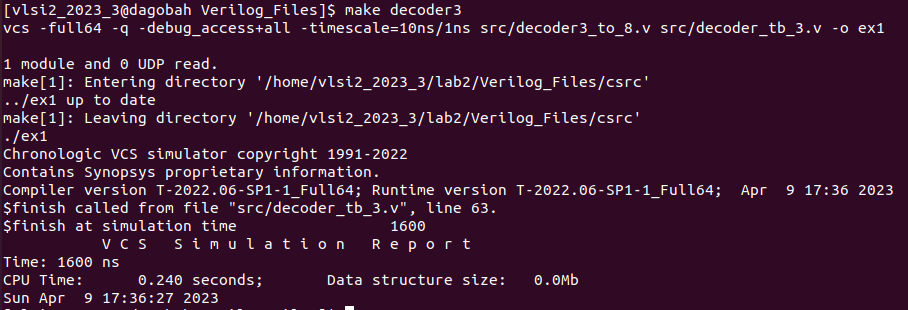
# 2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

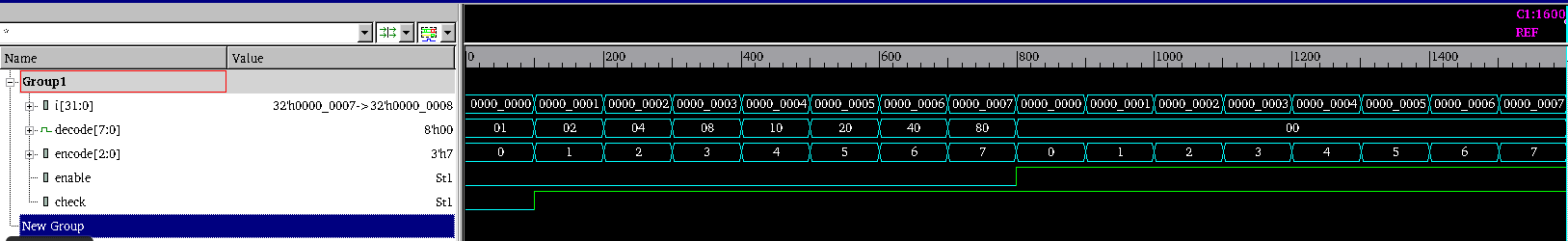
# “ΣΧΕΔΙΑΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ VLSI”

Ζητούμενα:

* 1. Για το ζητούμενο 1.1 μας ζητήθηκε να υλοποιήσουμε έναν decoder 3 σε 8. Για την υλοποίηση του decoder χρησιμοποιήσαμε την εντολή case της Verilog η οποία αφού ελέγχει την τιμή της εισόδου του κυκλώματος (enc), αφού μηδενίσει την τιμή της εξόδου, θέτει το αντίστοιχο bit της εξόδου στην τιμή 1. Ο κώδικας υλοποίησης επισυνάπτεται στο αρχείο \SRC\decoder3\_to\_8.v
  2. Για το ζητούμενο 1.2 μας ζητήθηκε η υλοποίηση ενός testbench για n-2n που να ελέγχει και την λειτουργία του decoder που υλοποιήθηκε στο ζητούμενο 1.1. Αυτό το πετύχαμε χρησιμοποιώντας μία μεταβλητή check η οποία έπαιρνε την τιμή της με βάση την συνθήκη “if(decode[encode]==1'b1)” , δηλαδή εάν το bit της εξόδου στην θέση που ορίζεται από την είσοδο encode είναι 1 τότε και η μεταβλητή check παίρνει την τιμή 1 και εγγράφεται η είσοδος και η έξοδος σε ένα εξωτερικό αρχείο (file1.txt, το οποίο επισσυνάπτεται στον φάκελο CODE\_PRODUCED\_FILES) αλλιώς η τιμή check γίνεται 0 και εμφανίζεται μήνυμα σφάλματος στο terminal το οποίο μας ενημερώνει σε ποια τιμή του encode βρέθηκε το σφάλμα. Επίσης γίνεται έλεγχος για την περίπτωση που το enable είναι στην τιμή 1 όπου κανονικά ο decoder θα πρέπει να έχει την τιμή 0. Ακολουθούν φωτογραφίες του terminal και των κυματομορφών κατά την εκτέλεση της προσομοίωσης. (Ο κώδικας για το testbench επισυνάπτεται στο αρχείο SRC\decoder\_tb\_3.v)



