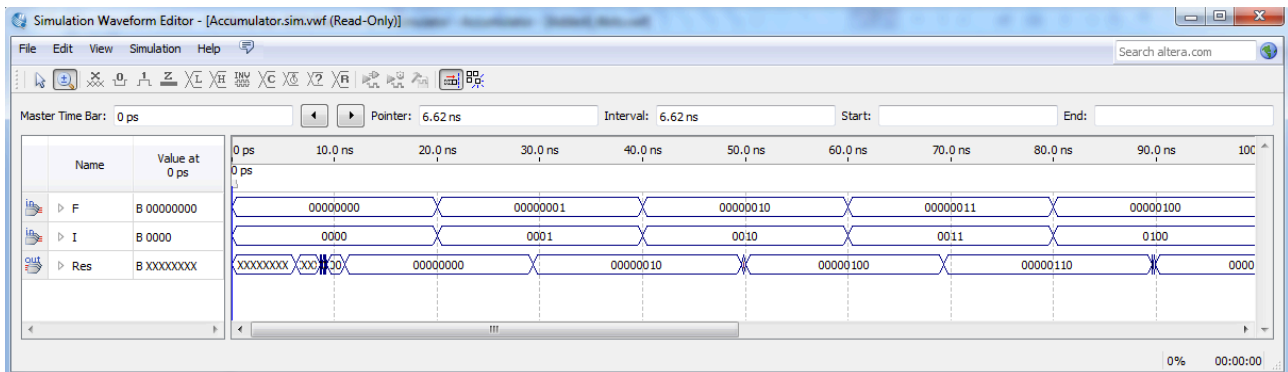
Πριν ελέγξουμε την σχεδίαση του αθροιστή, κάνουμε διπλό κλικ σε κάποιον πλήρη αθροιστή ή ημιαθροιστή. Θα παρατηρήσουμε τότε ότι ανοίγει το σχηματικό που σχεδιάσαμε προηγουμένως. Με άλλα λόγια κατεβήκαμε ένα επίπεδο στην ιεραρχία. Κλείνουμε το σχηματικό που ανοίξαμε και εκτελούμε σύνθεση.



Εικόνα 20

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 20. 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

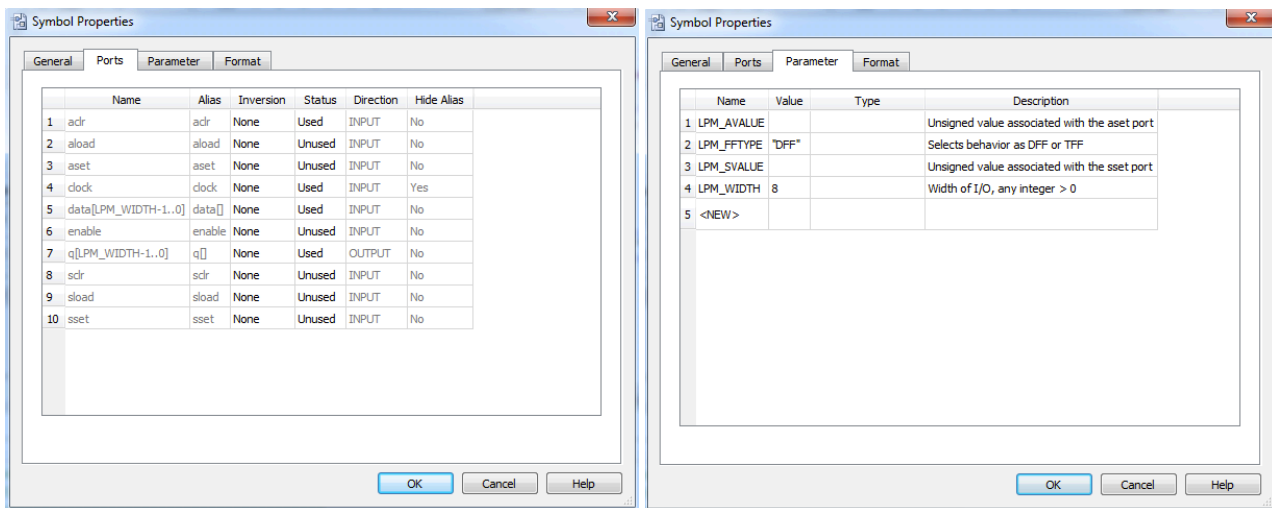
αθροιστή. Πριν ξεκινήσουμε την εξομίωση δημιουργούμε ένα νέο αρχείο κυματομορφών και το ορίζουμε ως το αρχείο κυματομορφών της εξομίωσης όπως έχουμε ήδη αναφέρει. Ένα μικρό τμήμα της εξομίωσης φαίνεται στην Εικόνα 21. Τέλος δημιουργούμε ένα σύμβολο για τον αθροιστή που σχεδιάσαμε.



Εικόνα 21. Εξομίωση 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.

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> quartus/libraries/> Megafunctions> Storage> LPM_FF* και τοποθετούμε την αντίστοιχη Megafunction. Προσέχουμε ότι οι Megafunctions είναι παραμετροποιήσιμες μονάδες οι οποίες μπορούν να δώσουν πολλές μονάδες με διαφορετικές δυνατότητες, οπότε πρέπει να καθορίσουμε τις παραμέτρους για να τις χρησιμοποιήσουμε σωστά.

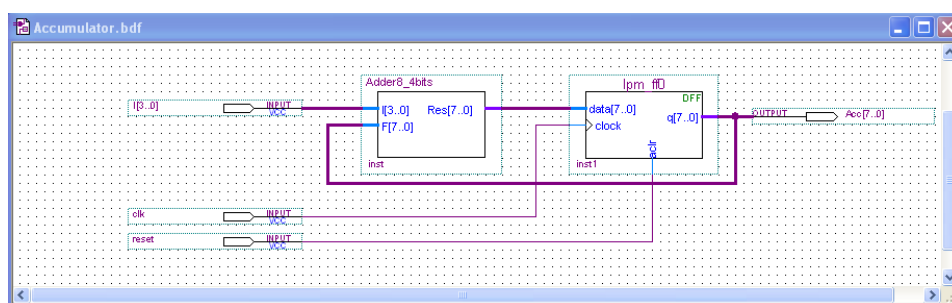


Εικόνα 22

Για να καθορίσουμε τον καταχωρητή που χρειαζόμαστε θα πρέπει να παραμετροποιήσουμε την Megafunction. Οι παράμετροι που μας ενδιαφέρουν είναι οι ακόλουθοι:

- aclr: χρησιμοποιείται για ασύγχρονο μηδενισμό
- clock: σήμα ρολογιού
- data: είσοδος δεδομένων καταχωρητή
- q: έξοδος καταχωρητή
- LPM_FFTYPE: τύπος flip flop που θα χρησιμοποιηθεί
- LPM_WIDTH: πλάτος/μέγεθος καταχωρητή.

Οι τιμές των παραμέτρων φαίνονται στην Εικόνα 22 και μπορούν να καθοριστούν επιλέγοντας την μονάδα στο σχηματικό και με δεξί κλικ *Properties*. Αφού ολοκληρώσουμε την διαμόρφωση του 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.



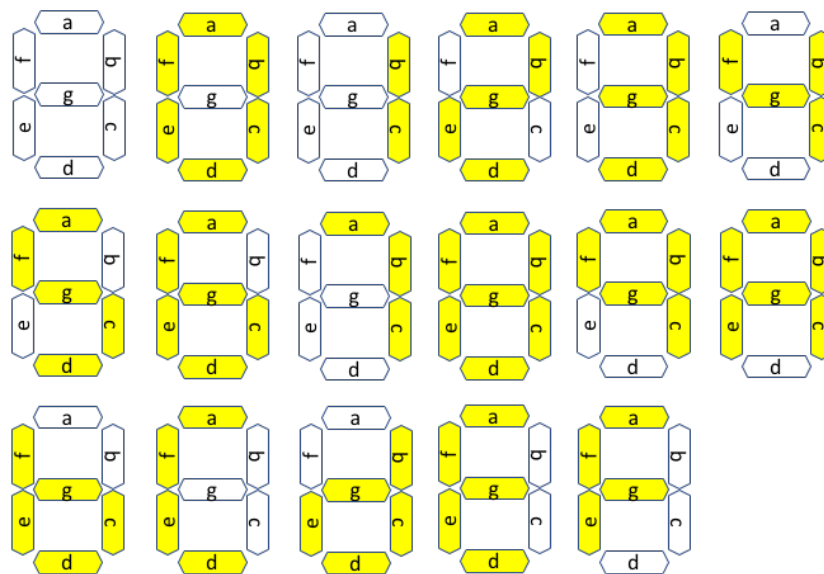
Εικόνα 23. Συσσωρευτής

Πριν ελέγξουμε την ορθότητα της σχεδίασης με χρονική εξομοίωση, ελέγχουμε τα αποτελέσματα της στατικής χρονικής ανάλυσης προκειμένου να βρούμε την συχνότητα ρολογιού που θα χρησιμοποιήσουμε. Η συχνότητα αυτή είναι περίπου 420.17 MHz (*Restricted FMax* στο *Slow Model Fmax Summary*) 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-hold time κατά την θετική ακμή). Επιβεβαιώνουμε ότι το κύκλωμα λειτουργεί σωστά.

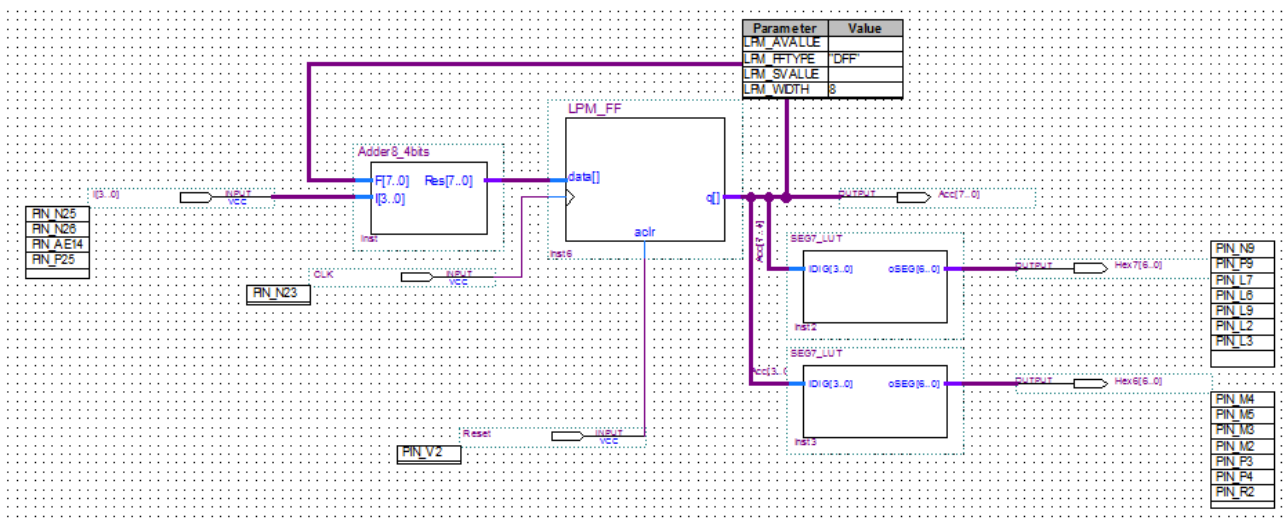
Υλοποίηση στο Board

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 (Switches), καθώς και τα 7segment displays της πλακέτας. Μπορούμε να οδηγήσουμε τους διακόπτες SW3...SW0 στις εισόδους I[3..0] με τον τρόπο που μάθαμε στην Άσκηση 1. Οδηγούμε επιπλέον το σήμα Reset από έναν ακόμη διακόπτη, και το σήμα CLK από ένα Pushbutton. Για τα 7segment displays πρέπει πάλι να ακολουθήσουμε την ίδια τεχνική, ωστόσο θα χρειαστεί προηγουμένως να παρεμβάλουμε έναν αποκωδικοποιητή ο οποίος μετατρέπει μία δυαδική τιμή τεσσάρων δυαδικών ψηφίων σε 7 εισόδους ελέγχου του 7seg display. Για να καταλάβουμε πως λειτουργεί το 7seg display θα πρέπει να δούμε την Εικόνα 24.



Εικόνα 24. 7 segment display

Παρατηρούμε ότι το 7 segment display έχει 7 leds, τα a, b, c, d, e, f και g και κάθε ένα από αυτά ανάβει και σβήνει ανεξάρτητα από τα υπόλοιπα με την χρήση αντίστοιχης εισόδου ελέγχου. Έτσι για παράδειγμα το '1' έχει αναμμένα τα led b, c, και το '5' έχει αναμμένα τα led a, c, d, f, g. Κάθε 7seg display του DE2 είναι συνδεδεμένο με κάποια pins του Cyclone, και συγκεκριμένα οι εισόδους a, b, c, d, e, f του κάθε display είναι συνδεδεμένες σε κάποιο από τα pins του Cyclone. Συνεπώς, για να μπορέσουμε να προβάλλουμε μία δεκαεξαδική τιμή που είναι αποθηκευμένη σε κάποιον καταχωρητή σε δυαδική μορφή θα πρέπει να παρεμβάλουμε έναν αποκωδικοποιητή ο οποίος θα ανάβει τα κατάλληλα leds για κάθε δεκαεξαδική τιμή (παρόμοιους αποκωδικοποιητές έχουμε σχεδιάσει στην Ψηφιακή Σχεδίαση I).



Εικόνα 25

Μπορείτε λοιπόν να τον σχεδιάσετε μόνοι σας, η και να χρησιμοποιήσετε έναν έτοιμο με το όνομα *SEG7_LUT* που βρίσκεται στην βιβλιοθήκη *Core Library*. Μπορείτε να εισάγετε δύο αποκωδικοποιητές για να οδηγήσετε δύο 7seg displays *Hex7* και *Hex6* όπως ακριβώς εισάγετε τις βασικές πύλες (αντί για την βιβλιοθήκη *Primitives* επιλέξετε την βιβλιοθήκη *CoreLibrary* και κατόπιν το component *SEG7_LUT*). Την είσοδο του *Hex7* οδηγήστε την από τα 4 πιο σημαντικά δυαδικά ψηφία του συσσωρευτή *Acc[7..4]*, και την είσοδο του *Hex6* οδηγήστε την από τα 4 λιγότερο σημαντικά δυαδικά ψηφία του συσσωρευτή *Acc[3..0]*. Τις εξόδους τους θα τις οδηγήσετε στα pins που σας υποδεικνύει το Παράρτημα και αντιστοιχούν στα *Hex7* και *Hex6*. Το τελικό σχηματικό φαίνεται στην Εικόνα 25 και η ανάθεση των pins στην Εικόνα 26. Εκτελέστε σύνθεση και προγραμματίστε το DE2. Επιβεβαιώστε ότι το κύκλωμα λειτουργεί σωστά και δείξτε το στους επιτηρητές σας.

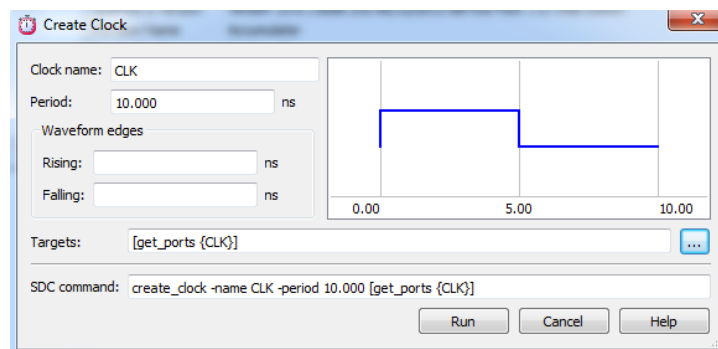
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location
in CLK	Input	PIN_N23	5	B5_N1	PIN_N23
out Hex6[6]	Output	PIN_M4	2	B2_N1	PIN_M4
out Hex6[5]	Output	PIN_M5	2	B2_N1	PIN_M5
out Hex6[4]	Output	PIN_M3	2	B2_N1	PIN_M3
out Hex6[3]	Output	PIN_M2	2	B2_N1	PIN_M2
out Hex6[2]	Output	PIN_P3	1	B1_N0	PIN_P3
out Hex6[1]	Output	PIN_P4	1	B1_N0	PIN_P4
out Hex6[0]	Output	PIN_R2	1	B1_N0	PIN_R2
out Hex7[6]	Output	PIN_N9	2	B2_N1	PIN_N9
out Hex7[5]	Output	PIN_P9	2	B2_N1	PIN_P9
out Hex7[4]	Output	PIN_L7	2	B2_N1	PIN_L7
out Hex7[3]	Output	PIN_L6	2	B2_N1	PIN_L6
out Hex7[2]	Output	PIN_L9	2	B2_N1	PIN_L9
out Hex7[1]	Output	PIN_L2	2	B2_N1	PIN_L2
out Hex7[0]	Output	PIN_L3	2	B2_N1	PIN_L3
in I[3]	Input	PIN_AE14	7	B7_N1	PIN_AE14
in I[2]	Input	PIN_P25	6	B6_N0	PIN_P25
in I[1]	Input	PIN_N26	5	B5_N1	PIN_N26
in I[0]	Input	PIN_N25	5	B5_N1	PIN_N25
in Reset	Input	PIN_V2	1	B1_N0	PIN_V2

Εικόνα 26

Χρονικοί Περιορισμοί με Στατική Χρονική Ανάλυση.

Έχουμε ήδη αναφέρει ότι κατά την διαδικασία της σύνθεσης εκτελείται στατική χρονική ανάλυση η οποία προσδιορίζει την μέγιστη συχνότητα ρολογιού που μπορεί να χρησιμοποιηθεί σε διάφορες

συνθήκες λειτουργίας του FPGA (fast/slow corner). Αυτή η πληροφορία είναι ιδιαίτερα χρήσιμη στον σχεδιαστή καθώς του επιτρέπει να χρησιμοποιήσει την σωστή συχνότητα για το ολοκληρωμένο, ανάλογα με τις απαιτήσεις για κατανάλωση ενέργειας / θερμοκρασία λειτουργίας. Ωστόσο, μία ακόμη χρησιμότερη λειτουργία που παρέχει ο στατικός αναλυτής είναι η καθοδήγηση της σύνθεσης με στόχο την επίτευξη συγκεκριμένων χρονικών περιορισμών. Έτσι για παράδειγμα μπορεί ο σχεδιαστής να καθορίσει την συχνότητα ρολογιού που θέλει να χρησιμοποιήσει, για να επιτρέψει στην σύνθεση να ελαχιστοποιήσει την επιφάνεια για αυτή την συχνότητα, ή να μεγιστοποιήσει την συχνότητα λειτουργίας δηλώνοντας κάποια μονοπάτια ως ψευδή (αργά μονοπάτια που δεν ενεργοποιούνται ποτέ) κλπ. Στην άσκηση αυτή θα μάθουμε να εισάγουμε απλούς περιορισμούς (constraints) και να τους χρησιμοποιούμε στην διαδικασία της σύνθεσης.



Εικόνα 27.

Εκτελούμε σύνθεση στον σχεδιασμό του συσσωρευτή (Start Compilation) και ανοίγουμε τον στατικό αναλυτή από την επιλογή *Tools/Time Quest Timing Analyzer*. Από το κεντρικό μενού επιλέγουμε *Create Timing Netlist* τσεκάροντας πρώτα το *Post Map*, και κατόπιν από το μενού *Constraints* επιλέγουμε *Create Clock* και το παράθυρο που ανοίγει το συμπληρώνουμε όπως φαίνεται στην Εικόνα 27. Το πεδίο Clock Name το συμπληρώνουμε μόνοι μας όπως και το πεδίο Period, ενώ το πεδίο Targets το συμπληρώνουμε μέσα από την εύρεση των σημάτων εισόδου με διαδικασία αντίστοιχη με εκείνη που χρησιμοποιούμε στην παραγωγή κυματομορφών (πίεστε το πλήκτρο «...» και προχωρήστε την διαδικασία επιλέγοντας την είσοδο που αντιστοιχεί στο ρολόι του συστήματος). Το πεδίο SDC command συμπληρώνεται αυτόματα. Με αυτές τις ρυθμίσεις έχουμε ορίσει ως ρολόι μας το σήμα CLK και έχουμε δηλώσει ως επιθυμητή περίοδο τα 10ns (συχνότητα 100MHz). Τέλος εκτελούμε RUN και πηγαίνουμε στο πεδίο Report όπου παρατηρούμε ένα ερωτηματικό δίπλα στην επιλογή *TimeQuest Analyzer Summary*. Με δεξί κλικ του ποντικιού πάνω του επιλέγουμε *Regenerate all Out of Date* (την πρώτη φορά δεν υπάρχει η επιλογή αφού δεν έχει δημιουργηθεί κανένα αρχείο, οπότε μπορούμε να κάνουμε απλώς Generate). Τέλος πηγαίνουμε στην τελευταία επιλογή του Task “*Write SDC File*” και με διπλό κλικ αποθηκεύουμε τους περιορισμούς στο αρχείο “*Accumulator.out.sdc*” που μας προτείνει το εργαλείο.

