Λογική Σχεδίαση Ι

Σετ Ασκήσεων Verilog

Ονοματεπώνυμο: Γιώργος Ντάκος
Αριθμός Μητρώου: 1059569

Κεφάλαιο 1

Άσκηση 1

1.1 Άσκηση 1 — Εγκατάσταση και βασική εξομοίωση ModelSim

Η πρώτη άσκηση έχει ως στόχο την εξοικείωση με τον εξομοιωτή ModelSim και τη βασική διαδικασία μεταγλώττισης και εκτέλεσης σχεδιασμών σε Verilog.

Στο πρώτο μέρος της άσκησης πραγματοποιείται η εγκατάσταση του εξομοιωτή, ενώ στο δεύτερο μέρος εκτελείται μία δοκιμαστική εξομοίωση του παρεχόμενου αρχείου 1.v, ώστε να επιβεβαιωθεί η ορθή λειτουργία του περιβάλλοντος.

1.1.1 Μεταγλώττιση και Εκτέλεση

Το αρχείο 1.v περιέχει πολλαπλά modules σε Verilog, μεταξύ των οποίων:

- testHam (top-level module)
- hamEncode
- hamDecode
- xor8
- deMux

Αρχικά δημιουργήθηκε βιβλιοθήκη work και έγινε μεταγλώττιση του αρχείου. Στη συνέχεια επιλέχθηκε για εξομοίωση το module testHam.

1.1.2 Παρακολούθηση Σημάτων

Προστέθηκαν στο παράθυρο κυματομορφών όλα τα σήματα του module μέσω της επιλογής:

Add → Wave → Signals in Region

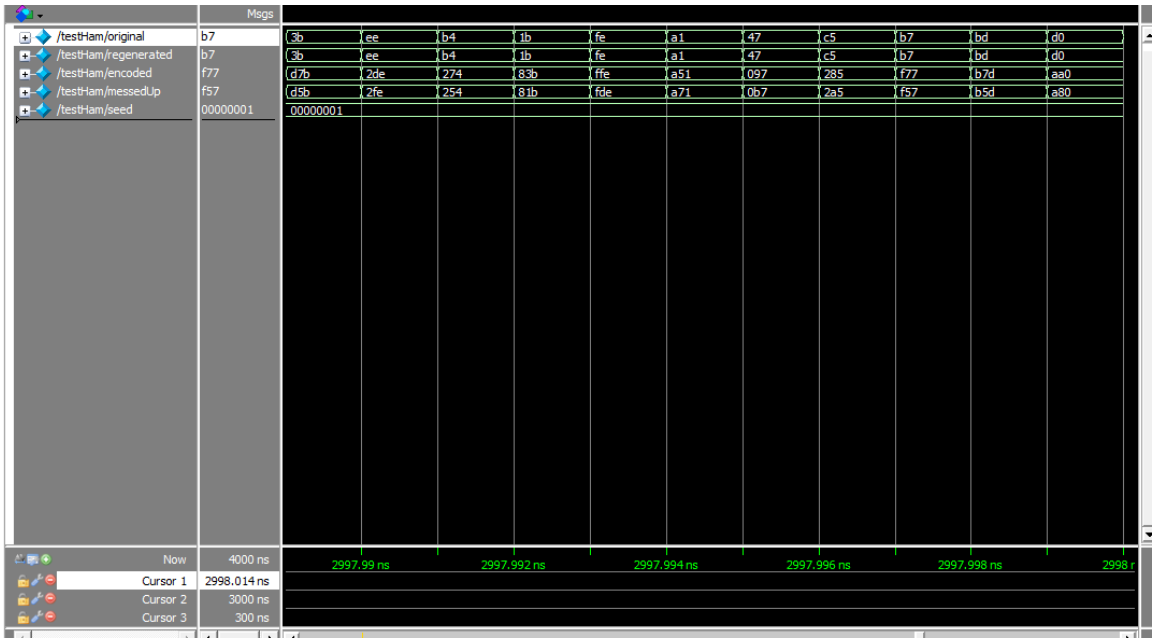
Η εξομοίωση εκτελέστηκε μέχρι χρόνο 3000 ns και έγινε εστίαση (zoom) στο διάστημα 2998–3000 ns, όπως ζητείται στην εκφώνηση.

1.1.3 Αποτελέσματα

Στο τελικό χρονικό διάστημα παρατηρούνται οι τιμές των σημάτων:

- original — αρχικά δεδομένα
- encoded — δεδομένα μετά την κωδικοποίηση Hamming
- messedUp — δεδομένα μετά από σκόπιμη εισαγωγή σφάλματος
- regenerated — δεδομένα μετά την αποκωδικοποίηση
- seed — αρχική τιμή γεννήτριας τυχαίων αριθμών

Παρατηρείται ότι το αποκωδικοποιημένο σήμα regenerated ταυτίζεται με το αρχικό original, γεγονός που επιβεβαιώνει τη σωστή λειτουργία του συστήματος ανίχνευσης και διόρθωσης σφαλμάτων.