Εικόνα 1: Decoder 3 to 8 Terminal Execution



Εικόνα : Decoder 3 to 8 Waveforms

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 3: Decoder 3 to 8 File\_Out

* 1. Για το ζητούμενο 1.3 μας ζητήθηκε να υλοποιήσουμε έναν decoder 4 σε 16 σε structural Verilog. Για την υλοποίηση αυτού του ερωτήματος χρησιμοποιήσαμε 2 decoders 3 σε 8 συνδεδεμένους με τις εισόδους enc στα 3 LSB της εισόδου enc του μεγάλου decoder και τις εισόδους enable (αντεστραμμένη στον έναν 3-8\_Decoder) συνδεδεμένη στο MSB της εισόδου enc του μεγάλου Decoder. Αυτό δεν μας επέτρεπε να χρησιμοποιήσουμε τα σήματα enable των Decoders 3-8 για να απενεργοποιήσουμε την έξοδο του μεγάλου Decoder αν δεν χρησιμοποιούσαμε εξωτερικό σήμα global enable. Για το testing του Decoder χρησιμοποιήσαμε το testbench που δημιουργήσαμε στο παραπάνω ερώτημα (αγνοώντας ωστόσο την περίπτωση που το enable είναι 1 καθώς ο Decoder μας δεν έχει την δυνατότητα για Enable χωρίς την προσθήκη πρόσθετης λογικής) (Ο κώδικας για τον Decoder 4-16 επισυνάπτεται στο αρχείο SRC\decoder4\_to\_16\_final.v)

Εικόνα που περιέχει διάγραμμα

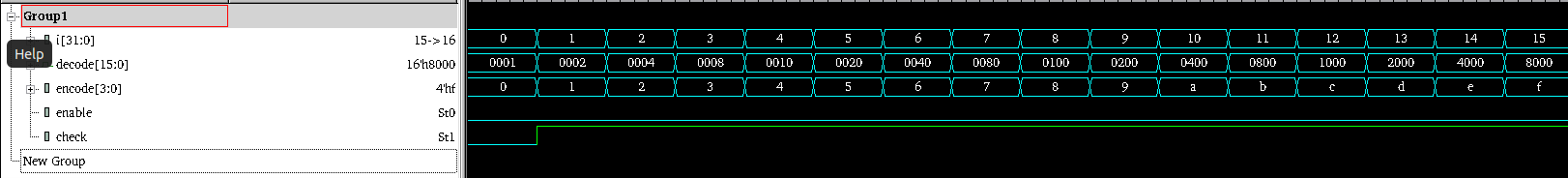
Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα :Σχηματικό Υλοποίησης 4-16 Decoder

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 5:Decoder 4 to 16 Terminal Execution