Επανερχόμαστε στο περιβάλλον του Quartus όπου θα προσθέσουμε στα αρχεία της σχεδίασης και το αρχείο με τους περιορισμούς που δημιουργήσαμε. Επιλέγουμε *Project>Add/Remove Files in Project..* και δίπλα στο *File Name* πιέζουμε τις τρεις τελείες οπότε στο νέο παράθυρο που εμφανίζεται εντοπίζουμε το αρχείο που δημιουργήσαμε (για να εμφανιστεί πρέπει να επιλέξουμε δίπλα στο *File Name* το *All Files (*.*)*). Αφού το επιλέξουμε πατάμε Add, Apply, OK με αυτή την σειρά οπότε τώρα μπορούμε να δούμε ότι έχει προστεθεί στα αρχεία του σχεδιασμού μας. Εκτελούμε και πάλι σύνθεση και ανοίγουμε την αναφορά του *TimeQuest Analyzer>Slow Model>Fmax Summary* η οποία μας ενημερώνει ότι με την τοποθέτηση που έχει επιλέξει η σύνθεση η μέγιστη συχνότητα λειτουργίας είναι 281.53 MHz. Αν και το κύκλωμα δεν είναι ιδιαίτερα περίπλοκο η συχνότητα είναι μικρή καθώς η σύνθεση προσπάθησε να χρησιμοποιήσει τις πιο αργές resources οι οποίες δεν παραβιάζουν τον περιορισμό της συχνότητας. Με αυτή την τακτική η σύνθεση διατηρεί τα resources που διαθέτει για άλλα πιο απαιτητικά κυκλώματα που μπορεί να συντεθούν στην συνέχεια.

Επαναλαμβάνουμε την ίδια διαδικασία θέτοντας αυτή τη φορά περίοδο ρολογιού 2 στον περιορισμό. Αυτή την φορά η σύνθεση μας δίνει το Warning “*Timing Requirements Not Met*” καθώς η ζητούμενη συχνότητα (500 MHz) δεν μπορεί να ικανοποιηθεί. Αυτή την φορά η αναφορά *Slow Model*

του *TimeQuest Analyzer* είναι κοκκινισμένη και μας δίνει μέγιστη συχνότητα 450.05 MHz περίπου. Αυτή είναι και η μέγιστη συχνότητα που μπορούμε να πετύχουμε. Αν θέσουμε αυτή την συχνότητα ως περιορισμό η σύνθεση θα ολοκληρωθεί χωρίς το Warning αυτή τη φορά. Ελέγξτε το κα δείξτε το στους επιτηρητές σας.

Παραδοτέα 2^{ης} Εργαστηριακής Άσκησης

1. Τροποποιήστε κατάλληλα το κύκλωμα του συσσωρευτή ώστε να μπορεί να εκτελεί μέτρηση προς τα πάνω και προς τα κάτω με το βήμα που θα ορίζει κάθε φορά ο χρήστης. Μπορείτε να χρησιμοποιήσετε ένα Switch που θα επιλέγει την μέτρηση προς τα πάνω ή προς τα κάτω. Για την πρόσθεση/αφαίρεση χρησιμοποιήστε αριθμητική συμπληρώματος ως προς 2.
2. Παρουσιάστε τα αποτελέσματα στους επιτηρητές σας.
3. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε καθώς και της στατικής ανάλυσης. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

Εργαστηριακή Άσκηση 3: Υλοποίηση απλών Data-Path και χρήση του LCD

Στην 3^η εργαστηριακή άσκηση θα δούμε αρχικά πως μπορούμε να χρησιμοποιήσουμε το LCD για την προβολή τιμών, και κατόπιν θα σχεδιάσουμε ένα απλό data-path που θα υπολογίζει τους πρώτους όρους της ακολουθίας Fibonacci.

Χρήση του LCD Display για τον συσσωρευτή.

Ας δούμε πως μπορούμε να χρησιμοποιήσουμε το LCD για να προβάλουμε το περιεχόμενο του συσσωρευτή που σχεδιάσαμε στην προηγούμενη εργαστηριακή άσκηση. Για να μπορέσουμε να αξιοποιήσουμε τα κυκλώματα που σχεδιάσαμε στην 2^η εργαστηριακή άσκηση θα πρέπει να αντιγράψουμε τα αρχεία του project της 2^{ης} άσκησης στον κατάλογο της 3^{ης} άσκησης. Για να γίνει αυτή η μεταφορά σωστά θα πρέπει να ακολουθήσουμε την διαδικασία που υποστηρίζει το Quartus και είναι η ακόλουθη:

1. Ανοίγουμε το project της 2^{ης} εργαστηριακής άσκησης.
2. Εκτελούμε Project > Archive Project και στο όνομα αφήνουμε το Accumulator.
3. Κλείνουμε το Quartus και από τον κατάλογο της 2^{ης} εργαστηριακής άσκησης αντιγράφουμε το αρχείο Accumulator.qar στον κατάλογο της 3^{ης} εργαστηριακής άσκησης.
4. Μεταβαίνουμε στον κατάλογο της 3^{ης} εργαστηριακής άσκησης και κάνουμε διπλό κλικ στο αρχείο Accumulator.qar. Αυτόματα ανοίγει το Quartus και μας ζητάει το path όπου θα αποσυμπιέσει το project. Προσέξτε ότι επιλέγει το τρέχων path και προσθέτει το όνομα Accumulator στο τέλος. Αν δεν θέλουμε να φτιάξει νέο κατάλογο με το όνομα Accumulator μπορούμε απλά να το σβήσουμε και να αφήσουμε το υπόλοιπο path.

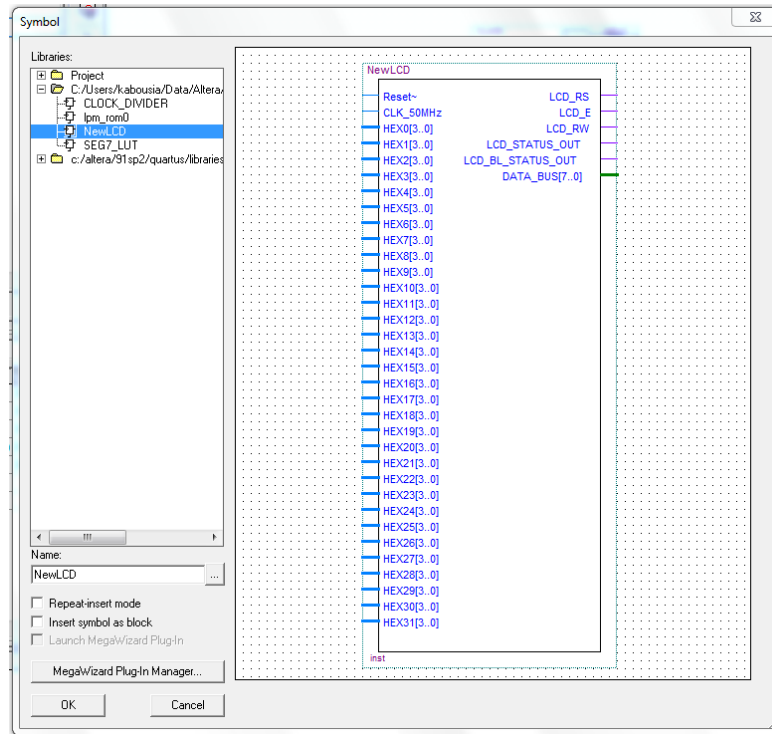
Τώρα πλέον έχουμε τα αρχεία project στον κατάλογο μας. Ανοίγουμε και πάλι το σχηματικό του Accumulator και αφαιρούμε τους αποκωδικοποιητές των Hex Displays για να τους αντικαταστήσουμε με το LCD Display, και μέσα από τον Assignment Editor (μενού Assignment) σβήνουμε όλες τις αναθέσεις των pins. Το LCD 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 σύμβολο για τον Accumulator και κατόπιν δημιουργούμε ένα νέο σχηματικό με όνομα AccumulatorLCD στο οποίο αρχικά εισάγουμε τον πυρήνα διαχείρισης του LCD Display που βρίσκεται στην βιβλιοθήκη *CoreLibrary* (αντί για την βιβλιοθήκη *Primitives* επιλέξτε την βιβλιοθήκη *CoreLibrary* και κατόπιν το component *NewLCD*, όπως φαίνεται στην Εικόνα 28). 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 28. 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	N	O
--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	n	o
-- 7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Εικόνα 29. 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 29. 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 30 (εάν 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 30. 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 29 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 LCD.hex στον κατάλογο του project. Αυτό το αρχείο θα αναζητηθεί αυτόματα κατά την σύνθεση οπότε θα πρέπει να βρεθεί στο project σας για να λειτουργήσει σωστά το LCD.

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

Σχηματικό συσσωρευτή με LCD.

32

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 Switches και τις οποίες τροφοδοτούμε στις εισόδους I[3..0] του συσσωρευτή. Η τιμή που θα δοθεί στα Switches 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 31. Τέλος κάνουμε τις αναθέσεις των pins που φαίνονται στην Εικόνα 32. 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. Δείτε 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_I3	2	B2_N1	3.3-V LVTTTL (default)	
DBUS[4]	Output	PIN_I4	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_I2	2	B2_N1	3.3-V LVTTTL (default)	
DBUS[0]	Output	PIN_I1	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 32.

Υπολογισμός Ακολουθίας Fibonacci.

Στο 2^ο μέρος της 3^{ης} εργαστηριακής άσκηση θα εφαρμόσουμε όσα μάθαμε στις προηγούμενες εργαστηριακές ασκήσεις για να σχεδιάσουμε ένα απλό data-path που θα υπολογίζει τους πρώτους όρους της ακολουθίας Fibonacci. Αρχικά σχεδιάστε δυο ολισθητές παράλληλης φόρτωσης και έναν αθροιστή των 8 δυαδικών ψηφίων. Επιβεβαιώστε την σωστή λειτουργία τους και χρησιμοποιήστε τους για να σχεδιάσετε ένα ακολουθιακό κύκλωμα που θα υπολογίζει τους 8 πρώτους όρους της ακολουθίας Fibonacci. Για την σχεδίαση του κυκλώματος ακολουθήστε τις παρακάτω οδηγίες:

- Διαμορφώστε αρχικά τους δύο ολισθητές A, B έτσι ώστε να μπορούν να φορτωθούν σειριακά από εξωτερική είσοδο SI η οποία οδηγείται από τα DipSwitches όπως φαίνεται στο ακόλουθο σχήμα. Ο καταχωρητής A θα περιέχει σε κάθε κύκλο ρολογιού τον N όρο της ακολουθίας και ο καταχωρητής B τον N-1 όρο.



- Συνδέστε τον αθροιστή με τέτοιο τρόπο ώστε να προσθέτει τα περιεχόμενα των καταχωρητών A, B και να αποθηκεύει σε έναν από τους δύο καταχωρητές το αποτέλεσμα (προσοχή: επιλέξτε τον κατάλληλο).
- Επεκτείνετε το κύκλωμα με τις κατάλληλες συνδέσεις για να πραγματοποιεί τον υπολογισμό της ακολουθίας.
- Η επιλογή της σειριακής φόρτωσης/υπολογισμού θα γίνεται μέσω εξωτερικής εισόδου επιλογής που θα οδηγείται από τα DipSwitches.
- Συνδέστε την έξοδο του καταχωρητή A, του B καθώς και την έξοδο του αθροιστή στο LCD display για να παρακολουθείτε τις τιμές της ακολουθίας. Δώστε την κατάλληλη σήμανση στην κάθε τιμή που εμφανίζεται στο LCD.

Εκτελέστε εξομοίωση του κυκλώματος (χωρίς να συμπεριλάβετε σε αυτήν το LCD display) ως εξής: αρχικά φορτώστε σειριακά τους δύο ολισθητές με τους δύο πρώτους όρους της ακολουθίας (1,1), και κατόπιν εκτελέστε τον υπολογισμό:

1^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

2^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

3^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

4^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

5^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

6^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

7^{ος} Κύκλος Υπολογισμού: **1 1 2 3 5 8 13 21**

κλπ

Κατά την διάρκεια της εξομοίωσης παρακολουθήστε τις τιμές των δύο καταχωρητών καθώς και το αποτέλεσμα του αθροιστή συνδέοντας τους σε θύρες εξόδου (οι θύρες εξόδου μας βοηθούν να παρακολουθούμε τα περιεχόμενα των καταχωρητών). Αφού επαληθεύσετε την σωστή λειτουργία του κυκλώματος προγραμματίστε το FPGA και δείξτε την λειτουργία του Data-Path στο LCD.

Παραδοτέα 3^{ης} Εργαστηριακής Άσκησης

1. Παρουσιάστε τα αποτελέσματα των εξομοιώσεων στους επιτηρητές σας.
2. Προγραμματίστε το FPGA και δείξτε την σωστή λειτουργία του κυκλώματος που σχεδιάσατε στους επιτηρητές.
3. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

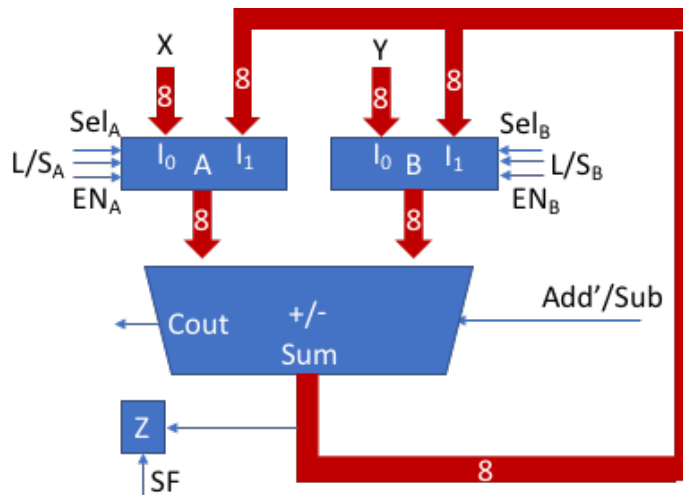
Εργαστηριακή Άσκηση 4

Στόχος της 4^{ης} εργαστηριακής άσκησης είναι η σχεδίαση ενός ολοκληρωμένου data-path το οποίο περιλαμβάνει και μονάδα ελέγχου. Συγκεκριμένα, θα υλοποιήσουμε ένα data-path καθώς και μία αλγοριθμική μηχανή καταστάσεων για τον υπολογισμό της παρακάτω έκφρασης

If $(X-Y \neq 0)$ then $X \leftarrow X+Y$ else $Y \leftarrow 2Y$; όπου X, Y : 8-bit προσημασμένοι αριθμοί

Οι αριθμοί X, Y θα φορτώνονται παράλληλα από τα DipSwitches στην εκκίνηση του υπολογισμού και το περιεχόμενο των X, Y σε κάθε κύκλο ρολογιού θα αποτυπώνεται στο LCD display.

Το ζητούμενο κύκλωμα θα υλοποιηθεί σε δύο στάδια. Αρχικά θα σχεδιάσουμε το data-path και θα ελέγξουμε την λειτουργία του, και κατόπιν θα σχεδιάσουμε την αλγοριθμική μηχανή για τον αυτόματο έλεγχο του data-path κατά τον υπολογισμό της παραπάνω έκφρασης. Το data-path θα πρέπει να έχει δύο καταχωρητές για την αποθήκευση των τιμών X, Y , έναν αθροιστή/αφαιρέτη για τον υπολογισμό της πρόσθεσης/αφαίρεσης, και μία σημαία μηδενικού αποτελέσματος Z . Μία δόκιμη υλοποίηση φαίνεται στο ακόλουθο σχήμα



Στο παραπάνω κύκλωμα οι δύο καταχωρητές A, B έχουν την δυνατότητα να επιλέξουν ανάμεσα σε δύο παράλληλες εισόδους με την χρήση των σημάτων Sel_A, Sel_B και να εκτελέσουν παράλληλη φόρτωση μίας εκ των δύο, ή αριστερή ολίσθηση με χρήση των σημάτων $L/S_A, L/S_B$. Στο συγκεκριμένο datapath ο καταχωρητής A θα χρησιμοποιηθεί για να αποθηκεύσει είτε α) την τιμή X είτε β) την έξοδο του αθροιστή, ο καταχωρητής B θα αποθηκεύει είτε το Y είτε θα κάνει αριστερή ολίσθηση, και το flip flop Z θα χρησιμοποιηθεί για την σημαία του μηδενικού αποτελέσματος (το κρατούμενο εξόδου μπορείτε να το αγνοήσετε). Όλοι οι καταχωρητές και τα flip flop μηδενίζονται αρχικά μέσω σήματος Reset το οποίο συνδέεται στα DipSwitches. Το σήμα CLK οδηγείται από τους χειροκίνητους διακόπτες επαναφοράς ρολογιού.

Μέρος Α: Σχεδίαση του Data Path

1. Σημαία Z μηδενικού αποτελέσματος

Αποθηκεύεται σε ένα JK flip-flop το οποίο πρέπει να διαμορφωθεί κατάλληλα ώστε να παρέχει τις λειτουργίες της αποθήκευσης και της διατήρησης των δεδομένων με την χρήση ενός σήματος ενεργοποίησης SF (Store Flag). Όταν $SF=0$ η σημαία που είναι ήδη αποθηκευμένη διατηρείται, ενώ όταν $SF=1$ η σημαία ενημερώνεται από το αποτέλεσμα του αθροιστή/αφαιρέτη στην επόμενη θετική ακμή του ρολογιού. Τα σήματα CLK και CLR του JK flip flop οδηγούνται από τα

γενικότερα σήματα CLK, Reset που οδηγούν και τα υπόλοιπα ακολουθιακά στοιχεία του κυκλώματος (το σήμα preset, αν υπάρχει, διατηρείται απενεργοποιημένο στην μονάδα).

2. Καταχωρητές Δεδομένων A, B

Για την σωστή λειτουργία του DataPath οι καταχωρητές A, B πρέπει να έχουν την δυνατότητα της παράλληλης φόρτωσης μίας εκ των δύο επιλεγόμενων εισόδων, της αριστερής ολίσθησης και της διατήρησης δεδομένων. Μέσω της αριστερής ολίσθησης ο καταχωρητής B θα μπορεί να εκτελέσει την πράξη $Y \leftarrow 2Y$. Η επίτρεψη (enable) κάθε καταχωρητή γίνεται με την χρήση σήματος EN (EN_A , EN_B) το οποίο είναι ενεργό στην τιμή 1 (όταν $EN=0$ ο καταχωρητής απλά διατηρεί τα δεδομένα του). Η επιλογή της λειτουργίας παράλληλης φόρτωσης ($L/S=0$) / αριστερής ολίσθησης ($L/S=1$) γίνεται με χρήση του σήματος L/S (L/S_A , L/S_B). Προσοχή: και οι δύο λειτουργίες Load – Shift απαιτούν ταυτόχρονα το σήμα $EN = 1$. Υλοποιήστε αρχικά τους καταχωρητές A, B και επιβεβαιώστε πειραματικά την λειτουργία τους. Συνδέστε και τους δύο καταχωρητές στο σήμα Reset ώστε να μηδενίζονται αρχικά.

Αθροιστής/Αφαιρέτης

Υλοποιήστε έναν αθροιστή-αφαιρέτη σε αριθμητική συμπληρώματος ως προς 2 σύμφωνα με όσα γνωρίζετε από την Ψηφιακή Σχεδίαση I. Η επιλογή πρόσθεσης/αφαίρεσης θα γίνεται μέσω εξωτερικού σήματος Add/Sub (0 για πρόσθεση, 1 για αφαίρεση) και το αποτέλεσμα της πράξης θα καθορίζει την τιμή της σημαίας Z (όταν τα 8 δυαδικά ψηφία του αποτελέσματος είναι 0 και το σήμα επίτρεψης αποθήκευσης $SF=1$ τότε η σημαία τίθεται στο 1 στην επόμενη θετική ακμή του ρολογιού). Το κρατούμενο εισόδου χρησιμοποιήστε το κατάλληλα για να υλοποιήσετε την πρόσθεση αφαίρεση (δεν απαιτείται χρήση κρατούμενου από προηγούμενη βαθμίδα). Προσέξτε ότι το datapath απαιτεί την αφαίρεση του τελούμενου B από το A ($A-B$), αλλά όχι και το αντίστροφο ($B-A$).