Σχήμα 1.1: Κυματομορφές εξομοίωσης του module testHam στο διάστημα 2998–3000 ns

1.1.4 Συμπέρασμα

Η επιτυχής εκτέλεση της εξομοίωσης και η ορθή αποκατάσταση των δεδομένων μετά την εισαγωγή σφάλματος επιβεβαιώνουν ότι ο εξομοιωτής έχει εγκατασταθεί σωστά και ότι το περιβάλλον ανάπτυξης λειτουργεί κανονικά.

Κεφάλαιο 2

Άσκηση 2 — Βασικές λογικές πύλες

2.1 Άσκηση 2 — Περιγραφή βασικών λογικών πυλών σε Verilog

Η δεύτερη άσκηση αφορά την περιγραφή και εξομοίωση βασικών λογικών πυλών (AND και OR) χρησιμοποιώντας τη γλώσσα Verilog.

2.1.1 Μέρος 1 — Πύλη AND

Η πύλη AND δύο εισόδων δίνει στην έξοδο την τιμή 1 μόνο όταν και οι δύο είσοδοι είναι 1. Η λογική συνάρτηση εκφράζεται ως:

$$C = A \cdot B$$

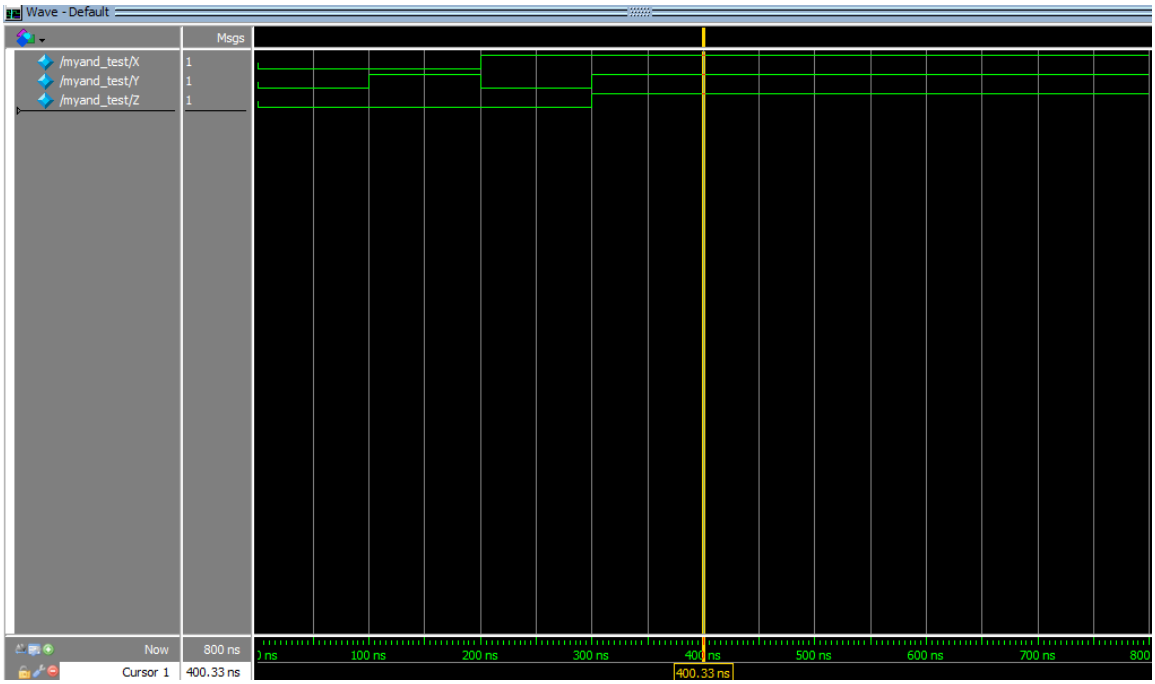
Στην υλοποίηση χρησιμοποιήθηκε η πρωταρχική συνάρτηση and της Verilog.

Για την εξομοίωση δημιουργήθηκε testbench (myand_test) που εφαρμόζει όλους τους δυνατούς συνδυασμούς εισόδων στις μεταβλητές X και Y, με χρονική απόσταση 100 ns μεταξύ τους.

Αποτελέσματα

Οι κυματομορφές επιβεβαιώνουν τον πίνακα αλήθειας της πύλης AND.