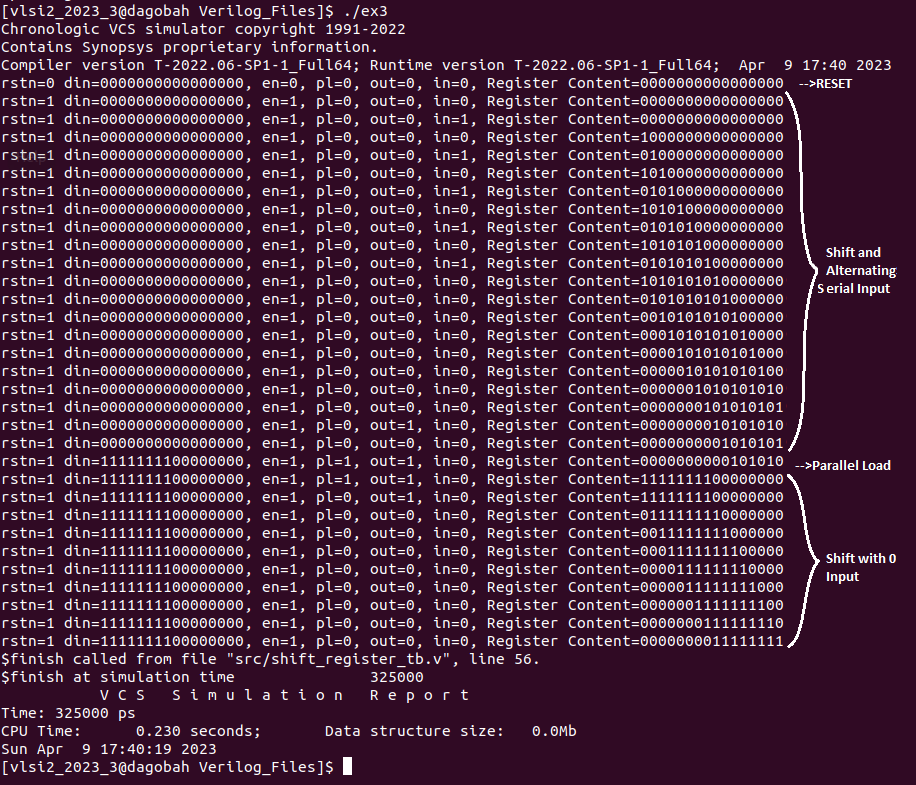
Εικόνα :Decoder 4 to 16 Waveforms

Εικόνα που περιέχει τραπέζι

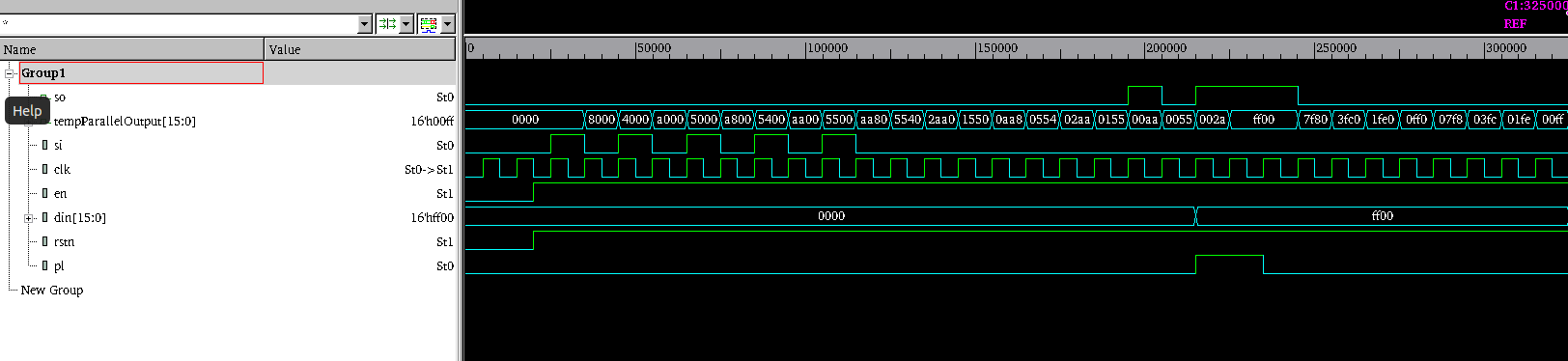
Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 7:Decoder 4 to 16 File\_Out