Διασύνδεση μονάδων

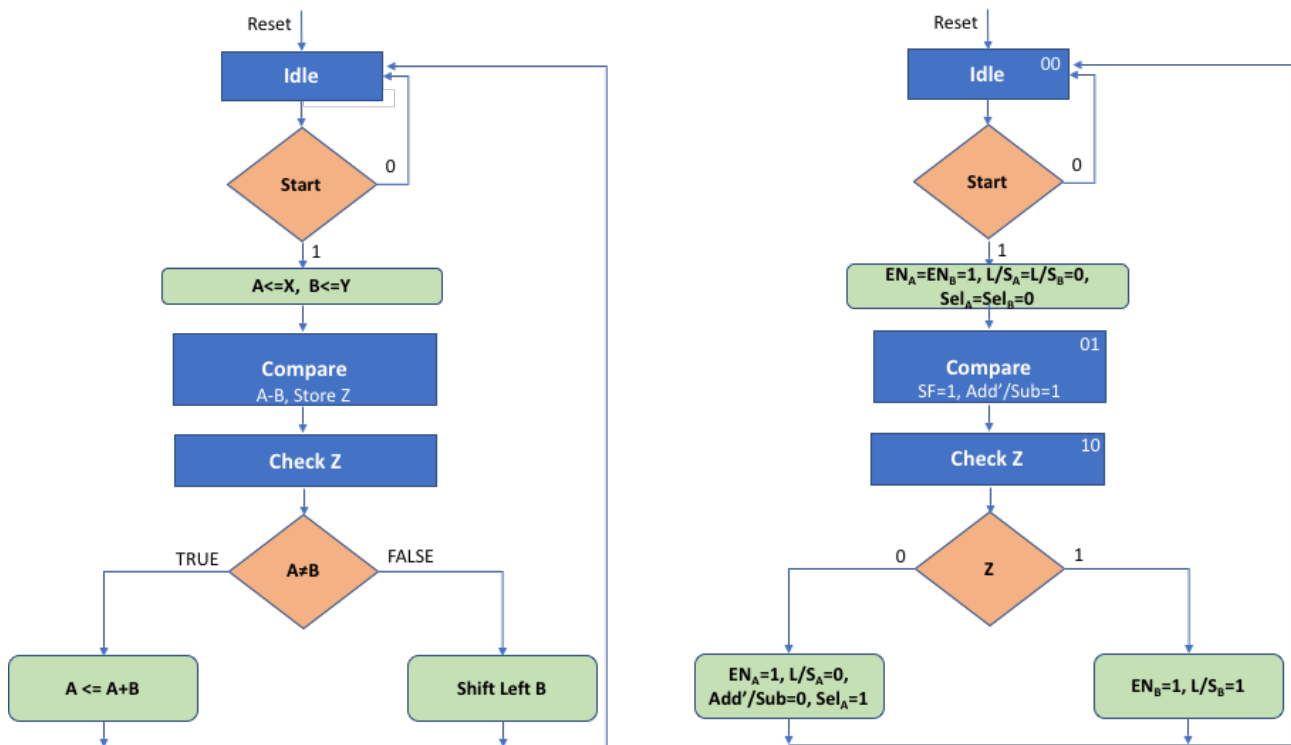
Διασυνδέστε τις μονάδες που σχεδιάσατε σύμφωνα με το κύκλωμα του data path, και προσθέστε όποιες μονάδες θεωρείτε απαραίτητες για να δοθεί η δυνατότητα της παράλληλης φόρτωσης των καταχωρητών A, B από εξωτερικές εισόδους X, Y όταν ένα σήμα Start είναι στο λογικό 1. Ελέγξτε την λειτουργία του συνολικά χρησιμοποιώντας χρονική εξομοίωση. Δείξτε την και στον επιτηρητή σας. Εκτός των σημάτων CLK, Reset που μοιράζονται όλες οι μονάδες, τα υπόλοιπα σήματα αφορούν αποκλειστικά την κάθε μονάδα χωριστά, και θα πρέπει να οδηγηθούν προσωρινά από εισόδους που θα οδηγήσετε από το αρχείο κυματομορφών με κατάλληλες τιμές ώστε να λειτουργεί το κύκλωμα σωστά. Σε αυτό το στάδιο δεν χρειάζεται επιβεβαίωση στο board. Ο συνολικός πίνακας λειτουργιών που παρέχει το datapath είναι ο ακόλουθος

	Περιγραφή	0 (Low)	1 (High)
ADD'/SUB	Πρόσθεση/Αφαίρεση	Πρόσθεση	Αφαίρεση
EN (EN_A, EN_B)	Επίτρεψη Αποθήκευσης	Μη-Ενεργή	Ενεργή
L/S (L/S_A, L/S_B)	Παράλληλη Φόρτωση / Αριστερή Ολίσθηση	$Reg \leftarrow I_0$ ή I_1	$Reg \ll 1$
Sel (Sel_A, Sel_B)	Επιλογή Εισόδου I_0 , I_1	I_0	I_1
SF	Επίτρεψη Αποθήκευσης	Μη-Ενεργή	Ενεργή

Μέρος Β: Σχεδίαση του Control Unit

Διάγραμμα ASMD

Για να υλοποιήσουμε την μονάδα ελέγχου του παραπάνω data path θα πρέπει να σχεδιάσουμε αρχικά το διάγραμμα ASMD το οποίο θα μας καθοδηγήσει στην σχεδίαση του πίνακα μετάβασης καταστάσεων. Η εκκίνηση θα γίνεται με την χρήση σήματος Start το οποίο ενεργοποιείται στην τιμή '1'. Στην αρχική κατάσταση το κύκλωμα θα βρεθεί μετά την ενεργοποίηση του σήματος Reset, ή κατά την διάρκεια της λειτουργίας του με επαναφορά μετά την ολοκλήρωση του υπολογισμού.



Όπως φαίνεται από το διάγραμμα το κύκλωμα αρχικά εισέρχεται στην κατάσταση **Idle (00)** στην οποία αναμένει την έναρξη υπολογισμού από το σήμα Start το οποίο θα τεθεί στην μονάδα. Μόλις το σήμα Start τεθεί στο 1 οι καταχωρητές A, B φορτώνονται παράλληλα από 16 Switches στην επόμενη θετική ακμή του ρολογιού και ξεκινά ο υπολογισμός περνώντας στην κατάσταση **Compare (01)**. Σε αυτή την κατάσταση εκτελείται η αφαίρεση A-B και υπολογίζεται η σημαία μηδενικού αποτελέσματος (Z) χωρίς να αποθηκεύεται το αποτέλεσμα της αφαίρεσης. Όταν προκύπτει Z=1 τότε προφανώς A=B, ενώ όταν προκύπτει Z=0 τότε A≠B. Έτσι ακολουθεί η κατάσταση **CheckZ (10)** η οποία επιλέγει την ενέργεια B<=2B στην πρώτη περίπτωση και A<=A+B στην δεύτερη. Προσέξτε ότι η τιμή 2B υπολογίζεται με αριστερή ολίσθηση του καταχωρητή B κατά 1 δυαδικό ψηφίο (η σειριακή είσοδος του καταχωρητή B πρέπει να είναι στην τιμή '0' για να εκτελεστεί σωστά η ολίσθηση). Μετά τον υπολογισμό η ακολουθιακή μηχανή επιστρέφει στην αρχική κατάσταση **Idle (00)**. Το σήμα Sel_A χρησιμοποιείται για την φόρτωση του καταχωρητή A. Φτιάξτε τον πίνακα μετάβασης καταστάσεων με βάση το παραπάνω διάγραμμα, και σχεδιάστε την αλγοριθμική μηχανή χρησιμοποιώντας 2 D ff και έναν αποκωδικοποιητή 2 σε 4 σύμφωνα με όσα μάθαμε στην θεωρία. Ελέγξτε με χρονική εξομοίωση την λειτουργία του κυκλώματος και δείξτε την στους επιτηρητές σας. Όλες τις κυματομορφές χρησιμοποιήστε τις για την σύνταξη της τελικής αναφοράς.

Υλοποίηση στο board

Για να ολοκληρώσουμε την σχεδίαση θα πρέπει να προγραμματίσουμε το Data-path στο board. Για τον λόγο αυτό πρέπει να προσέξουμε τα ακόλουθα:

1. Οι καταχωρητές A, B θα φορτώνονται παράλληλα σε ένα κύκλο όταν το σήμα Start = 1 από 16 switches SW15 – SW8 (τιμή X) και SW7 – SW0 (τιμή Y). Στον αμέσως επόμενο κύκλο το Start θα επανέρχεται στο 0.
2. Τα περιεχόμενα των καταχωρητών A, B θα προβάλλονται στο LCD συνεχώς.
3. Το αποτέλεσμα της ALU θα προβάλλεται σε δύο 7-segment displays του board: οδηγήστε τις εξόδους της ALU στις εισόδους δύο SEG7_LUT και συνδέστε τις εξόδους των συμβόλων SEG7_LUT με τα αντίστοιχα pin εξόδου HEX7[6..0] και HEX6[6..0] τα οποία θα βρείτε στο παράρτημα.
4. Η τιμή της σημαίας Z θα προβάλλεται σε ένα από τα LEDs συνεχώς.
5. Τρέξτε το κύκλωμα για διαδοχικούς κύκλους λειτουργίας μονάδος ASM, όπου θα δίνετε διαδοχικά τις τιμές $(X, Y) = (127, 32), (64, 85), (-2, 1), (-50, -50)$. Η εναλλαγή των τιμών γίνεται με την βοήθεια του σήματος Start. Καταγράψτε τα περιεχόμενα των A, B στο τέλος κάθε κύκλου της ASM.

Παραδοτέα 4^{ης} Εργαστηριακής Άσκησης

1. Παρουσιάστε τα αποτελέσματα των εξομοιώσεων στους επιτηρητές σας.
2. Προγραμματίστε το FPGA και δείξτε την σωστή λειτουργία του κυκλώματος που σχεδιάσατε στους επιτηρητές.
3. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

Εργαστηριακή Άσκηση 5: Σχεδίαση με VHDL

Στην παρούσα εργαστηριακή άσκηση θα δούμε πως μπορούμε να χρησιμοποιήσουμε την γλώσσα περιγραφής VHDL για να σχεδιάσουμε απλά κυκλώματα. Φυσικά ο κύριος στόχος της VHDL δεν είναι η σχεδίαση απλών κυκλωμάτων, αλλά η σχεδίαση σύνθετων και μεγάλων κυκλωμάτων. Αρχικά όμως θα πρέπει να εξοικειωθούμε με την χρήση της πριν προχωρήσουμε σε πιο σύνθετες σχεδιάσεις. Επιπλέον θα μάθουμε να χρησιμοποιούμε το ModelSim για να εξομοιώσουμε τις HDL περιγραφές αλλά και τα κυκλώματα που έχουμε συνθέσει, ενώ θα δημιουργήσουμε και τα πρώτα μας test-benches. Για να μπορέσουμε όμως να χρησιμοποιήσουμε το ModelSim θα πρέπει προηγουμένως να ρυθμίσουμε κάποιες παραμέτρους του Quartus:

1. Επιλέγουμε *Tools>Options>General>EDA Tool Options* και στην καρτέλα που ανοίγει συμπληρώνουμε την διαδρομή που έχουμε τοποθετήσει το ModelSim ως τιμή της παραμέτρου *ModelSim – Altera*. Το ModelSim εγκαθίσταται μαζί με το Quartus οπότε συνήθως και τα δύο βρίσκονται στον ίδιο κατάλογο *altera\13.0sp1*. Για παράδειγμα μία πιθανή θέση του είναι η «*C:\altera\13.0sp1\modelsim_ase\win32aloem*» αλλά στην δική σας εγκατάσταση μπορεί να διαφέρει.
2. Επιλέγουμε *Assignment>Settings>EDA tool Settings>Simulation* και στην καρτέλα που ανοίγει κάνουμε τις ακόλουθες ρυθμίσεις:
 - Tool Name: *ModelSim – Altera*
 - EDA Netlist Writer Settings: “*Format for Output Netlist: VHDL*”, Output directory: “*simulation/modelsim*”
 - Επιλέγουμε More EDA Netlist Writer Settings και στην επιλογή “Generate netlist for functional simulation only” επιλέγουμε “Off”
3. Επιλέγουμε *Assignment>Settings>Analysis and Synthesis Settings>VHDL Input* και στην καρτέλα που ανοίγει επιλέγουμε VHDL 2008.

Τις παραπάνω ρυθμίσεις πρέπει να ελέγχουμε σε κάθε νέο project που δημιουργούμε για να συνθέσουμε κυκλώματα με VHDL και να τα εξομοιώσουμε με ModelSim.

Μέρος 1^ο. Συνδυαστικά Κυκλώματα

Ερώτημα 1ο

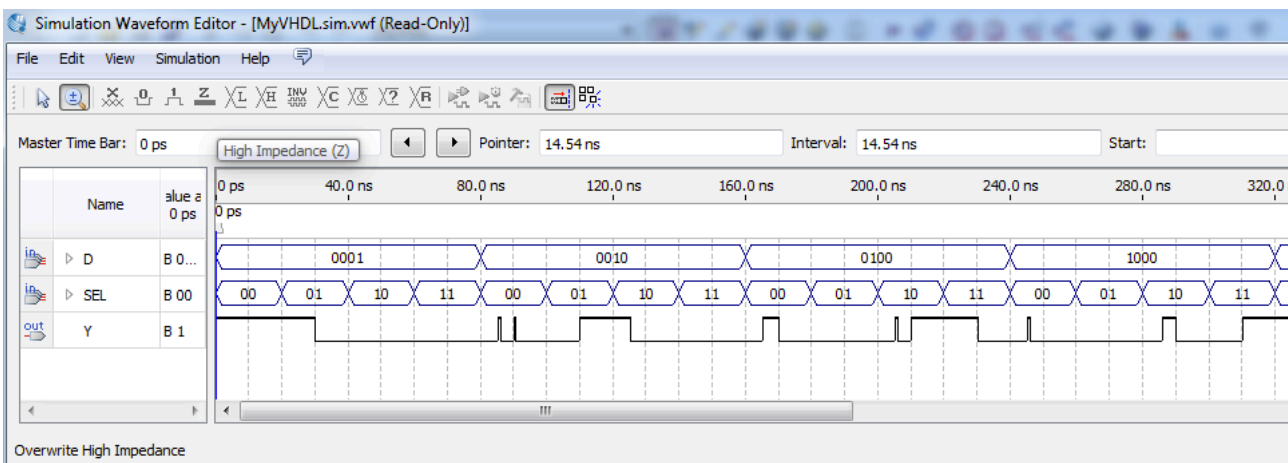
Αρχικά θα σχεδιάσουμε και θα εξομοιώσουμε το κύκλωμα ενός πολυπλέκτη 4 σε 1 με χρήση της γλώσσας VHDL. Δημιουργούμε ένα νέο project και εισάγουμε ένα νέο VHDL αρχείο (επιλογή *File>New>VHDL File*). Αποθηκεύουμε το κενό ακόμη αρχείο με το όνομα *MUX4_1.vhd*. Στον κειμενογράφο που έχει ανοίξει εισάγουμε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

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

entity MUX4_1 is
    port (
        D : in std_logic_vector (3 downto 0);
        SEL: in std_logic_vector (1 downto 0);
        Y: out std_logic);
end MUX4_1;

architecture RTL4_1 of MUX4_1 is
begin
    Y <= D(0) when SEL="00" else D(1) when SEL="01" else D(2) when SEL="10" else D(3);
```

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 ποντικού επιλέγουμε *Insert Template* και κατόπιν επιλέγουμε από το συντακτικό της VHDL την επιλογή *Constructs/Design Units/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 που σας παρέχει το Quartus καθώς θα σας φανούν πολύ χρήσιμες αργότερα. Εκτελούμε compile (αφού πρώτα θέσουμε το τρέχων κύκλωμα ως κορυφαία οντότητα του σχεδιασμού (top level entity) και δημιουργούμε σύμβολο με την γνωστή διαδικασία ώστε να είναι διαθέσιμο στην βιβλιοθήκη μας. Ανοίγουμε τις αναφορές της στατικής ανάλυσης όπου μπορούμε να διαπιστώσουμε ότι στο *Slow Model* η μέγιστη καθυστέρηση του κυκλώματος είναι λίγο πάνω από τα 10 ns.



Εικόνα 33. Κυματομορφή ελέγχου πολυπλέκτη 4 σε 1

Για να ελέγξουμε την λειτουργία του πολυπλέκτη θα πρέπει να δημιουργήσουμε ένα αρχείο κυματομορφών το οποίο θα είναι κατάλληλο για να ασκήσει όλες τις απαραίτητες τιμές στις εισόδους του πολυπλέκτη. Χρησιμοποιώντας μία περίοδο 20 ns (η οποία είναι μεγαλύτερη από την μέγιστη καθυστέρηση του κυκλώματος) δημιουργούμε το αρχείο που φαίνεται στην Εικόνα 33. Από τα αποτελέσματα της χρονικής εξομίσωσης που φαίνονται στην Εικόνα 33 μπορούμε να παρατηρήσουμε τα ακόλουθα:

1. Δεν απαιτείται να ασκήσουμε όλες τις πιθανές τιμές στις εισόδους D και SEL. Αρκεί μόνο να θέσουμε την επιλεγμένη είσοδο στην τιμή 1 και όλες τις υπόλοιπες στο 0 για να ελέγξουμε αν στην έξοδο εμφανίζεται σωστά η επιλεγμένη τιμή (προσέξτε ότι σε περίπτωση λάθους η έξοδος θα είναι στο 0). Το ίδιο πρέπει να κάνουμε και με την επιλεγμένη είσοδο στο 0 και όλες τις υπόλοιπες στο 1 για να επιβεβαιώσουμε και την συμπληρωματική κατάσταση.
2. Παρατηρούμε κάποιες στιγμιαίες μεταβολές της εξόδου οι οποίες οφείλονται στις διαφορετικές καθυστερήσεις των λογικών μονοπατιών του κυκλώματος. Οι μεταβολές αυτές δεν μας ενοχλούν καθώς έχουμε φροντίσει να διατηρούμε σταθερές τις εισόδους για χρόνο μεγαλύτερο από την μέγιστη καθυστέρηση του κυκλώματος.
3. Για κυκλώματα πιο σύνθετα η διαδικασία εισαγωγής τιμών στις εισόδους είναι περίπλοκη και χρονοβόρα. Για τον λόγο αυτό παρακάτω θα μάθουμε την χρήση των test-benches που μας απαλλάσσουν από την διαδικασία χειροκίνητης εισαγωγής των κυματομορφών.

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 σε structural VHDL. Δημιουργήστε ένα νέο αρχείο VHDL και εισάγετε την VHDL που φαίνεται στο ακόλουθο σχήμα:

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

entity MUX16_1 is
    port (    D : in std_logic_vector (15 downto 0);
            SEL: in std_logic_vector (3  downto 0);
            Y: out std_logic );
end MUX16_1;

architecture RTL of MUX16_1 is
    component MUX4_1
        port (    D : in std_logic_vector (3  downto 0);
                SEL: in std_logic_vector (1  downto 0);
                Y: out std_logic );
    end component;
    signal F : std_logic_vector (3  downto 0);
begin
    u0: MUX4_1 port map (D => D(3 downto 0),      SEL => SEL(1 downto 0), Y => F(0));
    u1: MUX4_1 port map (D => D(7  downto 4),      SEL => SEL(1 downto 0), Y => F(1));
    u2: MUX4_1 port map (D => D(11 downto 8),       SEL => SEL(1 downto 0), Y => F(2));
    u3: MUX4_1 port map (D => D(15 downto 12),     SEL => SEL(1 downto 0), Y => F(3));
    u4: MUX4_1 port map (D => F(3  downto 0),       SEL => SEL(3 downto 2), Y => Y);
end RTL;
```

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). Εκτελούμε σύνθεση και επιβεβαιώνουμε ότι η μέγιστη καθυστέρηση στο Slow Model είναι περίπου 12.25ns. Επαναλαμβάνουμε χρονική εξομοίωση και δείχνουμε ότι ο πολυπλέκτης λειτουργεί σωστά.

Ερώτημα 3ο

Στο σημείο αυτό θα πρέπει να έχει γίνει φανερό ότι η εξομοίωση πιο σύνθετων κυκλωμάτων είναι μία χρονοβόρα και επίπονη διαδικασία όταν οι κυματομορφές εισάγονται χειροκίνητα. Για τον λόγο αυτό θα πρέπει να μάθουμε να χρησιμοποιούμε test benches τα οποία μας διευκολύνουν πολύ στην διαδικασία της εξομοίωσης. Ωστόσο, τα test benches απαιτούν εμπειρία για την δημιουργία τους, οπότε θα αρκεστούμε προσωρινά στην αυτόματη παραγωγή τους πάλι με χρήση χειροκίνητων κυματομορφών, και αργότερα θα μάθουμε πως θα τα φτιάχνουμε μόνοι μας διατηρώντας όμως το design flow που θα δούμε εδώ.