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



Σχήμα 2.1: Κυματομορφές εξομοίωσης της πύλης AND

2.1.2 Μέρος 2 — Πύλη OR

Η πύλη OR δύο εισόδων δίνει στην έξοδο την τιμή 1 όταν τουλάχιστον μία είσοδος είναι 1. Η λογική συνάρτηση εκφράζεται ως:

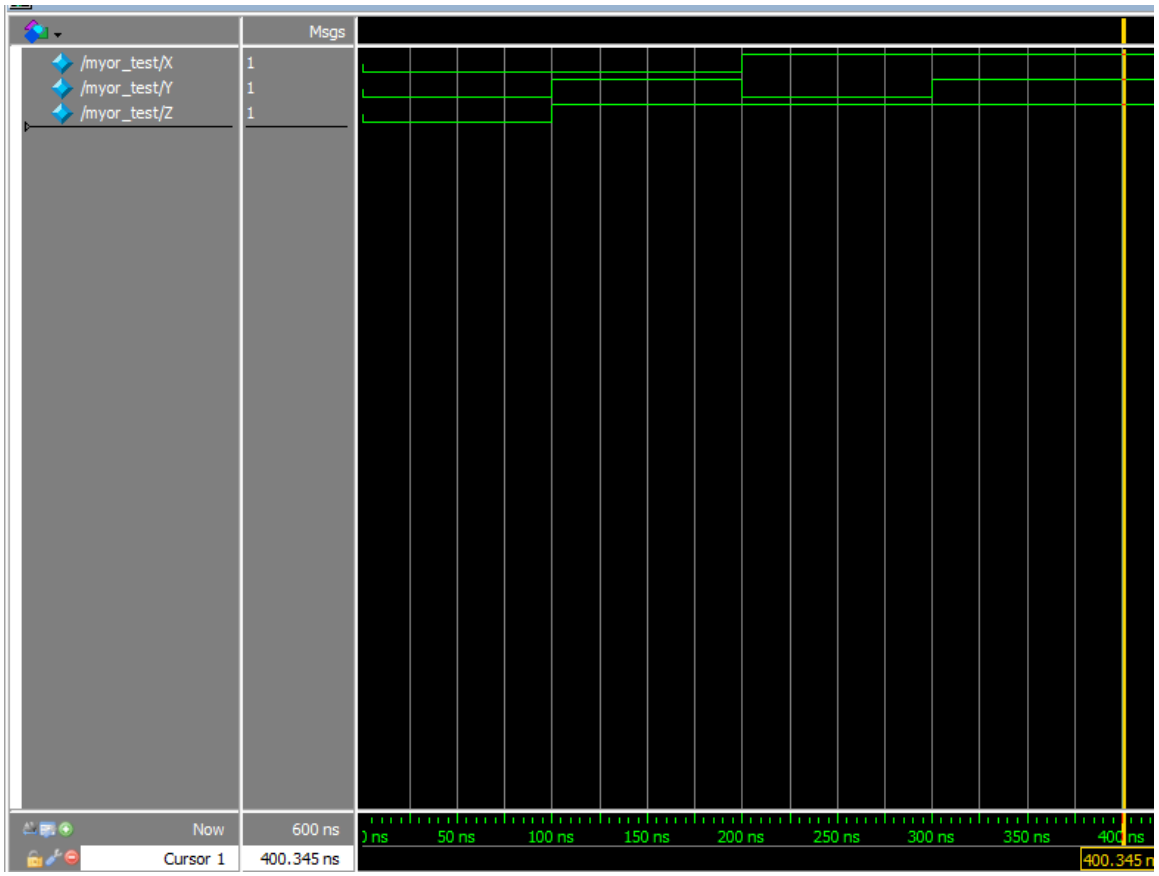
$$C = A + B$$

Η υλοποίηση έγινε με τη πρωταρχική συνάρτηση or της Verilog και δημιουργήθηκε αντίστοιχο testbench (myor_test).

Αποτελέσματα

Οι κυματομορφές επιβεβαιώνουν τον πίνακα αλήθειας της πύλης OR.

X	**Y**	**Z**
0	0	0
0	1	1
1	0	1
1	1	1



Σχήμα 2.2: Κυματομορφές εξομοίωσης της πύλης OR

2.1.3 Συμπέρασμα

Η εξομοίωση των δύο πυλών επιβεβαιώνει τη σωστή λειτουργία των βασικών λογικών τελεστών της Verilog και την αντιστοίχισή τους με τις αντίστοιχες λογικές συναρτήσεις της Άλγεβρας Boole.