* 1. Για το δεύτερο ζητούμενο μας ζητήθηκε να υλοποιήσουμε έναν shift register των n bits με δυνατότητα παράλληλης φόρτωσης. Αφού δηλώσαμε το module με την δυνατότητα παραμετροποίησης “module shift\_reg #(parameter n=8)” όπου n το μέγεθος του shift register, και αφού ελέγχουμε το ασύγχρονο σήμα rstn το οποίο αν είναι ενεργοποιημένο (τιμή 0) θα πρέπει ασύγχρονα ο shift register να πάρει την τιμή 0, ελέγχουμε την τιμή του σήματος pl μέσω μιας if ώστε να διαπιστώσουμε εάν θα πρέπει να φορτωθεί παράλληλα η τιμή του din στον καταχωρητή. Για να μπορέσουμε να ελέγξουμε την τιμή του περιεχομένου του καταχωρητή, χρησιμοποιήσαμε την μεταβλητή tempParallelOutput στην οποία σε κάθε κύκλο που γίνεται shift ξανααναθέτουμε τα n-1 σημαντικότερα bit της στα n-1 λιγότερο σημαντικά και στο MSB της εισάγουμε τo σήμα si ενώ στην έξοδο so αναθέτουμε το LSB πριν την ανάθεση της νέας τιμής στην μεταβλητή. Για την υλοποίηση του testbench υλοποιήσαμε έναν 16bit shift Register τον οποίο αφού τον αρχικοποιήσαμε θέτοντας τον σε λειτουργία reset, του εισάγαμε διάφορες τιμές στις εισόδους του και παρατηρήσαμε «με το μάτι» αν οι αντίστοιχες λειτουργίες εκτελούνται σωστά στο terminal καθώς και χρησιμοποιώντας κυματομορφές. (Οι κώδικες για το κύκλωμα και το testbench επισυνάπτονται στο αρχείο SRC\shift\_register.v και shift\_register\_tb.v)

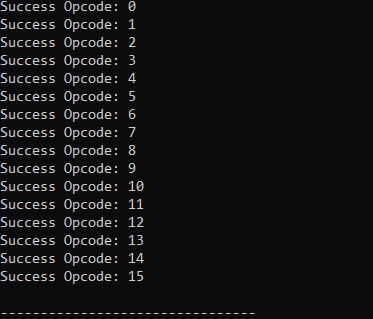


Εικόνα : Shift Register Terminal Execution

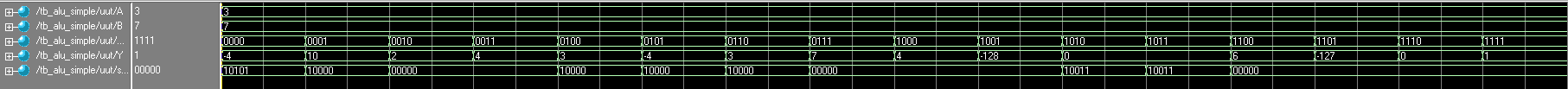


Εικόνα : Shift Register Waveforms

* 1. Το 3.1 ζητούμενο δεν υλοποιήθηκε.
  2. Για το ζητούμενο 4.1. ζητήθηκε η υλοποίηση μιας Αριθμητικής και Λογικής Μονάδας καθώς και τα status flags της. Για την υλοποίηση της ALU χρησιμοποιήθηκε η case της Verilog με βάση την οποία επιλέγουμε την πράξη που θα κάνουμε μέσα σε ένα Always Block το οποίο ενεργοποιείται κάθε φορά που αλλάζει κάποια από τις 3 εισόδους (OpCode, A, B). Επίσης μαζί με τον υπολογισμό κάθε πράξης υπολογίζονται και τα αντίστοιχα bit της εξόδου status. Για τον έλεγχο του κυκλώματος χρησιμοποιήσαμε το testbench για να θέσουμε διάφορες τιμές στις εισόδους A και B του κυκλώματος και να εκτελέσουμε όλες τις δυνατές πράξεις της ALU, ενώ ταυτόχρονα γράφουμε τα αποτελέσματα σε εξωτερικό αρχείο (επισυνάπτεται στο CODE\_PRODUCED\_FILES\ALU.txt) το οποίο ελέγχεται στη συνέχεια από εξωτερικό πρόγραμμα γραμμένο σε C++ του οποίου ο κώδικας επισυνάπτεται στο αρχείο alu\_check.cpp (από το πρόγραμμα ελέγχονται μόνο τα αποτελέσματα των αριθμητικών πράξεων και των λογικών, πλην των ολισθήσεων και όχι τα status bits). (Οι κώδικες για το κύκλωμα και το testbench επισυνάπτονται στο αρχείο SRC\alu.v και alu\_tb\_simple.v).



Εικόνα : Έλεγχος εκτέλεσης testbench με εξωτερικό πρόγραμμα



Εικόνα :ALU Execution Waveforms on Modelsim

Εικόνα που περιέχει τραπέζι

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα : ALU Produced File\_Out