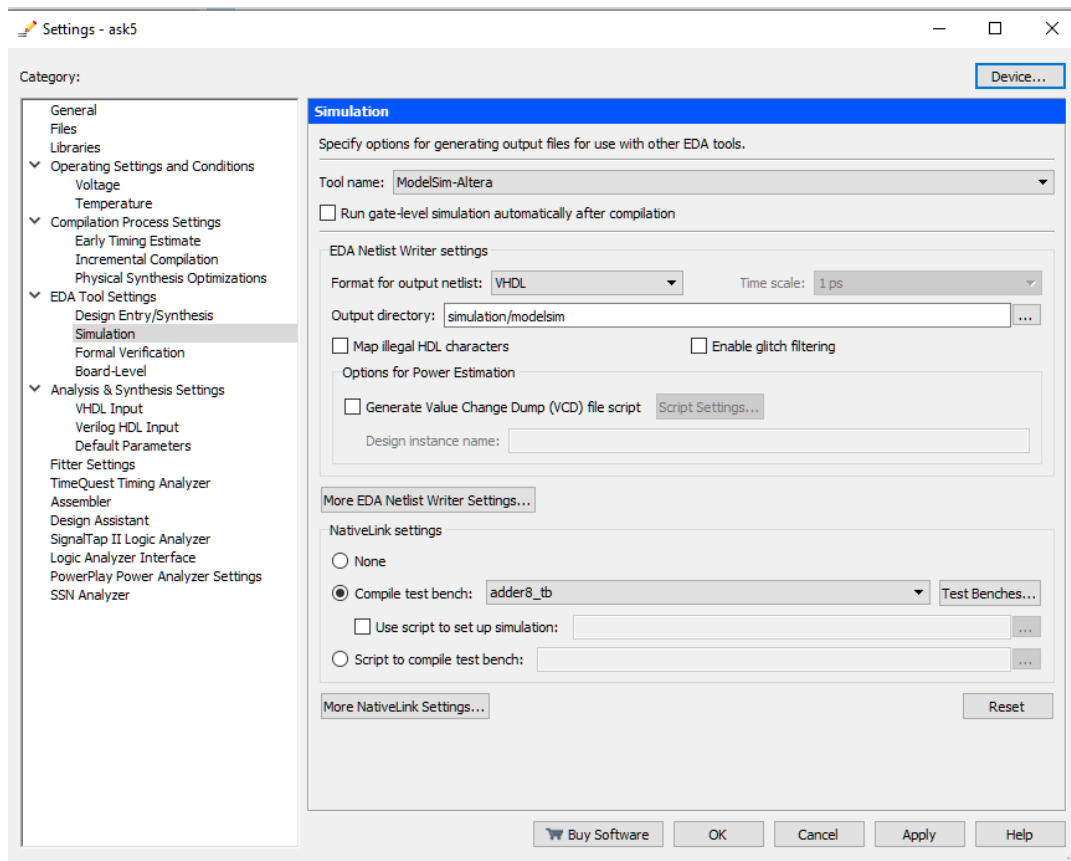
Για να δούμε την διαδικασία της εξομοίωσης με test bench θα σχεδιάσουμε έναν αθροιστή των 8 δυαδικών ψηφίων χρησιμοποιώντας περιγραφή δομής. Αρχικά θα περιγράψουμε έναν πλήρη αθροιστή σε VHDL και κατόπιν θα τον χρησιμοποιήσουμε για να περιγράψουμε έναν αθροιστή ριπής των 8 δυαδικών ψηφίων χρησιμοποιώντας 8 πλήρης αθροιστές σε περιγραφή δομής. Η δήλωση της οντότητας του αθροιστή 8 bits είναι η ακόλουθη (το σώμα της αρχιτεκτονικής δημιουργήστε το μόνοι σας όπως επίσης και τον πλήρη αθροιστή).

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

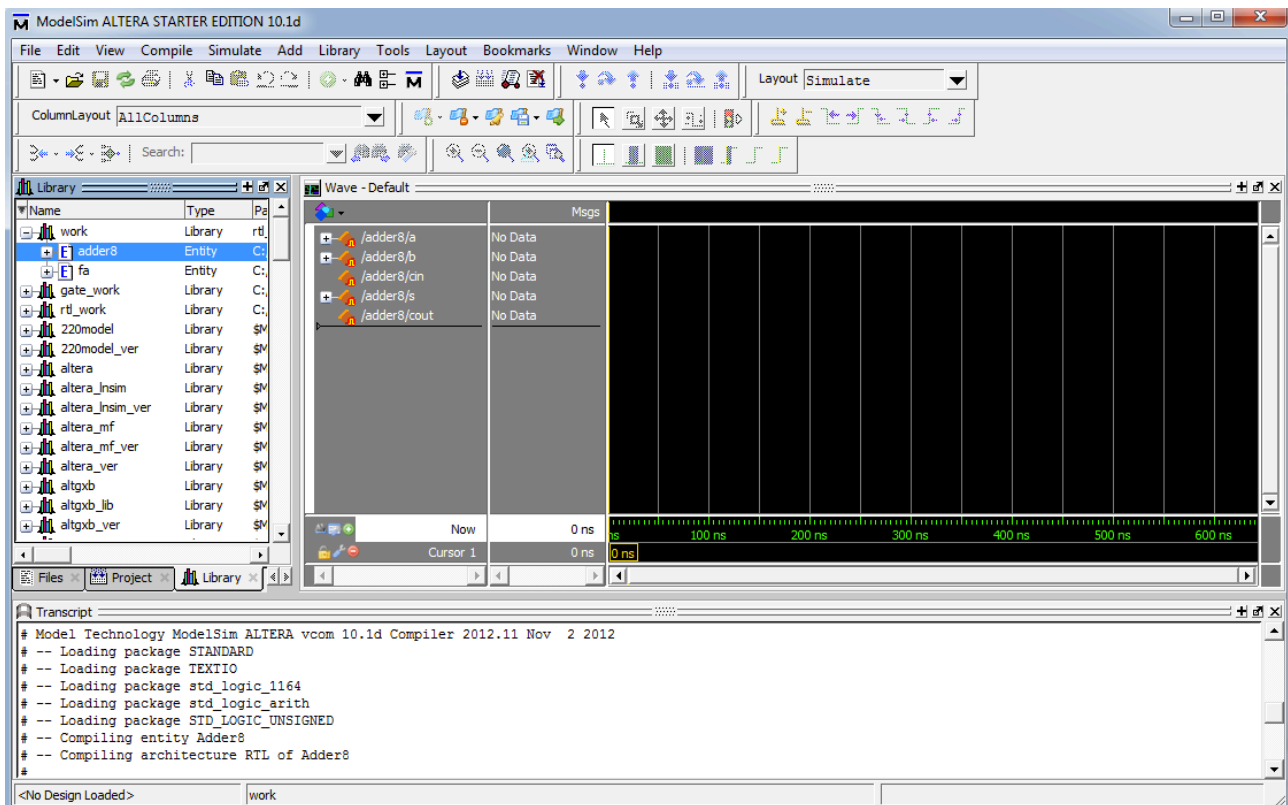
entity Adder8 is
    port (    A, B : in std_logic_vector (7 downto 0);
            Cin : in std_logic;
            Y: out std_logic_vector (7 downto 0);
end Adder8;
```

```
end Adder8; Cout: out std_logic );
```

Εκτελούμε σύνθεση και επιβεβαιώνουμε την συντακτική ορθότητα της περιγραφής. Για να εκτελέσουμε εξομοίωση θα ακολουθήσουμε μία διαφορετική διαδικασία αυτή την φορά. Συγκεκριμένα θα χρησιμοποιήσουμε το ModelSim το οποίο είναι ένα ευρέως χρησιμοποιούμενο εργαλείο για εκτέλεση εξομοιώσεων. Πριν δείξουμε την διαδικασία θα πρέπει να επιβεβαιώσουμε ότι οι ρυθμίσεις του συστήματος είναι σωστές. Για τον λόγο αυτό αρχικά μεταβαίνουμε στην καρτέλα *Assignment>Settings>EDA Tool Settings>Simulation* και επιλέγουμε *Tool name: ModelSim-Altera* όπως φαίνεται στην Εικόνα 34. Κατόπιν επιλέγουμε *Tools>Run Simulation Tool>RTL Simulation* οπότε ανοίγει το παράθυρο εξομοίωσης του ModelSim.

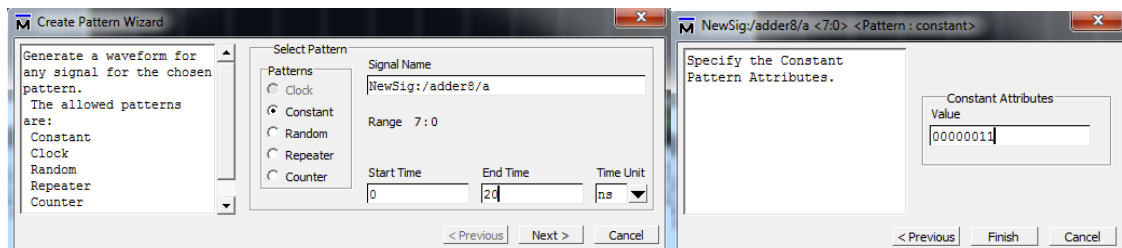


Εικόνα 34. Ρυθμίσεις για εξομοίωση με το ModelSim



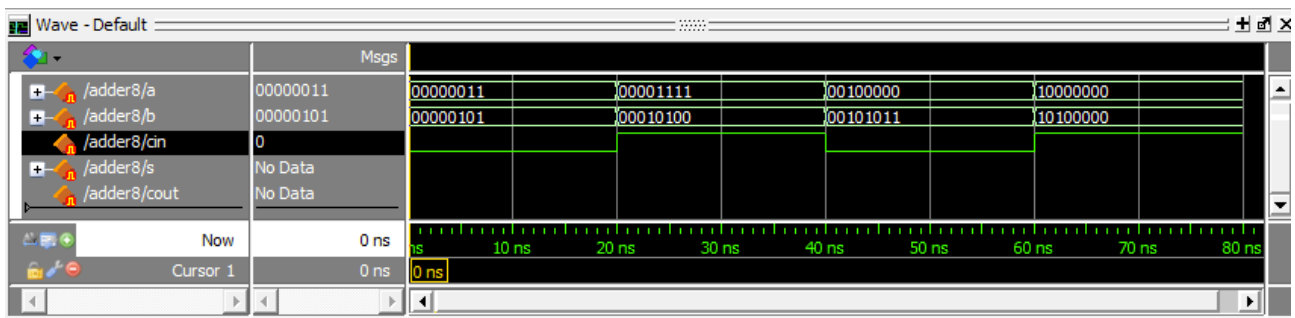
Εικόνα 35. ModelSim

Στο παράθυρο Library επιλέγουμε την βιβλιοθήκη work (είναι η βιβλιοθήκη με τον σχεδιασμό μας) και εκεί τον adder8, όπως φαίνεται στην Εικόνα 35. Με δεξί κλικ του ποντικιού πάνω στον Adder8 εκτελούμε την εντολή *Create Wave* οπότε εμφανίζονται τα σήματα εισόδου/εξόδου του αθροιστή όπως φαίνεται στο *Wave Default* στην Εικόνα 35. Πατάμε δεξί κλικ πάνω στο /Adder8/a και επιλέγουμε *Edit>Create/Modify Waveform* οπότε ανοίγει το παράθυρο που φαίνεται στην Εικόνα 36.



Εικόνα 36. Δημιουργία κυματομορφής

Θα δώσουμε πέντε διαδοχικές τιμές στις εισόδους A, B, Cin για να ελέγξουμε τις προσθέσεις $3+5+(0)$ από 0 έως 20 ns, $15+20+(1)$ από 20 έως 40 ns, $32+43+(0)$ από 40 έως 60 ns και $128+160+(1)$ από 60 έως 80 ns όπου οι τιμές είναι όλες στο δεκαδικό σύστημα και οι τιμές μέσα στις παρενθέσεις είναι το κρατούμενο εισόδου. Τις τιμές αυτές θα δώσουμε ως σταθερές στο δυαδικό σύστημα δηλώνοντας τα αντίστοιχα διαστήματα στα πεδία Start Time, End Time όπως φαίνεται στην Εικόνα 36 για την τιμή 3 στο A. Επαναλαμβάνουμε το ίδιο και για τις υπόλοιπες τιμές οπότε προκύπτουν οι κυματομορφές που βλέπουμε στην Εικόνα 37.



Εικόνα 37

Έχοντας πλέον εισάγει τις τιμές στα σήματα για την εξομοίωση μπορούμε να δημιουργήσουμε αυτόματα το test-bench το οποίο θα παράγει αυτές ακριβώς τις κυματομορφές για να εκτελεστεί η εξομοίωση. Το test-bench δεν είναι τίποτα άλλο από μία μη-συνθέσιμη περιγραφή σε VHDL η οποία μπορεί να παράγει σήματα εισόδου και να παρακολουθεί σήματα εξόδου, ενώ ταυτόχρονα μπορεί να εμφανίζει μηνύματα που παρέχουν πληροφορίες για την πρόοδο της εξομοίωσης. Επιλέγουμε *File>Export>Waveform* και στο παράθυρο διαλόγου που ανοίγει επιλέγουμε *VHDL testbench* και δίνουμε ως όνομα αρχείου το *Adder8_tb*. Προσέξτε ότι ο default κατάλογος στον οποίο τοποθετείται το αρχείο είναι ο κατάλογος *Simulation>ModelSim*. Αφού λοιπόν επιβεβαιώσουμε την δημιουργία του test bench μπορούμε να κλείσουμε το ModelSim και να επιστρέψουμε στο Quartus.

Αρχικά θα εισάγουμε το testbench στο project. Επιλέγουμε *Project>Add/Remove Files in Project* και αναζητούμε το αρχείο στον κατάλογο *simulation/modelsim/Adder8_tb.vhd*. Αφού το προσθέσουμε μπορούμε να το αναζητήσουμε πλέον στο πεδίο Files του Project Navigator και να το ανοίξουμε για να το διαβάσουμε. Αξίζει να δούμε τα περιεχόμενα του ώστε να καταλάβουμε με ποιον τρόπο μπορούμε να το φτιάχνουμε και εμείς χωρίς να χρειάζεται να εισάγουμε κάθε φορά τις κυματομορφές με τον χρονοβόρο τρόπο που περιγράψαμε παραπάνω. Το test bench φαίνεται στην Εικόνα 38. Το αριστερό τμήμα δείχνει τις δηλώσεις του component και των σημάτων που θα χρησιμοποιηθούν ως είσοδοι/έξοδοι, το μεσαίο δείχνει το instantiation του component καθώς και την κυματομορφή για το A ως μία process. Στο δεξιό τμήμα βλέπουμε την κυματομορφή για το B ως μία ξεχωριστή process καθώς και του κρατούμενου εισόδου. Το κρατούμενο εισόδου έχει οριστεί ως ένας μετρητής με περίοδο 20 ns οπότε η κυματομορφή του διαφέρει από τις σταθερές που είδαμε προηγουμένως.

Προσοχή: το βήμα της προσθήκης του test-bench στο project μας είναι προαιρετικό και μας διευκολύνει στην επεξεργασία του. Ωστόσο, μετά την εισαγωγή του το Quartus θα εκτελεί αυτόματα μεταγλώττιση εφόσον θα ανήκει στο project. Αυτό ενδεχομένως να δημιουργήσει κάποια προβλήματα σε συγκεκριμένες δομές που θα χρησιμοποιήσουμε παρακάτω, και δεν θα μας επιτρέπει να επανασυνθέσουμε την σχεδίαση μας σε περίπτωση που θέλουμε να διορθώσουμε κάποιο λάθος. Αυτό οφείλεται σε διαφορά φιλοσοφίας των εργαλείων Quartus και ModelSim, καθώς το ένα στοχεύει στην σύνθεση και το άλλο στην εξομοίωση των περιγραφών. Αν λοιπόν συναντήσουμε αυτό το πρόβλημα, το μόνο που έχουμε να κάνουμε είναι να **αφαιρέσουμε το test-bench από τα αρχεία του project** στο Quartus, χωρίς αυτό να μας δημιουργήσει κανένα πρόβλημα στην ροή των εργασιών μας (θυμηθείτε το σε επόμενη άσκηση που θα σας χρειαστεί).

Στις επόμενες εργαστηριακές ασκήσεις θα κληθούμε να δημιουργήσουμε test bench μόνοι μας. Ωστόσο, μπορούμε να επιταχύνουμε την δημιουργία του αν εκτελέσουμε όλη την παραπάνω ακολουθία βημάτων χωρίς να εισάγουμε όλα τα σήματα, εκτός από ένα (όποιο επιθυμούμε) σε μία οποιαδήποτε σταθερή τιμή ώστε να παραχθεί το αρχείο. Εμείς μπορούμε κατόπιν να το επεξεργαστούμε και να εισάγουμε μία process για κάθε μία είσοδο που θέλουμε να διαμορφώσουμε, και να επεκτείνουμε το test bench με ειδικές process που θα ελέγχουν τα αποτελέσματα της εξομοίωσης αυτόματα (δοκιμάστε να επαναλάβετε την διαδικασία χωρίς να καθορίσετε όλα τα σήματα εισόδου και μελετήστε το test bench που δημιουργήσατε).


```

LIBRARY ieee ;
LIBRARY std ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;
USE ieee.std_logic_textio.all ;
USE ieee.STD_LOGIC_UNSIGNED.all ;
USE ieee.std_logic_unsigned.all ;
USE std.textio.all ;
ENTITY Adder8_tb IS
END ;

ARCHITECTURE Adder8_tb_arch OF Adder8_tb IS
    SIGNAL A : std_logic_vector (7 downto 0) ;
    SIGNAL cin : STD_LOGIC ;
    SIGNAL B : std_logic_vector (7 downto 0) ;
    SIGNAL cout : STD_LOGIC ;
    SIGNAL S : std_logic_vector (7 downto 0) ;
    COMPONENT Adder8
    PORT (
        A : in std_logic_vector (7 downto 0) ;
        cin : in STD_LOGIC ;
        B : in std_logic_vector (7 downto 0) ;
        cout : out STD_LOGIC ;
        S : out std_logic_vector (7 downto 0) ;
    ) ;
END COMPONENT ;

BEGIN
    DUT : Adder8
    PORT MAP (
        A => A ,
        cin => cin ,
        B => B ,
        cout => cout ,
        S => S ) ;

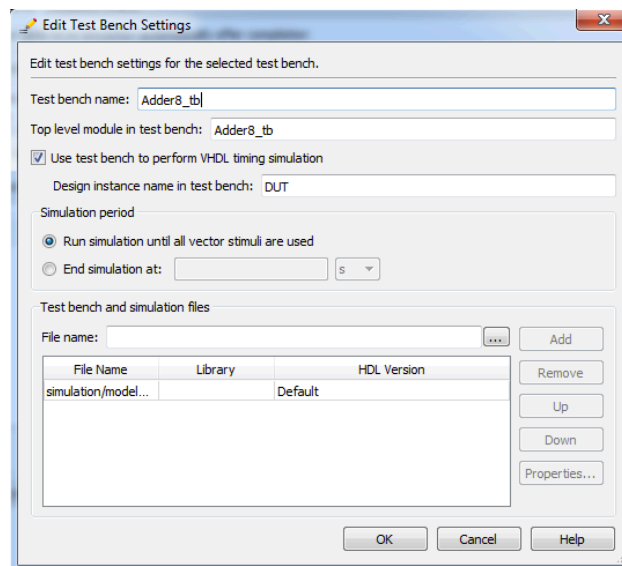
    -- "Constant Pattern"
    -- Start Time = 60 ns, End Time = 80 ns, Period = 0 ns
    Process
    Begin
        b <= "00000101" ; wait for 20 ns ;
        b <= "00010100" ; wait for 20 ns ;
        b <= "00101011" ; wait for 20 ns ;
        b <= "10100000" ; wait for 20 ns ;
        -- dumped values till 80 ns
        wait ;
    End Process ;

    -- "Counter Pattern" (Range-Up) : step = 1 Range(0-1)
    -- Start Time = 0 ns, End Time = 80 ns, Period = 20 ns
    Process
    Variable VARcin : std_logic_vector(0 downto 0) ;
    Begin
        for Z in 1 to 2
        loop
            VARcin := "0" ;
            for repeatLength in 1 to 2
            loop
                cin <= VARcin (0) ;
                wait for 20 ns ;
                VARcin := VARcin + 1 ;
            end loop ;
        end loop ;
        -- 80 ns, repeat pattern in loop.
    end loop ;
    wait ;
End Process ;

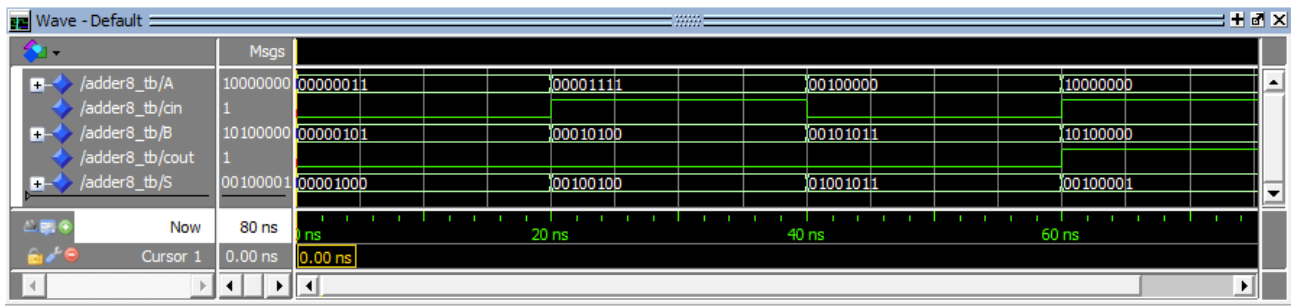
```

Εικόνα 38 Test Bench

Το επόμενο βήμα είναι η εκτέλεση εξομοίωσης με χρήση του test bench. Εκτελούμε και πάλι *Assignment>Settings>EDA Tool Settings>Simulation* και στο *NativeLink settings* επιλέγουμε *Compile test bench* και από την επιλογή *Test Benches...* επιλέγουμε το test bench όπως φαίνεται στην Εικόνα 39. Στο πεδίο *Test bench name* βάζουμε το όνομα της κορυφαίας οντότητας του test bench η οποία είναι *Adder8_tb* στην δική μας περίπτωση. Επιλέγουμε το *"Use test bench to perform VHDL timing simulation"* και στο πεδίο *Design instance name in test bench* βάζουμε το όνομα που έχει δοθεί στο test bench για την σχεδιαστική μας οντότητα (στην Εικόνα 38 Test Bench Εικόνα 38 το όνομα αυτό είναι η ετικέτα *DUT* που βρίσκεται αριστερά από το *Adder8* στο instantiation του module – προσέξτε το δικό σας test bench γιατί μπορεί να είναι διαφορετικό). Στο πεδίο *File Name* αναζητούμε το αρχείο με το test bench πατώντας τις τρεις τελείες, και αφού το εντοπίσουμε το επιλέγουμε και το προσθέτουμε με το πλήκτρο *Add*. Τέλος πατάμε *OK*, στο επόμενο παράθυρο πάλι πατάμε *OK* και τέλος *Apply* και πάλι *OK*. Τώρα είμαστε έτοιμοι να εκτελέσουμε την εξομοίωση επιλέγοντας *Tools>Run Simulation Tool>RTL Simulation* (εναλλακτικά πιέζουμε το αντίστοιχο πλήκτρο) οπότε και πάλι ανοίγει το *ModelSim* και μας επιστρέφει το αποτέλεσμα της εξομοίωσης το οποίο φαίνεται στην Εικόνα 40.

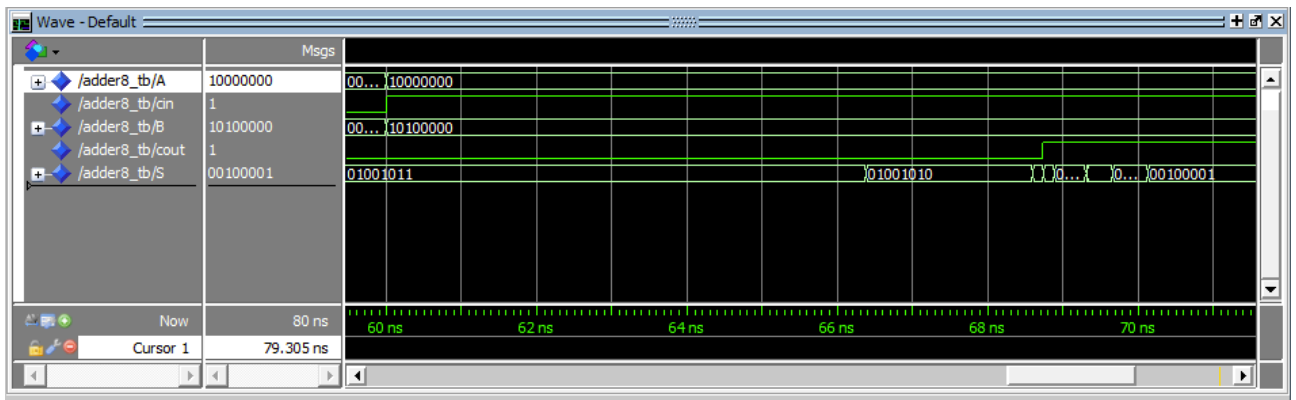


Εικόνα 39



Εικόνα 40

Με τον ίδιο τρόπο μπορούμε να τρέξουμε εξομοίωση χρονική χρησιμοποιώντας το κύκλωμα που έχει συντεθεί και όχι την HDL περιγραφή. Αφού κλείσουμε το ModelSim εκτελούμε *Tools>Run Simulation Tool>Gate Level Simulation* και κατόπιν μπορούμε να επιλέξουμε είτε το Slow Model είτε το Fast Model. Το αποτέλεσμα της εξομοίωσης για το ίδιο test bench την χρονική περίοδο 60-70ns φαίνεται στην Εικόνα 41 όπου είναι σαφής η καθυστέρηση μεταβολής της εξόδου μετά την αλλαγή των εισόδων στα 60 ns.



Εικόνα 41. Χρονική εξομοίωση

Σημαντική παρατήρηση: η εξομοίωση σε RTL μπορεί να εκτελεστεί μόνο όταν η σχεδίαση μας είναι σε μορφή VHDL, Verilog, AHDL, και υποχρεωτικά μόνο σε μία από αυτές. Αν εισάγουμε άλλου τύπου κυκλώματα (πχ πύλες βιβιοθήκης, μονάδες που έχουμε σχεδιάσει σε gate-level, μικτές σχεδιάσεις σε διάφορες γλώσσες περιγραφής, αρχεία bdf κλπ), θα πρέπει πρώτα να μετατρέψουμε σε VHDL αρχείο την κορυφαία οντότητα. Προσέξτε ότι αυτή η διαδικασία δεν είναι απαραίτητη σε μονάδες που δεν μπορούμε να εξομοιώσουμε όπως το LCD ή τα HEX DISPLAYS που είναι σε VERILOG. Για τον λόγο αυτό συνιστούμε να εξομοιώνετε με RTL εξομοίωση (αν είναι απαραίτητο) μόνο τα τμήματα του σχεδιασμού που έχουν περιγραφεί VHDL, και σε όλες τις υπόλοιπες περιπτώσεις να εκτελείτε **gate-level εξομοίωση**.

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:

```

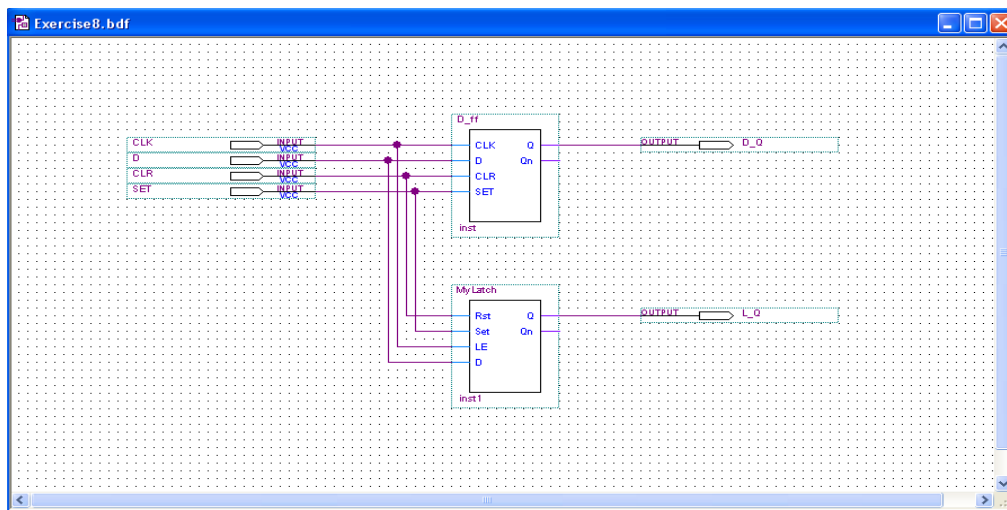
library IEEE;
use IEEE.std_logic_1164.all;

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

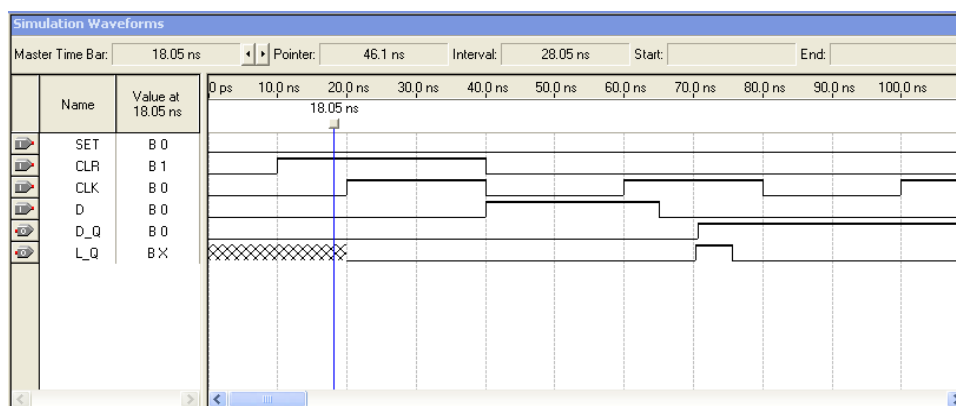
architecture RTL of MyLatch is
    signal FF: std_logic;
begin
    seq0 : process (Rst, Set, D, LE)
    begin
        if Rst = '1' then FF <= '0';
        elsif Set = '1' then FF <= '1';
        elsif LE = '1' then FF <= D;
        end if;
    end process;
    Q <= FF;
    Qn <= not FF;
end RTL;

```

Κατόπιν σχεδιάστε το σχηματικό που φαίνεται στην Εικόνα 42, 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 χρονική εξομοίωση που φαίνεται στην Εικόνα 43.



Εικόνα 42



Εικόνα 43

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). Αποθηκεύστε τον καταχωρητή στο αρχείο με όνομα Reg8.vhd και με χρονική εξομοίωση δείξτε ότι ο καταχωρητής που σχεδιάσατε λειτουργεί σωστά σε όλες τις λειτουργίες του. Παραθέστε τις κυματομορφές στην αναφορά σας. Δώστε τα αποτελέσματα της στατικής χρονικής ανάλυσης και υπολογίστε την μέγιστη συχνότητα ρολογιού που μπορεί να χρησιμοποιηθεί 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 χρονική εξομοίωση χρησιμοποιώντας test-bench και αποδείξτε την σωστή λειτουργία του μετρητή. Με στατική χρονική ανάλυση βρείτε την μέγιστη ταχύτητα λειτουργίας του και αναφέρετε την στην αναφορά σας.

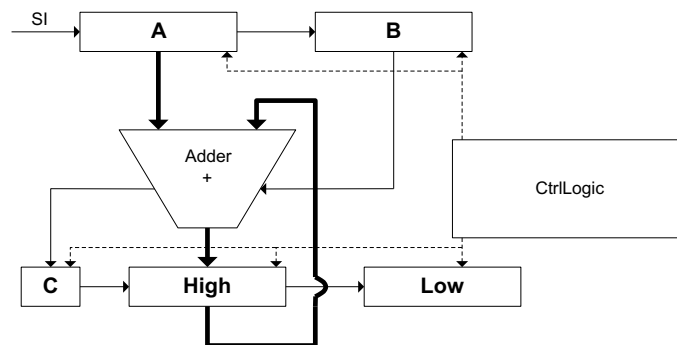
Παραδοτέα 5^{ης} Εργαστηριακής Άσκησης

1. Παρουσιάστε τα αποτελέσματα των εξομοιώσεων στους επιτηρητές σας.
2. Προγραμματίστε στο FPGA 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). Προβάλετε το περιεχόμενο του καταχωρητή σε δύο Hex Displays και δείξτε την σωστή λειτουργία του κυκλώματος που σχεδιάσατε στους επιτηρητές.
3. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

Εργαστηριακή Άσκηση 6: 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). Επιπλέον θα δημιουργήσουμε το δικό μας test-bench και θα μάθουμε πως να χρησιμοποιήσουμε μερικές από τις δυνατότητες εκ-σφαλμάτωσης της σχεδίασης που παρέχει το ModelSim.

Το γενικό σχηματικό κύκλωμα του πολλαπλασιαστή φαίνεται στο ακόλουθο σχήμα:



Εικόνα 44. 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 φυσικά η 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 η εξής:

1. Στους $2n$ πρώτους κύκλους φορτώνονται σειριακά 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 ολισθητή μήκους $2n+1$ που δημιουργείται από τους καταχωρητές C \rightarrow High \rightarrow Low.

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

Σχεδίαση Βασικής Οντότητας Καταχωρητή

Η πρώτη οντότητα που θα σχεδιάσουμε είναι ο καταχωρητής. Αν και χρειαζόμαστε 5 διαφορετικούς καταχωρητές θα περιγράψουμε μόνο μία οντότητα καταχωρητή εμπλουτίζοντας την 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.

```

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;

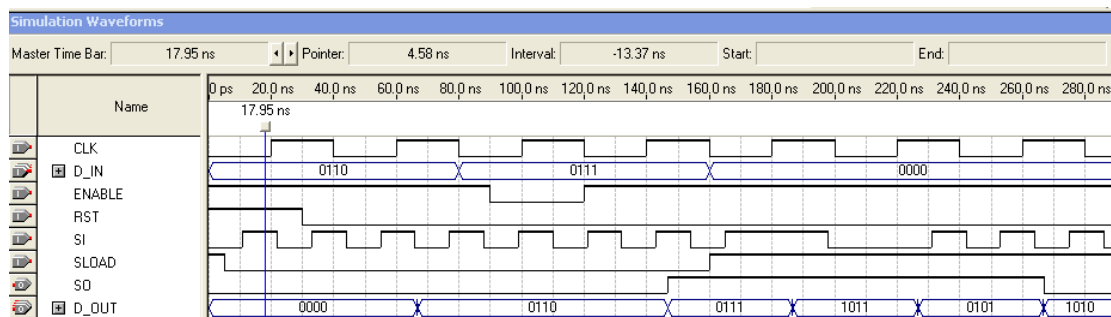
```

Εικόνα 45. Οντότητα Καταχωρητή

Στην Εικόνα 45 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 46

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 γίνεται 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 η πρόσθεση έχει μία ιδιομορφία: από τον αλγόριθμο του πολλαπλασιασμού γίνεται πρόσθεση $A + \text{High} \Rightarrow (C, \text{High})$ όταν το λιγότερο σημαντικό ψηφίο του καταχωρητή B είναι ίσο με '1' ($B[0] = '1'$) ή εναλλακτικά πρόσθεση $0 + \text{High} \Rightarrow (C, \text{High})$ όταν $B[0] = 0$. Την περίπτωση αυτή μπορούμε να την χειριστούμε εσωτερικά στον αθροιστή, ή να την χειριστούμε εξωτερικά προκειμένου να χρησιμοποιήσουμε μία γενική περιγραφή αθροιστή η οποία θα μας φανεί πολύ χρήσιμη και σε άλλα κυκλώματα που θα φτιάξουμε αργότερα. Στην συγκεκριμένη άσκηση θα ακολουθήσουμε την δεύτερη περιγραφή η οποία είναι πιο ορθή, και η οποία φαίνεται στην Εικόνα 47 (την επιλογή ανάμεσα στην τιμή του καταχωρητή A και το 0 την αναβάλλουμε για την μονάδα του πολλαπλασιαστή). Εκτελέστε χρονική εξομοίωση για να δείξετε όλες τις δυνατότητες του αθροιστή.

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

entity Adder is
    generic ( n: integer := 4 );
    port (   A, B: in std_logic_vector (n-1 downto 0);
            Sum: out std_logic_vector (n-1 downto 0);
            Cout: out std_logic );
end Adder;

architecture DataFlow of Adder is
    signal F: std_logic_vector (n downto 0);
begin
    F <= ('0' & A) + ('0' & B);
    Cout <= F(n);
    Sum <= F(n-1 downto 0);
end DataFlow;
```

Εικόνα 47. Αθροιστής

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 αναλοίωτα. 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 από την σειριακή είσοδο SI. 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 $C \rightarrow \text{High} \rightarrow \text{Low}$ (δείτε Εικόνα 44). 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 44 και Εικόνα 45). 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. Ας δούμε ακόλουθα κάποιες δηλώσεις που απαιτούνται στην περιγραφή της μονάδας ελέγχου:

```
type state_type is (LOAD, ADD, SHIFT, FINISH);
signal state : state_type;
signal count : std_logic_vector (n 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 ADD είναι η κατάσταση πρόσθεσης, ενώ η κατάσταση SHIFT είναι η κατάσταση ολίσθησης. Τέλος η κατάσταση FINISH 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. Όπως έχουμε ήδη αναφέρει, χρειάζονται εναλλάξ n κύκλοι ρολογιού ADD και n κύκλοι ρολογιού SHIFT για να ολοκληρωθεί.

Αρχικά παραθέτουμε τον κώδικα του μετρητή με το όνομα count ο οποίος ελέγχει κάθε φορά τους απαραίτητους κύκλους που πρέπει να εκτελεστούν. Με το σήμα Rst μηδενίζεται το περιεχόμενο του μετρητή ασύγχρονα (ο μετρητής μηδενίζεται παραμετρικά έτσι ώστε η μονάδα ελέγχου να μπορεί να χρησιμοποιηθεί για οποιοδήποτε μήκος πολλαπλασιασμού), ενώ ακόλουθα μετράει συνεχώς κατά 1. Οι τιμές που μας ενδιαφέρουν είναι α) η τιμή '2n' οπότε και έχει τελειώσει η σειριακή φόρτωση, και β) η τιμή '4n' οπότε και έχει τελειώσει η διαδικασία προσθέσεων-ολισθήσεων. Προσέξτε ότι το μέγεθος του καταχωρητή count είναι τουλάχιστον 'n+1' δυαδικά ψηφία ώστε να μπορεί να φθάσει στις τιμές μέτρησης που μας ενδιαφέρουν. Επίσης προσέξτε τον τρόπο με τον οποίο γίνεται η αύξηση της τιμής του μετρητή.

```
p0: process(Rst, CLK)
begin
```

```

    if (Rst='1') then
        count <= (n downto 0 => '0');
    elsif (CLK'event and CLK='1') then
        count <= count + '1';
    end if;
end process;

```

Ακόλουθα παραθέτουμε τον κώδικα σε VHDL της μηχανής κατάστασης της μονάδος ελέγχου. 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 (προσέξτε ότι ταυτόχρονα μηδενίζει τον μετρητή στην προηγούμενη process).

```

p1: process(Rst, CLK)
begin
    if (Rst='1') then state <= LOAD;
    elsif (CLK'event and CLK='1') then
        case state is
            when LOAD => if (conv_integer(count) = 2*n-1) then state <= ADD; end if;
            when ADD => state <= SHIFT;
            when SHIFT => if (conv_integer(count) = 4*n-1) then state <= FINISH; else state <= ADD; end if;
            when FINISH => null;
        end case;
    end if;
end process;

```

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 μονάδος ελέγχου. Στην κατάσταση LOAD η μονάδα παραμένει για $2n$ κύκλους. Έτσι ελέγχει την τιμή του μετρητή count μέχρι να φθάσει στην τιμή $2n-1$ οπότε μεταβαίνει στην κατάσταση ADD. Κατόπιν εναλλάσσεται η λειτουργία για τους επόμενους κύκλους ανάμεσα σε SHIFT και ADD έως ότου φθάσει ο μετρητής την τιμή $4n-1$ οπότε περνάει στην κατάσταση FINISH καθώς έχει ολοκληρωθεί ο πολλαπλασιασμός. Στην κατάσταση FINISH δεν εκτελείται καμία ενέργεια περιμένοντας να ενεργοποιηθεί το σήμα αρχικοποίησης.

Έχοντας πλέον καθορίσει την ακολουθία καταστάσεων απομένει να καθορίσουμε τις τιμές των σημάτων ελέγχου για την λειτουργία των καταχωρητών σε κάθε κατάσταση σε μία περιγραφή ροής δεδομένων. Επειδή η λειτουργία του κάθε καταχωρητή διαφοροποιείται ανάλογα με την κατάσταση της μονάδος ελέγχου, απαιτείται σαφής προσδιορισμός των καταστάσεων. Ας δούμε αναλυτικότερα τα σήματα αυτά για κάθε καταχωρητή χωριστά:

Καταχωρητής A:

Ο καταχωρητής A λειτουργεί μόνο κατά την διάρκεια της σειριακής φόρτωσης LOAD και τον υπόλοιπο χρόνο διατηρεί τα δεδομένα του αναλοίωτα. Κατά την διάρκεια της σειριακής φόρτωσης το σήμα SL_A πρέπει να είναι στην τιμή '1' (δείτε λειτουργία καταχωρητή). Έτσι προκύπτουν τα ακόλουθα σήματα (προσέξτε ότι στο σήμα SL_A ενεργοποιείται μαζί με το EN_A αφού ο καταχωρητής εκτελεί μόνο ολίσθηση)

```

EN_A <= '1' when state=LOAD else '0';
SL_A <= '1' when state=LOAD else '0';

```

Καταχωρητής B:

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. Όμοια με τον καταχωρητή A και ο καταχωρητής B ολισθαίνει πάντα οπότε το σήμα SL_B ενεργοποιείται μαζί με το EN_B.

```
EN_B <= '1' when (state=LOAD or state= SHIFT) else '0';  
SL_B <= '1' when (state=LOAD or state= SHIFT) else '0';
```

Καταχωρητής High (H):

Ο καταχωρητής H λειτουργεί μόνο κατά την διάρκεια του πολλαπλασιασμού είτε για να αποθηκεύσει το αποτέλεσμα της πρόσθεσης, είτε για να ολισθήσει δεξιά. Έτσι το σήμα EN_H είναι ενεργό και στις δύο καταστάσεις ADD, SHIFT, ενώ το σήμα SL_H που καθορίζει εάν ο καταχωρητής θα κάνει ολίσθηση ή παράλληλη φόρτωση είναι ενεργό (SL_H=1) μόνο στην κατάσταση SHIFT (SL_H=0 στην κατάσταση ADD). Ο κώδικας είναι ο παρακάτω

```
EN_H <= '1' when (state = ADD or state = SHIFT) else '0';  
SL_H <= '1' when state = SHIFT else '0';
```

Καταχωρητής Low (L):

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

```
EN_L <= '1' when (state = SHIFT) else '0';  
SL_L <= '1' when (state = SHIFT) else '0';
```

Καταχωρητής C:

Πρόκειται για ένα απλό flip flop το οποίο αποθηκεύει το κρατούμενο της πρόσθεσης. Λειτουργεί μόνο στον κύκλο ADD. Πάντα φορτώνει το δεδομένο από την παράλληλη θύρα οπότε δεν κάνει ποτέ ολίσθηση.

```
EN_C <= '1' when (state = ADD) else '0';  
SL_C <= '0';
```

Εισάγετε την παραπάνω περιγραφή και εκτελέστε μεταγλώττιση-σύνθεση.

Εξομοίωση Μονάδας Ελέγχου

Η εξομοίωση της μονάδας ελέγχου απαιτεί ιδιαίτερη προσοχή καθώς είναι σημαντικά δυσκολότερη από την εξομοίωση των υπόλοιπων μονάδων. Αυτό οφείλεται στην δυσκολία παρακολούθησης σημάτων ελέγχου καθώς δεν είναι πάντα προφανής η χρήση και λειτουργία τους (για παράδειγμα τέτοια σήματα 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.

Αρχικά θέστε την μονάδα ελέγχου ως την κορυφαία μονάδα σχεδίασης και εκτελέστε μεταγλώττιση. Διορθώστε λάθη εάν υπάρχουν. Κατόπιν με την διαδικασία που περιγράψαμε σε προηγούμενη άσκηση ξεκινήστε λογική (functional) εξομοίωση στην CtrlLogic και δημιουργήστε ένα σήμα ρολογιού με περίοδο 20 ns και Duty Cycle 50% το οποίο θα αναθέσετε στο σήμα CLK στο ModelSim. Τέλος δημιουργήστε αυτόματα ένα test-bench με το όνομα CtrlLogic_tb (προσοχή, χρειάζεται να δημιουργήσετε τουλάχιστον ένα σήμα για να φτιαχτεί αυτόματα το test-bench). Προσθέστε το test-bench στο project και ανοίξτε το. Θα παρατηρήσετε ότι έχει ήδη τοποθετηθεί η CtrlLogic όπως επίσης και μία process που καθορίζει το σήμα ρολογιού. Σε αυτό το test-bench θα προσθέσουμε μία ακόμη ανάθεση με την οποία θα ρυθμίσουμε το σήμα Rst στην τιμή 1 για τα πρώτα 20 ns και στο 0 μετά. Αυτό μπορεί να γίνει με μία νέα process, ή και με μία απλή ανάθεση ροής που τοποθετείται στην αρχιτεκτονική του CtrlLogic_tb μετά από την process του ρολογιού:

```
Rst <= '1', '0' after 20 ns;
```

Τώρα μπορούμε να εκτελέσουμε εξομοίωση με τον τρόπο που περιγράψαμε στην προηγούμενη άσκηση και να παρατηρήσουμε τα σήματα ελέγχου. Θα δούμε ότι, πράγματι, είναι εξαιρετικά δύσκολο να κατανοήσουμε αν είναι σωστή η περιγραφή μας. Ωστόσο, αν μπορούσαμε να παρακολουθήσουμε μερικά ακόμη σήματα εσωτερικά της μονάδος ελέγχου αυτή η διαδικασία θα γινόταν πολύ ευκολότερη. Ένα από αυτά τα σήματα το οποίο θα διευκόλυνε τον έλεγχο μας είναι το σήμα count της μονάδος ελέγχου. Για να μπορέσουμε να το παρακολουθήσουμε θα πρέπει να δηλώσουμε ένα επιπλέον σήμα ίδιου τύπου στην αρχιτεκτονική του test-bench, το οποίο είναι

signal monitor_count : **std_logic_vector** (n downto 0);

και κατόπιν θα πρέπει να προσθέσουμε και μία ανάθεση σε αυτό το σήμα όπως φαίνεται ακόλουθα

monitor_count <= << **signal** DUT.count : **std_logic_vector** (n downto 0) >>;

Παρατηρήστε σε αυτή την ανάθεση ότι χρησιμοποιείται η ιεραρχική προσπέλαση του σήματος μέσα από την μονάδα ελέγχου DUT.count (DUT είναι το όνομα που δώσαμε κατά την δημιουργία του στιγμιότυπου της μονάδας ελέγχου). Με τον ίδιο τρόπο μπορούμε να προσπελάσουμε οποιοδήποτε σήμα θέλουμε ακόμη και πιο βαθιά εμφωλευμένο. Ωστόσο, πρέπει να γνωρίζουμε ότι η παρακολούθηση εσωτερικών σημάτων είναι δυνατή μόνο στην λειτουργική και όχι στην χρονική εξομοίωση. Ο λόγος είναι ότι η χρονική εξομοίωση προκύπτει από την σύνθεση της περιγραφής και τα ονόματα σημάτων παύουν να ισχύουν και αντικαθίστανται από πραγματικά σήματα όπως προκύπτουν από το υλικό που δημιουργείται (σε πολλές περιπτώσεις δεν υπάρχουν ούτε καν ως σήματα μετά την βελτιστοποίηση). Αν λοιπόν επιθυμούμε να παρακολουθήσουμε ένα εσωτερικό σήμα σε χρονική εξομοίωση θα πρέπει προσωρινά να το συνδέσουμε σε θύρα εξόδου. Ωστόσο, η λειτουργική εξομοίωση είναι επαρκής για να διορθώσουμε όλα τα λάθη, και αν τηρήσουμε και τους χρονικούς περιορισμούς που προκύπτουν από την στατική ανάλυση δεν θα χρειαστεί να μπούμε σε αυτή την διαδικασία.

Αφού λοιπόν εισάγετε την παραπάνω περιγραφή μπορείτε να εκτελέσετε και πάλι εξομοίωση. Μπορείτε να κλείσετε το ModelSim και να επαναλάβετε την εκτέλεση της εξομοίωσης σε RTL από το περιβάλλον του Quartus, ή απλά να πληκτρολογήσετε την εντολή εκτέλεσης εξομοίωσης στο παράθυρο transcript του ModelSim. Το αρχείο που περιλαμβάνει τα βασικά βήματα εξομοίωσης βρίσκεται στον κατάλογο του project simulation/ModelSim και έχει την κατάληξη .do (...rtl_vhdl.do για εξομοίωση περιγραφής και ...gate_vhdl.do για χρονική εξομοίωση). Στην δική μας περίπτωση πρέπει να εκτελέσουμε στο παράθυρο transcript την εντολή

do Multiplier_run_msim_rtl_vhdl.do

ή απλά να πιέσουμε (όσο βρισκόμαστε στο transcript) το πάνω βελάκι οπότε ανακαλείται η τελευταία εντολή που εκτελέστηκε που συνήθως είναι η εντολή που ζητάμε (προσοχή, αν έχετε δώσει άλλο όνομα στο project θα πρέπει να βρείτε το σωστό όνομα του αρχείου).

Τα αποτελέσματα της εξομοίωσης δείχνουν ότι πράγματι η τιμή του μετρητή αυξάνει με κάθε κύκλο ρολογιού όπως θα έπρεπε. Ωστόσο, η πληροφορία αυτή δεν επαρκεί για να διαπιστώσουμε την ορθότητα της περιγραφής μας. Θα ήταν πολύ χρήσιμο να μπορούμε να δούμε και την κατάσταση που βρίσκεται η μονάδα ελέγχου, δηλαδή το σήμα state της μονάδος DUT. Εδώ όμως έχουμε ένα πρόβλημα: δεν μπορούμε να ορίσουμε ένα σήμα που θα μας αποδίδει την τιμή της κατάστασης καθώς δεν έχουμε ορίσει τον τύπο του state_type ο οποίος έχει οριστεί στο αρχείο CtrlLogic.vhd. Εδώ λοιπόν μπορούμε πλέον να αντιληφθούμε την αξία των πακέτων. Αντί να ορίσουμε τον συγκεκριμένο τύπο εσωτερικά στην μονάδα ελέγχου, θα δημιουργήσουμε ένα νέο πακέτο στο οποίο θα την τοποθετήσουμε και θα μπορούμε να την κάνουμε ορατή και στο test-bench. Δημιουργούμε λοιπόν ένα αρχείο Declare.vhd το οποίο προσθέτουμε στο project με την παρακάτω περιγραφή

library ieee;

use ieee.std_logic_1164.all;

package Declarations **is**

type state_type **is** (LOAD, ADD, SHIFT, FINISH);

end Declarations;

Επιπλέον καλούμε το πακέτο μέσα από την περιγραφή της μονάδος ελέγχου (CtrlLogic.vhd) και από το test-bench ως εξής:

use work.Declarations.all;

ενώ πλέον σβήνουμε την αντίστοιχη δήλωση από την μονάδα ελέγχου. Τέλος, όπως και πριν δηλώνουμε ένα σήμα για να παρακολουθούμε την κατάσταση της μονάδος ελέγχου, και προσθέτουμε την ανάθεση του με τις παρακάτω εντολές (τοποθετήστε τις στα σωστά σημεία του test-bench):

signal monitor_state : state_type;

monitor_state <= << **signal** DUT.state : state_type >>;

Τώρα πρέπει να επαναλάβετε την σύνθεση καθώς οι περιγραφές άλλαξαν (αν χρειαστεί αφαιρέστε το test-bench από τα αρχεία του project όπως περιγράψαμε στην προηγούμενη άσκηση) και αφού κλείσετε το ModelSim εκτελέστε και πάλι εξομοίωση RTL (αν δεν κλείσετε το ModelSim δεν θα μπορέσετε να το ανοίξετε δεύτερη φορά). Παρατηρήστε τα αποτελέσματα για τα δύο σήματα που προσθέσατε. Επιβεβαιώστε ότι τα σήματα εξόδου της μονάδος ελέγχου είναι σωστά.

Σχεδίαση Πολλαπλασιαστή

Έχοντας σχεδιάσει τις βασικές δομικές μονάδες, απομένει πλέον να σχεδιάσουμε την κορυφαία οντότητα του 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 (    A, B      : in      std_logic_vector (n-1 downto 0);
            SUM       : 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;

Μαζί με τα παραπάνω component θα πρέπει να δηλωθούν και τα σήματα που θα χρησιμοποιηθούν για την διασύνδεση των μονάδων. Οι αντίστοιχες δηλώσεις είναι οι παρακάτω (προσέξτε την δήλωση των σημάτων του ff που διατηρεί το κρατούμενο – η χρήση της οντότητας Reg και για αυτό μας υποχρεώνει να δηλώσουμε τα σήματα ως διανύσματα μοναδιαίου μήκους).

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

Το instantiation των καταχωρητών του πολλαπλασιαστή γίνεται όπως φαίνεται ακόλουθα. Προσέξτε ότι η θύρα εξόδου Low συνδέεται απευθείας στον καταχωρητή R_L καθώς δεν απαιτείται να διαβαστεί, σε αντίθεση με την θύρα εξόδου High η οποία θα πρέπει να οδηγηθεί από την H που είναι η έξοδος του καταχωρητή R_H καθώς πρέπει να οδηγηθεί και στον αθροιστή (συνεπώς πρέπει να διαβαστεί).

```
R_A:  Reg    generic map (n=>n)  
      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);  
  
R_B:  Reg    generic map (n=>n)  
      port map ( D_IN => (n-1 downto 0 => '0'), SI => SO_A, CLK => CLK, RST => RST,  
                SLOAD => SL_B, ENABLE => EN_B, SO => open, D_OUT => B);  
  
R_C:  Reg    generic map (n=>1)  
      port map ( D_IN => COUT, SI => '0', CLK => CLK, RST => RST, SLOAD => SL_C,  
                ENABLE => EN_C, SO => open, D_OUT => C);  
  
R_H:  Reg    generic map (n=>n)  
      port map ( D_IN => SUM, SI => C(0), CLK => CLK, RST => RST, SLOAD => SL_H,  
                ENABLE => EN_H, SO => SO_H, D_OUT => H);  
  
High <= H;  
  
R_L:  Reg    generic map (n=>n)  
      port map ( D_IN => (n-1 downto 0 => '0'), SI => SO_H, CLK => CLK, RST => RST,  
                SLOAD => SL_L, ENABLE => EN_L, SO => open, D_OUT => Low);
```

Το στιγμιότυπο του αθροιστή είναι το ακόλουθο (προσέξτε ότι στην θύρα B του αθροιστή μπαίνει το περιεχόμενο του καταχωρητή H μετά το λογικό ΚΑΙ με το λιγότερο σημαντικό δυαδικό ψηφίο του καταχωρητή B, το B[0]).

```
U_Ad: Adder  generic map (n=>4)  
      port map (A => H, B => ( (n-1 downto 0 => B(0)) and A), SUM => SUM, COUT =>  
                COUT(0));
```

Το στιγμιότυπο της μονάδας ελέγχου φαίνεται ακόλουθα:

```
U_Ctl: 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);
```

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 R_A, R_B. Τα σήματα SL_A, SL_B ελέγχουν την σειριακή φόρτωση τους. Ο καταχωρητής R_A δέχεται ως σειριακή είσοδο την σειριακή είσοδο του πολλαπλασιαστή SI ενώ ο καταχωρητής R_B δέχεται ως σειριακή είσοδο την σειριακή έξοδο του A, SO_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 R_H (High), R_L (Low) για το υψηλό και χαμηλό nibble του αποτελέσματος. Επίσης ο καταχωρητής R_C κρατάει το κρατούμενο της πρόσθεσης. Προσέξτε από την συνδεσμολογία των τριών αυτών καταχωρητών ότι η σειριακή έξοδος του R_C περνάει στην σειριακή είσοδο του R_H, η σειριακή έξοδος του R_H περνάει στην σειριακή είσοδο του R_L. 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 κρατούμενου, αλλά προτιμήσαμε να κρατήσουμε τον αριθμό των οντοτήτων ελάχιστο. Επίσης παρατηρήστε ότι οι σειριακές εξοδοι των καταχωρητών R_B, R_C, R_L αφήνονται ασύνδετοι (open) για να μην δηλώνουμε επιπλέον σήματα τα οποία δεν χρησιμοποιούνται.

Μια άλλη οντότητα που χρησιμοποιούμε είναι εκείνη του αθροιστή ο οποίος δέχεται ως εισόδους την έξοδο του R_H και του R_A (μετά την λογική πράξη ΚΑΙ με το B[0]) και προωθεί το αποτέλεσμα της πρόσθεσης στον καταχωρητή R_H, και το κρατούμενο στον καταχωρητή R_C. Τέλος η τιμή του καταχωρητή R_H ανατίθεται στην θύρα εξόδου High, όπως εξηγήσαμε παραπάνω.

Εισάγετε την παραπάνω περιγραφή στο κατάλληλο αρχείο VHDL το οποίο και θα ορίσετε ως κορυφαία οντότητα της σχεδίασης σας. Δημιουργήστε ένα test-bench και επιβεβαιώστε την ορθότητα της σχεδίασης εκτελώντας εξομοίωση με τον πολλαπλασιασμό $7 \times 5 = 35$. Χρησιμοποιήστε την περίοδο ρολογιού που σας επιτρέπει η στατική χρονική ανάλυση. Για να εισάγετε σειριακά τα δεδομένα (0101,0111) σε 8 διαδοχικούς κύκλους ρολογιού μπορείτε να χρησιμοποιήσετε την παρακάτω ανάθεση (από δεξιά προς τα αριστερά για περίοδο ρολογιού 20 ns)

SI <= '1' after 0 ns, '1' after 20 ns, '1' after 40 ns, '0' after 60 ns, '1' after 80 ns, '0' after 100 ns, '1' after 120 ns, '0' after 140 ns;

και λαμβάνοντας υπόψη ότι τα πρώτα 20 ns ενεργοποιούμε το σήμα Reset, μπορούμε να γράψουμε πιο συνοπτικά

SI <= '1', '0' after 80 ns, '1' after 100 ns, '0' after 120 ns, '1' after 140 ns, '0' after 160 ns;

Στο τελευταίο βήμα αυτής της άσκησης θα δούμε μία ακόμη δυνατότητα που μας παρέχει ένα test-bench, την δυνατότητα του αυτόματου ελέγχου. Συγκεκριμένα, θα ελέγξουμε το αποτέλεσμα του πολλαπλασιασμού όταν αυτό θα είναι διαθέσιμο. Για να συμβεί αυτό θα πρέπει να ισχύουν τα ακόλουθα:

1. Η κατάσταση της μονάδος ελέγχου να έχει αλλάξει σε FINISH
2. Να έχει παρέλθει ένας κύκλος ρολογιού για να έχουν αποθηκευτεί τα αποτελέσματα στους καταχωρητές (εναλλακτικά μπορούμε να αλλάξουμε την σχεδίαση της μονάδος ελέγχου ώστε το σήμα FINISH να τίθεται μετά την αποθήκευση).
3. Τα περιεχόμενα των καταχωρητών Low και High να είναι 0011 και 0010 αντίστοιχα για να είναι σωστό το αποτέλεσμα.

Προσθέστε λοιπόν στο test-bench ένα σήμα με όνομα monitor_state το οποίο θα παρακολουθεί το σήμα κατάστασης της μονάδος ελέγχου (όπως μάθαμε παραπάνω)


```
monitor_state <= << signal DUT.U_Ctl.state : state_type >>;
```

και κατόπιν προσθέστε την ακόλουθη process

```
process
begin
    wait on monitor_state;
    if (monitor_state = FINISH) then
        wait on clk;
        assert (FALSE) report "Checking..." severity note;
        assert (Low="0011" and High="0010") report "Check Failed" severity error;
    end if;
end process;
```

Προσέξτε ότι η process περιμένει αλλαγή στην κατάσταση του σήματος monitor_state και όταν αυτή συμβεί ελέγχει αν η τιμή του είναι FINISH. Αν η τιμή του είναι διαφορετική μπλοκάρει και πάλι αλλιώς περιμένει την επόμενη αλλαγή του ρολογιού οπότε και είναι σίγουρο πλέον ότι τα αποτελέσματα έχουν αποθηκευτεί στους καταχωρητές Low και High. Το πρώτο assert τυπώνει απλά το μήνυμα του ελέγχου, και το δεύτερο ελέγχει την ορθότητα της συνθήκης του αποτελέσματος. Αν δεν είναι αληθής τότε τυπώνει το μήνυμα "Check Failed" αλλιώς ολοκληρώνεται η εξομοίωση χωρίς κανένα επιπλέον μήνυμα. Τα μηνύματα εμφανίζονται στο παράθυρο transcript του ModelSim.

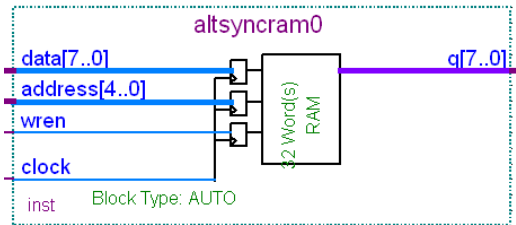
Εκτελέστε RTL εξομοίωση και επιβεβαιώστε την λειτουργία του πολλαπλασιαστή. Τροποποιήστε το test-bench ώστε να εκτελεί διαδοχικούς πολλαπλασιασμούς και ελέγξτε αυτόματα κάθε φορά το αναμενόμενο αποτέλεσμα. Κατόπιν βρείτε μια διαφορετική εκδοχή ώστε να μπορείτε να ελέγξετε τα αποτελέσματα και με χρονική εξομοίωση.

Παραδοτέα 6^{ης} Εργαστηριακής Άσκησης

1. Παρουσιάστε τα αποτελέσματα των εξομοιώσεων στους επιτηρητές σας.
2. Προγραμματίστε στο FPGA τον πολλαπλασιαστή στο board χρησιμοποιώντας τα πλήκτρα KEY0, KEY1 ως ασύγχρονο reset και ρολόι αντίστοιχα. Χρησιμοποιήστε το bit 1 από τα dip switches ως σειριακή είσοδο. Ελέγξτε την λειτουργία του πολλαπλασιαστή δίνοντας διαδοχικούς παλμούς από το push button που χρησιμοποιείται ως ρολόι. Προβάλετε το περιεχόμενο των εξόδων Low και High, και των καταχωρητών R_A, R_B, R_C στο LCD Display και δείξτε την σωστή λειτουργία του κυκλώματος που σχεδιάσατε στους επιτηρητές.
3. Τροποποιήστε την περιγραφή για να φορτώνονται οι καταχωρητές A, B παράλληλα σε ένα κύκλο ρολογιού από τα Switches μετά από την απενεργοποίηση του σήματος RST, και δημιουργήστε κατάλληλο σήμα που θα σηματοδοτεί την λήξη του πολλαπλασιασμού. Επεκτείνετε το μήκος των A, B σε 8 δυαδικά ψηφία και ελέγξτε με χρήση αυτόματου ρολογιού κοντινού στην συχνότητα λειτουργίας του πολλαπλασιαστή σας το αποτέλεσμα μερικών παραδειγμάτων.
4. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, τα αποτελέσματα των εξομοιώσεων που εκτελέσατε καθώς και ότι ζητήθηκε στα πλαίσια της άσκησης. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

Εργαστηριακή Άσκηση 7: Σχεδίαση με μνήμες

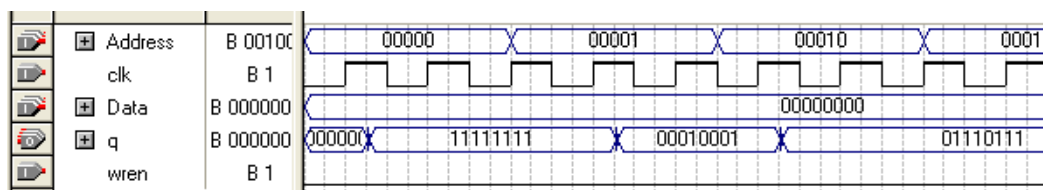
Στην 7^η εργαστηριακή άσκηση θα μάθουμε πώς μπορούμε να συνθέσουμε πιο σύνθετα κυκλώματα τα οποία ενσωματώνουν και μνήμη. Κατόπιν θα δούμε ένα παράδειγμα σχεδίασης με VHDL.



Εικόνα 48. RAM

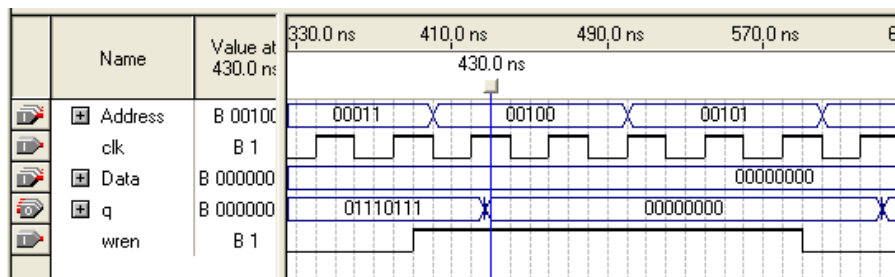
Ενσωμάτωση μνήμης

Μία από τις πλέον απαραίτητες μονάδες σε ένα περιβάλλον σχεδίασης είναι η 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 και με συγκεκριμένη οργάνωση. Το εργαλείο σχεδίασης Quartus δίνει πολλές δυνατότητες για σχεδίαση μνήμης. Ας δούμε όμως λίγο πιο προσεκτικά πως χρησιμοποιείται η μνήμη.



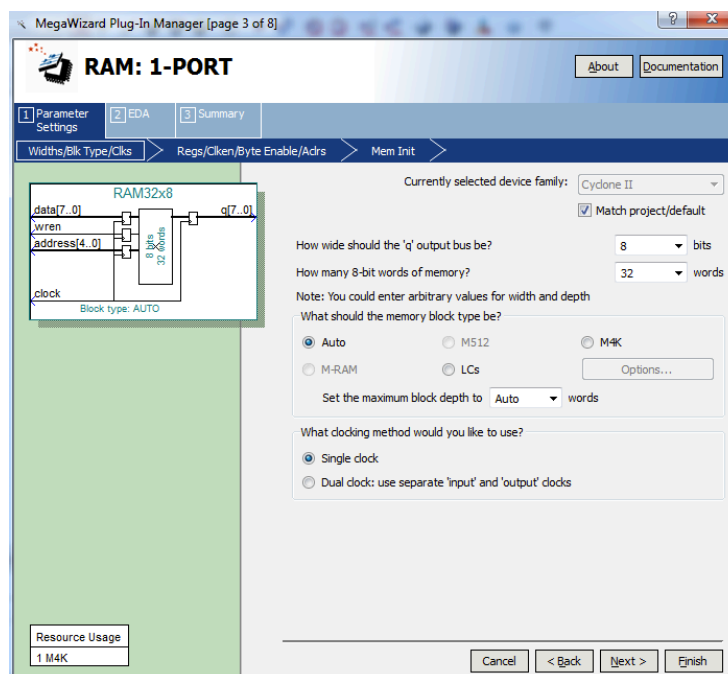
Εικόνα 49. Κύκλος ανάγνωσης

Στην Εικόνα 48 φαίνεται ένα από τα πιθανά interfaces 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* αποτελούνται από 8 δυαδικά ψηφία. 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 49 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).



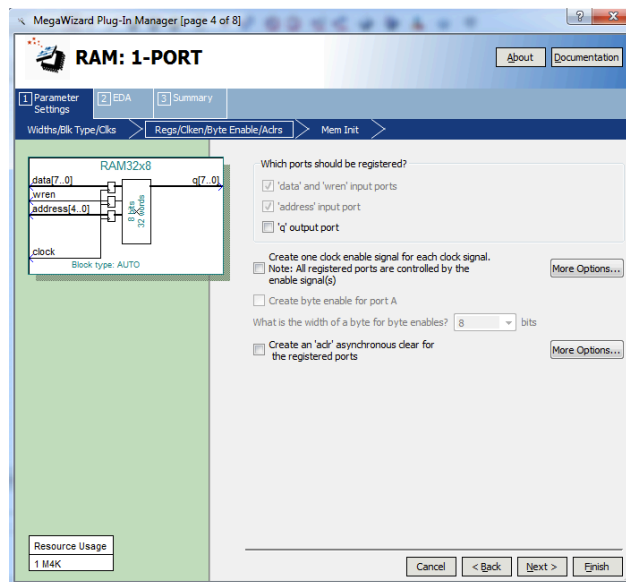
Εικόνα 50. Κύκλος εγγραφής

Για να εγγράψουμε δεδομένα σε μία θέση μνήμης θα πρέπει να ακολουθήσουμε μία ανάλογη τακτική. Αυτή την φορά θα πρέπει να τοποθετήσουμε στον δίαυλο διευθύνσεων Address την διεύθυνση και στον δίαυλο δεδομένων Data το περιεχόμενο προς εγγραφή πριν την θετική άνοδο του ρολογιού. Ταυτόχρονα ενεργοποιούμε και το σήμα wren ώστε να υποδείξουμε ότι πρόκειται για εγγραφή. Όπως παρατηρούμε και στην Εικόνα 50 η τιμή 00 γράφεται στην θέση μνήμης 4 και 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 51.

Ας δούμε τώρα πως μπορούμε να εισάγουμε μία τέτοια μνήμη στην σχεδίαση μας. Δημιουργούμε ένα νέο project (ΔΕΝ ΞΕΧΝΑΜΕ ΠΟΤΕ ΟΛΕΣ ΤΙΣ ΡΥΘΜΙΣΕΙΣ ΠΟΥ ΕΧΟΥΜΕ ΑΝΑΦΕΡΕΙ ΣΤΙΣ ΠΡΟΗΓΟΥΜΕΝΕΣ ΑΣΚΗΣΕΙΣ). Εισάγουμε μία *Megafunction* RAM (*Tools>MegaWizard Plug-In Manager*) και επιλέγουμε “Create a new custom megafunction variation” και κατόπιν *Memory Compiler/RAM: 1-PORT* ενώ συμπληρώνουμε στο πεδίο “Which type of output file do you want to create?” την επιλογή VHDL και στο όνομα του αρχείου RAM32x8. Στο πρώτο βήμα (Εικόνα 51) δηλώνουμε το μέγεθος της μνήμης (32 x 8 bits) καθώς και το μονό ρολόι στην μέθοδο χρονισμού.



Εικόνα 52.

Στο επόμενο βήμα δηλώνουμε ότι δεν θέλουμε καταχωρητή στην έξοδο (Εικόνα 52). Συγκεκριμένα, 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 και θα καθυστερούν έναν κύκλο ρολογιού παραπάνω. Καθώς δεν θέλουμε να συμβαίνει κάτι τέτοιο στην συγκεκριμένη περίπτωση, απενεργοποιούμε αυτή την δυνατότητα.

Στη συνέχεια καθορίζουμε εάν θέλουμε να χρησιμοποιήσουμε αρχείο αρχικοποίησης της μνήμης. Χωρίς να κλείσουμε τον wizard της μνήμης επιλέγουμε FILE->NEW->Memory Initialization File με παραμέτρους number of words 32, word size 8 και το αποθηκεύουμε με το όνομα RAM32x8.mif. Επανερχόμαστε στον wizard και αφού επιλέξουμε την αντίστοιχη ρύθμιση αναζητούμε και επιλέγουμε το αρχείο από την επιλογή Browse. Στα επόμενα παράθυρα απλά πατάμε Next και Finish και τέλος προσθέτουμε το αρχείο RAM32x8.qip στο project. Έπειτα ανοίγουμε τα αρχεία RAM32x8.mif και αρχικοποιούμε την θέση 00 με την τιμή FF, την θέση 01 με την τιμή 11 και τις υπόλοιπες με την τιμή 77. Εναλλακτικά μπορούμε να ανοίξουμε το αρχείο .mif με κάποιον εξωτερικό editor (Notepad++) να σβήσουμε τα περιεχόμενά του και να εισάγουμε τον ακόλουθο κώδικα:

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

Κατόπιν στον Project Navigator/Files επιλέγουμε RAM32x8.qip>RAM32x8.vhd και το κάνουμε Top Level Entity. Δημιουργούμε test-bench με την γνωστή μέθοδο και κατόπιν τρέχουμε εξομοίωση για να επιβεβαιώσουμε την λειτουργία της μνήμης. Στην εξομοίωση RTL μπορούμε να εκτελέσουμε μερικές αναγνώσεις και εγγραφές για να διαπιστώσουμε τον τρόπο λειτουργίας της μνήμης. Προσοχή: στις μνήμες μπορούμε να δούμε τα δεδομένα με τα οποία τις έχουμε αρχικοποιήσει μόνο μέσω της χρονικής εξομοίωσης. Έτσι προτιμούμε να εκτελούμε αμέσως χρονική εξομοίωση (και για την κατασκευή του test-bench) όταν έχουμε ενσωματωμένη μνήμη.

Στο επόμενο βήμα θα χρησιμοποιήσουμε την μνήμη που δημιουργήσαμε. Αν και μπορούμε να το κάνουμε απευθείας σε VHDL θα δούμε μία ακόμη εναλλακτική, που είναι η δημιουργία συμβόλου και η διασύνδεση σε σχηματικό. Δημιουργούμε λοιπόν ένα σύμβολο επιλέγοντας στον Project Navigator το

tab Files/RAM32x8.qip/RAM32x8.vhd και με δεξί πλήκτρο του ποντικιού *Create Symbol Files for Current File*.

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 32x8 χρησιμοποιώντας τον αλγόριθμο διάταξης φυσαλίδας (BubbleSort). Ως μνήμη RAM θα χρησιμοποιήσουμε την μνήμη που περιγράψαμε παραπάνω, 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 μπορέσουμε 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 διατάσσει σύμφωνα με τον παρακάτω αλγόριθμο:

```
do
    flag=0;
    for i = 0 to 31
        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 *delay=1* 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*, 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 επόμενη 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, τις *SendAddrA_w*, *WriteA*.

Ολοκληρώνοντας με την εναλλαγή ελέγχουμε εάν έχει ολοκληρωθεί η τρέχουσα επανάληψη, δηλαδή εάν ο μετρητής *count* έχει φθάσει στην μέγιστη τιμή του και σε τέτοια περίπτωση μεταβαίνουμε 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.

```
entity Sort is
port (
    clk, launch, reset      : in std_logic;
    DataIn                  : in std_logic_vector (7 downto 0);
    Address                  : out std_logic_vector (4 downto 0);
    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);
    type StateArray is array (state_type, bit) of state_type;
    constant NextState : StateArray := (
        Waiting          => ('0' => Waiting, '1' => SendAddrA_r),
        SendAddrA_r      => (others => GetA),
        GetA              => ('0' => GetA, '1' => SendAddrB_r),
        SendAddrB_r      => (others => GetB),
        GetB              => ('0' => GetB, '1' => Compare),
        Compare          => ('0' => CheckLoop, '1' => SendAddrB_w),
        SendAddrB_w      => (others => WriteB),
        WriteB            => (others => SendAddrA_w),
        SendAddrA_w      => (others => WriteA),
        WriteA            => (others => CheckLoop),
        CheckLoop         => ('0' => SendAddrA_r, '1' => CheckFlag),
        CheckFlag         => ('0' => Waiting, '1' => SendAddrA_r));
```

```

signal state : state_type;
signal dataA, dataB : std_logic_vector (7 downto 0);
signal count : std_logic_vector (4 downto 0);
signal Flag, Swap, CountEnd, condition, delay : std_logic ;
begin
  FSM: process (clk, reset)
  begin
    if (reset = '1') then state <= Waiting;
    elsif (clk'event and clk = '1') then
      state <= NextState(state, to_bit(condition));
    end if;
  end process;

  with state select
    condition <=  launch when Waiting, swap when Compare, CountEnd when CheckLoop, Flag when CheckFlag,
    delay when GetA | GetB, 'X' when others;

  Cnt: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (state = Waiting or (state = CheckFlag and Flag = '1')) then count <= "00000";
      elsif ((state = GetA and delay = '1') or state = WriteB) then count <= count + "00001";
      elsif (state = Compare and Swap = '1') then count <= count - "00001";
      end if;
    end if;
  end process;

  MemWait: process (clk, reset)
  begin
    if (reset = '1') then delay <= '0';
    elsif (clk'event and clk = '1') then
      if (state = GetA or state = GetB) then delay <= not delay; end if;
    end if;
  end process;

  Seq: process (clk)
  begin
    if (clk'event and clk = '1') then
      case (state) is
        when Waiting | CheckFlag      => Flag <= '0';
        when GetA                    => if (delay = '1') then dataA <= DataIn; end if;
        when GetB                    => if (delay = '1') then dataB <= DataIn; end if;
        when Compare                  => if (Swap = '1') then Flag <= '1'; end if;
        when others                    => null;
      end case;
    end if;
  end process;

  DataOut <= dataA when (state = SendAddrA_w or state = WriteA) else dataB;
  Address <= count;
  WR <= '1' when state=WriteB or state=WriteA else '0';
  Swap <= '1' when dataA>dataB else '0';
  CountEnd <= '1' when count = "11111" else '0';
  Complete <= '1' when state=Waiting else '0';

end RTL;

```

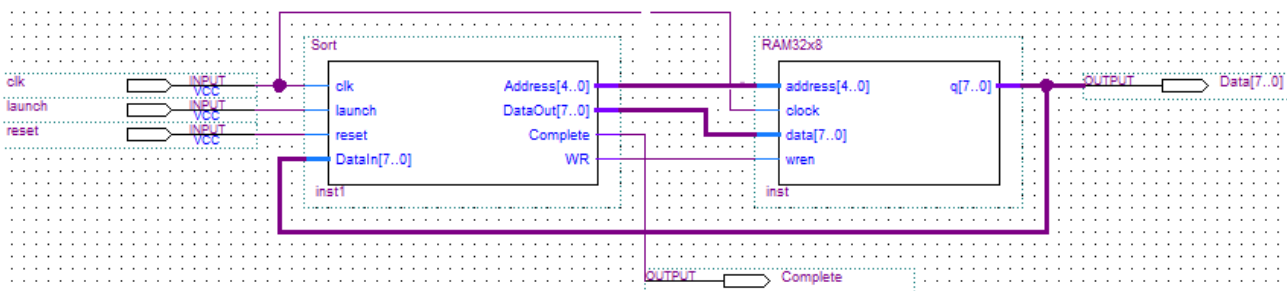
Εικόνα 53. Περιγραφή κυκλώματος ταξινόμησης

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. Ο κώδικας για την περιγραφή του παραπάνω συστήματος φαίνεται στην Εικόνα 53. Αρχικά ορίζουμε τον καταχωρητή κατάστασης state ο οποίος παίρνει τιμές του τύπου απαρίθμησης state_type, τον μετρητή count, τους καταχωρητές δεδομένων dataA, dataB, το σήμα σημαίας και διάφορα άλλα βοηθητικά σήματα. Η αρχιτεκτονική αποτελείται από τις ακόλουθες process

1. **FSM:** Έχει την ευθύνη της μετάβασης των καταστάσεων σε κάθε κύκλο. Καθώς η επόμενη κατάσταση εξαρτάται από την παρούσα και από την τιμή ενός σήματος κάθε φορά, ο πίνακας καταστάσεων έχει δύο διαστάσεις: την παρούσα κατάσταση καθώς και την τιμή του σήματος. Η επιλογή του σωστού σήματος σε κάθε κατάσταση γίνεται με την ανάθεση ροής που ακολουθεί στο σήμα condition (για παράδειγμα, στην κατάσταση Waiting η επόμενη κατάσταση εξαρτάται από την τιμή του σήματος launch).

2. **Cnt:** Μηδενίζει, αυξάνει ή μειώνει την τιμή του μετρητή count για να διαπεράσει το διάνυσμα των τιμών στην μνήμη και να συντονίσει τις εναλλαγές.
3. **MemWait:** Έχει την ευθύνη της αναμονής των δεδομένων από την μνήμη, μέσω του σήματος delay.
4. **Seq:** Πρόκειται για αναθέσεις των ακολουθιακών σημάτων Flag, data, dataB που γίνονται μέσω καταχωρητών (registered) ώστε να διατηρούν τις τιμές τους σε περισσότερους του ενός κύκλου ρολογιού.

Εκτός των παραπάνω process υπάρχουν και κάποιες αναθέσεις σημάτων συνδυαστικές οι οποίες δεν εξαρτώνται από το ρολόι, όπως τα σήματα Address, DataOut, WR κλπ. Παρατηρούμε ότι όσο το κύκλωμα ταξινομεί τον πίνακα, το σήμα Complete είναι στο '0' ενώ όσο βρίσκεται σε αναμονή στην κατάσταση Waiting το σήμα Complete είναι στο '1'.



Εικόνα 54. sorter

Ας δούμε τώρα και την υλοποίηση από την πλευρά του εργαλείου Quartus. Δημιουργούμε ένα νέο αρχείο με κατάληξη .vhd. Όταν ανοίξει ο κειμενογράφος εισάγουμε τον κώδικα που επιθυμούμε. Θυμίζουμε ότι 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.

Εκτελούμε compilation, επιβεβαιώνουμε ότι δεν υπάρχουν λάθη και δημιουργούμε σύμβολο. Το επόμενο βήμα είναι η προσθήκη μνήμης. Δημιουργούμε ένα νέο σχηματικό με όνομα **Sorter** και εισάγουμε το σύμβολο του κυκλώματος ταξινόμησης καθώς και το σύμβολο της μνήμης 32x8 που έχουμε δημιουργήσει. Η διασύνδεση γίνεται με τον τρόπο που φαίνεται στην Εικόνα 54. Αρχικοποιούμε τις 32 θέσεις της μνήμης ως εξής (διευθύνσεις 00-1F) FF, FE, FD, FC, ..., E1, E0 αντιστοίχως. (Βρείτε πως μπορεί αυτό να γίνει μαζικά επιλέγοντας όλες τις θέσεις της μνήμης).

Εξομοίωση του κυκλώματος

Όπως έχουμε προαναφέρει το modelsim δε μπορεί να κάνει εξομοίωση σε αρχεία σχηματικού. Σε πολλές περιπτώσεις το Quartus κάνει αυτόματα την μετατροπή ενός αρχείου σχηματικού σε περιγραφή HDL ώστε να μπορεί να γίνει εξομοίωση με το ModelSim. Μπορείτε εύκολα να ελέγξετε αν ισχύει αυτό στην δική σας περίπτωση προσπαθώντας να εκτελέσετε εξομοίωση με το ModelSim. Εάν επιτύχετε μπορείτε να παραλείψετε την υπόλοιπη παράγραφο, αλλιώς θα πρέπει να μετατρέψετε μόνοι σας το αρχείο του σχηματικού sorter.bdf σε vhd. Αυτό γίνεται από το μενού *File->create/update->create HDL design file from current file*. Στο μενού που εμφανίζεται επιλέγουμε VHDL και πατάμε OK. Αφαιρούμε από το project το αρχείο sorter.bdf, προσθέτουμε το αρχείο sorter.vhd, το κάνουμε top level entity και εκτελούμε compilation.

Κατόπιν δημιουργούμε test-bench για το κύκλωμα Sorter χρησιμοποιώντας χρονική εξομοίωση με το ModelSim. Ορίζουμε μόνο το ρολόι με περίοδο 20ns για μερικούς κύκλους και ολοκληρώνουμε την διαδικασία όπως την έχουμε μάθει. Το test-bench που δημιουργήσαμε θα το τροποποιήσουμε ώστε να γίνει πιο εύχρηστο. Αρχικά θα πρέπει να καθορίσουμε την εκκίνηση της εξομοίωσης χρησιμοποιώντας

το σήμα Reset και το σήμα Launch. Αυτό μπορεί να γίνει με την χρήση της ακόλουθης process, η οποία αρχικοποιεί τα σήματα reset και launch (ενεργοποιεί το reset για ένα μικρό διάστημα ώστε να μηδενιστούν οι καταχωρητές) και κατόπιν θέτει για 1-2 κύκλους ρολογιού το σήμα launch για να προκαλέσει την εκκίνηση του υπολογισμού:

```
process
begin
    reset <= '1';
    launch <= '0';
    wait for 12 ns;
    reset<='0';
    launch<='1';
    wait for 50 ns;
    launch <= '0';
    wait;
end process;
```

Η επόμενη τροποποίηση αφορά τον χρόνο εκτέλεσης της εξομοίωσης. Στην συγκεκριμένη περίπτωση μπορούμε με ακρίβεια να υπολογίσουμε σε πόσο χρόνο θα ολοκληρωθεί ο υπολογισμός αφού ο αλγόριθμος bubble-sort έχει συγκεκριμένο αριθμό βημάτων για το σύνολο των δεδομένων που έχουμε χρησιμοποιήσει. Ωστόσο, ακόμη και αυτός ο υπολογισμός είναι περίπλοκος, ενώ σε τυχαία δεδομένα θα είναι αδύνατος. Έτσι θα πρέπει να προσαρμόσουμε έξυπνα το test-bench ώστε να μπορεί να σταματήσει όταν ολοκληρωθεί η εξομοίωση. Ένα σήμα το οποίο μπορεί να μας βοηθήσει είναι το σήμα Complete το οποίο βρίσκεται στην τιμή '0' κατά την διάρκεια του υπολογισμού, και '1' στην κατάσταση Waiting, δηλαδή πριν την εκκίνηση και μετά το πέρας του υπολογισμού. Αν λοιπόν επιβεβαιώσουμε ότι η τιμή του σήματος Complete είναι '1' τότε αρκεί να επιβεβαιώσουμε ότι βρισκόμαστε στο τέλος και όχι στην αρχή του υπολογισμού. Κατά την εκκίνηση της χρονικής εξομοίωσης η τιμή του σήματος Complete είναι απροσδιόριστη 'U' και με την αρχικοποίηση Reset = '1' γίνεται Complete = '1', οπότε αρκεί η ακόλουθη συνθήκη για να γνωρίζουμε το τέλος του υπολογισμού:

(Complete'last_value = '0' and Complete = '1')

Συνεπώς, μετατρέπουμε το loop που έχει δημιουργήσει το ModelSim για τον ορισμό του ρολογιού (αριστερά) ως εξής (δεξιά):

Ρολόι με προκαθορισμένο αριθμό επαναλήψεων	Ρολόι με μεταβλητό αριθμό επαναλήψεων και αυτόματο τερματισμό
<pre>Process begin clk <= '0' ; wait for 10 ns ; for z in 1 to 5 loop clk <= '1' ; wait for 10 ns ; clk <= '0' ; wait for 10 ns ; end loop; clk <= '1' ; wait for 10 ns ; wait; End Process;</pre>	<pre>Process begin clk <= '0' ; wait for 10 ns ; loop clk <= '1' ; wait for 10 ns ; clk <= '0' ; wait for 10 ns ; exit when (Complete'last_value = '0' and Complete = '1'); end loop; clk <= '1' ; wait for 10 ns ; wait; End Process;</pre>

Εκτελούμε εξομοίωση με το επεξεργασμένο test-bench και παρατηρούμε ότι η εξομοίωση τερματίζεται τον χρόνο 199.1 msec περίπου. Προσοχή, απαιτείται κάποιος χρόνος για να ολοκληρωθεί η εξομοίωση καθώς θα εκτελεστούν πολλοί κύκλοι ρολογιού. Μπορούμε να παρακολουθήσουμε τις τιμές των σημάτων εξόδου (μέσα σε αυτές είναι και τα δεδομένα της κάθε διεύθυνσης μνήμης που κλειδώνεται στον MAR) τα οποία τα οδηγήσαμε στην έξοδο ακριβώς για αυτόν τον λόγο. Για

καλύτερη παρατήρηση των αποτελεσμάτων αλλά και για τα screenshots που θα πρέπει να βάλετε στην αναφορά σας επιλέξτε την εμφάνιση του σήματος Data να είναι στο **δεκαεξαδικό**.

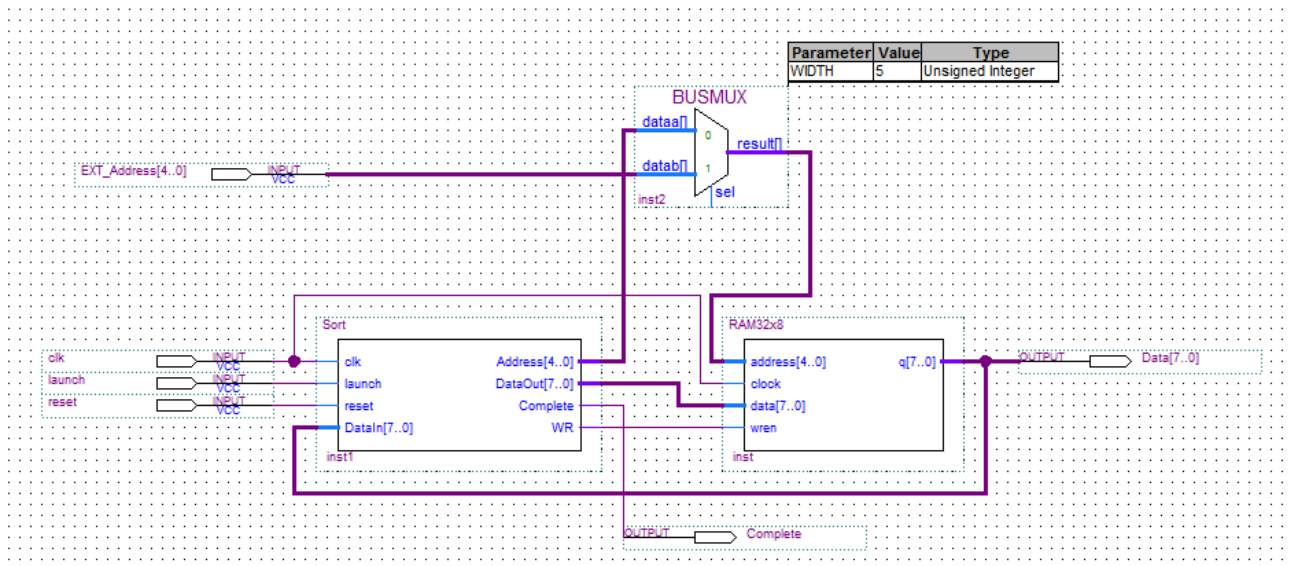
Αν παρακολουθήσουμε τους τελευταίους κύκλους ρολογιού ξεκινώντας περίπου από τον χρόνο 194.1 μsec θα παρατηρήσουμε ότι οι διευθύνσεις τοποθετούνται διαδοχικά από την 00 έως την 1F και τα δεδομένα που βγαίνουν είναι σε αύξουσα σειρά καθώς δεν γίνεται πλέον καμία αντιμετάθεση. Έτσι μπορούμε να καταλάβουμε ότι ο αλγόριθμος έτρεξε σωστά. Αν παρόλα αυτά θέλουμε να επιβεβαιώσουμε τα δεδομένα στην μνήμη μετά το πέρας της εξομοίωσης επιλέγουμε το tab Memory List (VIEW ->Memory List) και εκεί βρίσκουμε την μνήμη RAM 32 θέσεων την οποία μπορούμε να ανοίξουμε και να δούμε με διπλό κλικ τα περιεχόμενα οποιασδήποτε θέσης της. Προσοχή, υπάρχουν δύο στιγμιότυπα της μνήμης, το ένα περιέχει τις αρχικές τιμές και το άλλο τις τελικές.

Στην αναφορά σας σε σχέση με αυτή την εξομοίωση θα πρέπει να συμπεριλάβετε 2 screensots με κατάλληλο zoom έτσι ώστε να φαίνονται καθαρά οι τιμές των σημάτων αλλά και να φαίνεται όσο το δυνατόν μεγαλύτερο μέρος της εξομοίωσης. ΠΡΟΣΟΧΗ: πρέπει οπωσδήποτε να φαίνονται τα ονόματα των σημάτων και ο άξονας του χρόνου. Το πρώτο screenshot να δείχνει την αρχή της εξομοίωσης και το δεύτερο πρέπει να δείχνει το τέλος της εξομοίωσης. **Για το πρώτο αναφέρετε ποιές εναλλαγές τιμών φαίνονται ολοκληρωμένες μέσα σε αυτό (δικαιολογήστε συνοπτικά)**. Υπενθυμίζεται ότι όταν τροποποιούμε ΜΟΝΟ τον κώδικα του testbench, για να ξανατρέξει η εξομοίωση αρκεί να εκτελέσουμε εκ νέου την τελευταία εντολή στη γραμμή εντολών του modelsim (transcript) η οποία ανακαλείται με το επάνω βελάκι.

Υλοποίηση και έλεγχος στο board

Για να δοκιμάσουμε την λειτουργία του κυκλώματος στο board θα πρέπει να δώσουμε την δυνατότητα οπτικού ελέγχου του αποτελέσματος. 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 55 (μέσω των ιδιοτήτων properties/WIDTH μπορούμε να καθορίσουμε το πλάτος εισόδων/εξόδου σε 5). Στην είσοδο SEL του πολυπλέκτη συνδέεται το σήμα complete.

Κάνουμε σύμβολο το τροποποιημένο κύκλωμα του sorter και το τοποθετούμε σε νέο σχηματικό, το οποίο θα είναι και το τελικό top level entity για το board. Στο ίδιο σχηματικό τοποθετούμε έναν μετρητή των 5 bits τον οποίο σχεδιάζουμε χρησιμοποιώντας την Megafunction lpm_counter. Με την βοήθεια του MegaWizard Plug-In Manager παραμετροποιούμε τον μετρητή, ο οποίος θα έχει είσοδο ασύγχρονης μηδένισης καθώς και είσοδο επίτρεψης μέτρησης (enable).



Εικόνα 55. sorter version 2

Συνδέουμε την έξοδο Complete στην είσοδο επίτρεψης 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. Το ρολόι του sorter θα είναι το clock 50MHz. Στο ίδιο σχηματικό τοποθετούμε το LCD Display και κάνουμε τις απαραίτητες συνδέσεις ώστε να δίνει το ακόλουθο μήνυμα που φαίνεται στον παρακάτω πίνακα, όπου XX, YY οι διευθύνσεις-περιεχόμενα της μνήμης. Προκειμένου να συνδέσετε το σήμα της διεύθυνσης (το οποίο είναι 5bit) στις κατάλληλες εισόδους του LCD DISPLAY (οι οποίες έχουν πλάτος 4bit η κάθε μία) θα χρειαστεί να υλοποιήσετε ένα κατάλληλο κύκλωμα μετατροπής. 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. Αρχικοποιούμε την μνήμη RAM στις τιμές FF, FE, FD, ..., E0 και προγραμματίζουμε το FPGA (στο εργαστήριο)

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

Παραδοτέα 7^{ης} Εργαστηριακής Άσκησης

1. Παρουσιάστε τα αποτελέσματα των εξομοιώσεων στους επιτηρητές σας.
2. Για αυτό το ερώτημα θα πρέπει να κάνετε τα εξής:
 - a. Αλλάξτε τον αριθμό των θέσεων της μνήμης σε 256 με αρχικοποίηση: διευθύνσεις 0-255, τιμές FF – 00 (θα χρειαστείτε αρχείο για αρχικοποίηση).
 - b. Τροποποιήστε τα κυκλώματα **sort** και **sorter** (εικ 55 και 54) έτσι ώστε i) να συνεργάζονται με τη νέα μνήμη και ii) να υπάρχει δυνατότητα ταξινόμησης μιας επιλεγμένης περιοχής της μνήμης από τη διεύθυνση A μέχρι και την διεύθυνση B όπου τα A, B θα καθορίζονται κατά την εκτέλεση μέσω των Switches (A, B μεταξύ 0-255 και $A \leq B$ με ευθύνη του χρήστη δηλαδή δεν θα το ελέγχει το κύκλωμα).
 - c. Επαναλάβετε τη διαδικασία που περιγράφεται στην ενότητα “Εξομοίωση του κυκλώματος” και στη συνέχεια τροποποιείτε κατάλληλα το sorter_tb ώστε να κάνετε εξομοίωση στο νέο κύκλωμα του sorter. Στο σήμα EXT_Address δώστε μια τυχαία σταθερή τιμή (δεν παίζει ρόλο για το τμήμα της εξομοίωσης που παρακολουθούμε). Αρχικά κάντε εξομοίωση για $A=0$, $B=31$ και επιβεβαιώστε ότι η εξομοίωση είναι ίδια με αυτή του αρχικού κυκλώματος.

- d. Κάντε εξομοίωση με A ίσο με τα 2 τελευταία ψηφία του AM ενός εκ των μελών της ομάδας (να γράψετε στην αναφορά ποιος είναι αυτός ο αριθμός) και $B=A+3$. Την εξομοίωση αυτή να παρουσιάσετε ολόκληρη στην αναφορά σας με screenshots (θα χρειαστούν 3-4) σύμφωνα με τις ίδιες οδηγίες που ακολουθήσατε για τις εξομοιώσεις του αρχικού κυκλώματος.
3. Κάντε σύμβολο το νέο sorter, τοποθετήστε τον στο σχηματικό που είναι το top level entity για το board. Κάντε ΟΛΕΣ τις απαραίτητες τροποποιήσεις ώστε το κύκλωμα να είναι πλήρως λειτουργικό.
 4. Προγραμματίστε στο FPGA το κύκλωμα ταξινόμησης και ελέγξτε την λειτουργία του όπως περιγράψαμε παραπάνω.
 5. Συντάξτε σύντομη και περιληπτική αναφορά όπου θα συμπεριλάβετε όσα κυκλώματα σχεδιάσατε στα πλαίσια της άσκησης, καθώς και τα αποτελέσματα των εξομοιώσεων που εκτελέσατε. Η αναφορά πρέπει να υποβληθεί στην ιστοσελίδα που σας έχει υποδειχθεί από τον διδάσκοντα του μαθήματος εντός της προκαθορισμένης προθεσμίας.

Εργαστηριακή Άσκηση 8: Σχεδίαση με μνήμες

Στην 8^η εργαστηριακή άσκηση θα αναπτύξουμε μία μονάδα πολλαπλασιασμού διανυσμάτων $A = [A(0), A(1), A(2), \dots, A(n-1)]$ και $B^T = [B(0), B(1), B(2), \dots, B(n-1)]$ με $n \leq 128$.

$$A \times B = [A(0), A(1), A(2), \dots, A(n-1)] \times [B(0), B(1), B(2), \dots, B(n-1)]^T = \\ A(0) \bullet B(0) + A(1) \bullet B(1) + \dots + A(n-1) \bullet B(n-1)$$

Όλες οι τιμές των διανυσμάτων $A[i]$, $B[i]$ είναι 4-bit η κάθε μία, και το τελικό αποτέλεσμα είναι $A \times B$ είναι των 16 bit.

1. Για το κύκλωμα πολλαπλασιασμού των διανυσμάτων θα χρησιμοποιήσετε τον ακολουθιακό πολλαπλασιαστή των 4 δυαδικών ψηφίων παράλληλης φόρτωσης που σχεδιάσατε στην άσκηση 6, τον οποίο θα εισάγετε ως σύμβολο σε σχηματικό. Τα δεδομένα των διανυσμάτων A , B θα είναι αποθηκευμένα στην μνήμη μεγέθους 256×4 , με το διάνυσμα A να βρίσκεται στις θέσεις $0 \dots 127$, και το διάνυσμα B να βρίσκεται στις θέσεις $128 \dots 255$ για $n=128$. Το αποτέλεσμα θα το αποθηκεύσετε σε έναν καταχωρητή και θα το προβάλετε στο LCD Display.
2. Για την εύκολη επαλήθευση της ορθής λειτουργίας του κυκλώματος, τα διανύσματα A , B θα περιέχουν πάντα τους αριθμούς $0-15$ σε αύξουσα σειρά και με κυκλική επανάληψη (δηλ ... 14, 15, 0, 1, 2 ...). Η μόνη διαφοροποίηση που θα υπάρχει θα είναι η τιμή **a** της πρώτης θέσης μνήμης του A , δηλαδή η τιμή του **A[0]**, και αντίστοιχα η τιμή **b** του **B[0]**. Αν πχ $a=5$ θα έχουμε το $A=[5, 6, 7, \dots]$. Έτσι θα είναι εύκολο να φτιάξετε ένα πρόγραμμα ή ένα φύλλο excel το οποίο θα υπολογίζει το αποτέλεσμα του πολλαπλασιασμού για οποιαδήποτε a , b και να επαληθεύσετε τα αποτελέσματα που δίνει το κύκλωμα.
Να εκτελέσετε 2 προσομοιώσεις, μία για $a=b=0$ και μία με τα καθένα από τα a , b να είναι ίσο το τελευταίο ψηφίο των AM των μελών της ομάδας σας. Για κάθε προσομοίωση να συμπεριλάβετε 2 screenshots, ένα στην αρχή και ένα στο τέλος της προσομοίωσης. **ΠΡΟΣΟΧΗ:** στην προσομοίωση τα σήματα να παρουσιάζονται απαραίτητα στο **δεκαεξαδικό** σύστημα.
3. Σχεδιάστε μία δεύτερη αρχιτεκτονική πολλαπλασιασμού πίνακα (array multiplier) για τον ίδιο πολλαπλασιαστή (θα αντιστοιχεί στο ίδιο entity). Χρησιμοποιήστε περιγραφή δομής βασισμένη σε περιγραφές συμπεριφοράς ημιαθροιστή και πλήρους αθροιστή. Αντικαταστήστε τον ακολουθιακό πολλαπλασιαστή και επαναλάβετε τον υπολογισμό του πολλαπλασιασμού των διανυσμάτων. Επαναλάβετε με το νέο κύκλωμα τις ίδιες προσομοιώσεις και συμπεριλάβετε στην αναφορά σας τα ίδια screenshots που περιγράφονται στο βήμα 2.
4. Συγκρίνετε τα δύο πειράματα σε ταχύτητα/επιφάνεια. Μπορείτε να αυξήσετε την ταχύτητα υπολογισμού; Προτείνετε λύση και δείξτε την στον επιτηρητή σας.
5. Για όλες τις εξομοιώσεις χρησιμοποιήστε μόνο το ModelSim και χρονική εξομοίωση με χρήση test-benches.

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