Κεφάλαιο 3

Άσκηση 3

3.1 Άσκηση 3 — Δομική περιγραφή συνδυαστικών κυκλωμάτων

Σκοπός της άσκησης είναι η εξοικείωση με τον δομικό τρόπο περιγραφής κυκλωμάτων στη Verilog, χρησιμοποιώντας τις πρωταρχικές συναρτήσεις λογικών πυλών.

3.1.1 Μέρος 1 — Κύκλωμα με AND και OR

Το κύκλωμα αποτελείται από δύο πύλες AND και μία πύλη OR με εισόδους A, B, C και έξοδο Z.

Οι ενδιάμεσες μεταβλητές είναι:

$$K = A \cdot B$$

$$L = B \cdot C$$

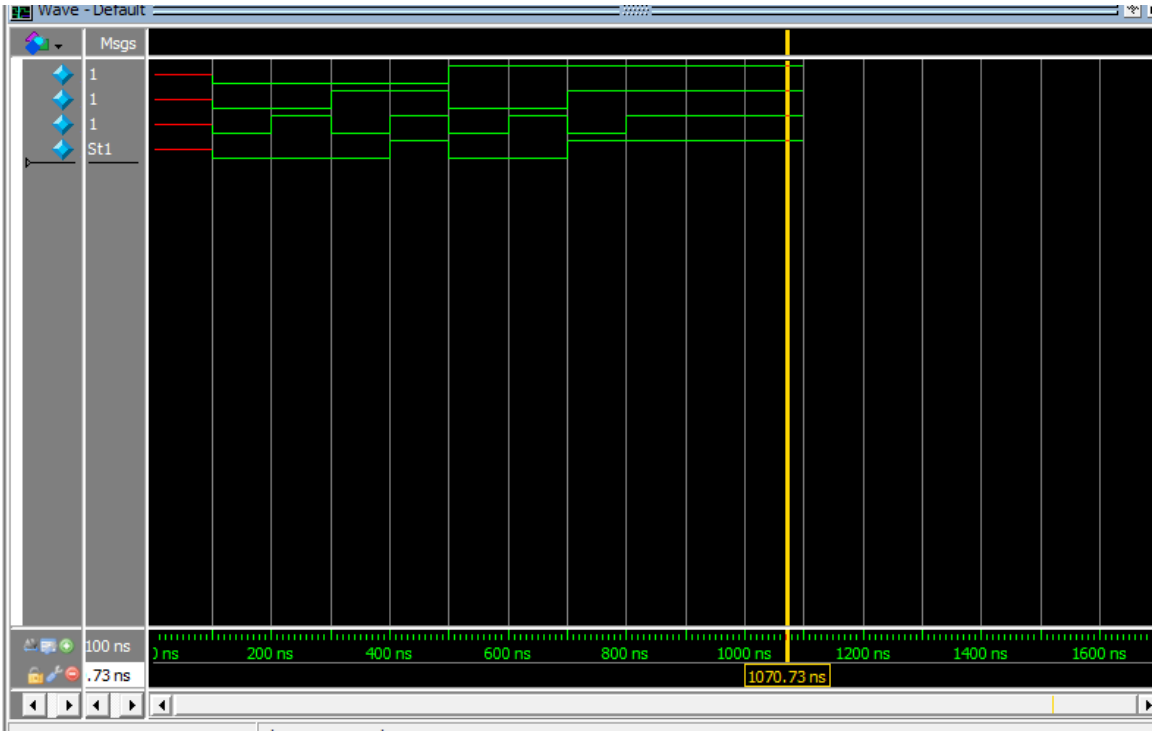
και η τελική έξοδος:

$$Z = K + L = AB + BC$$

Στην υλοποίηση χρησιμοποιήθηκαν οι πρωταρχικές συναρτήσεις and και or της Verilog.

Εξομοίωση

Στο testbench εφαρμόστηκαν όλοι οι δυνατοί συνδυασμοί εισόδων ($2^3 = 8$) ανά 100 ns.



Σχήμα 3.1: Κυματομορφές εξομοίωσης του κυκλώματος circuit3_1

Παρατηρείται ότι η έξοδος Z συμφωνεί με τον θεωρητικό πίνακα αληθείας.

Αιχμές (Spikes)

Στη χρονική στιγμή περίπου 300 ns εμφανίζεται στιγμιαία μεταβολή στην έξοδο (spike). Το φαινόμενο αυτό οφείλεται σε διαφορετικές καθυστερήσεις διάδοσης των πυλών, με αποτέλεσμα να δημιουργείται προσωρινά συνδυασμός εισόδων που οδηγεί σε διαφορετική τιμή εξόδου.

3.1.2 Μέρος 2 — Κύκλωμα με AND, NAND και XOR

Στο δεύτερο κύκλωμα χρησιμοποιούνται οι πύλες:

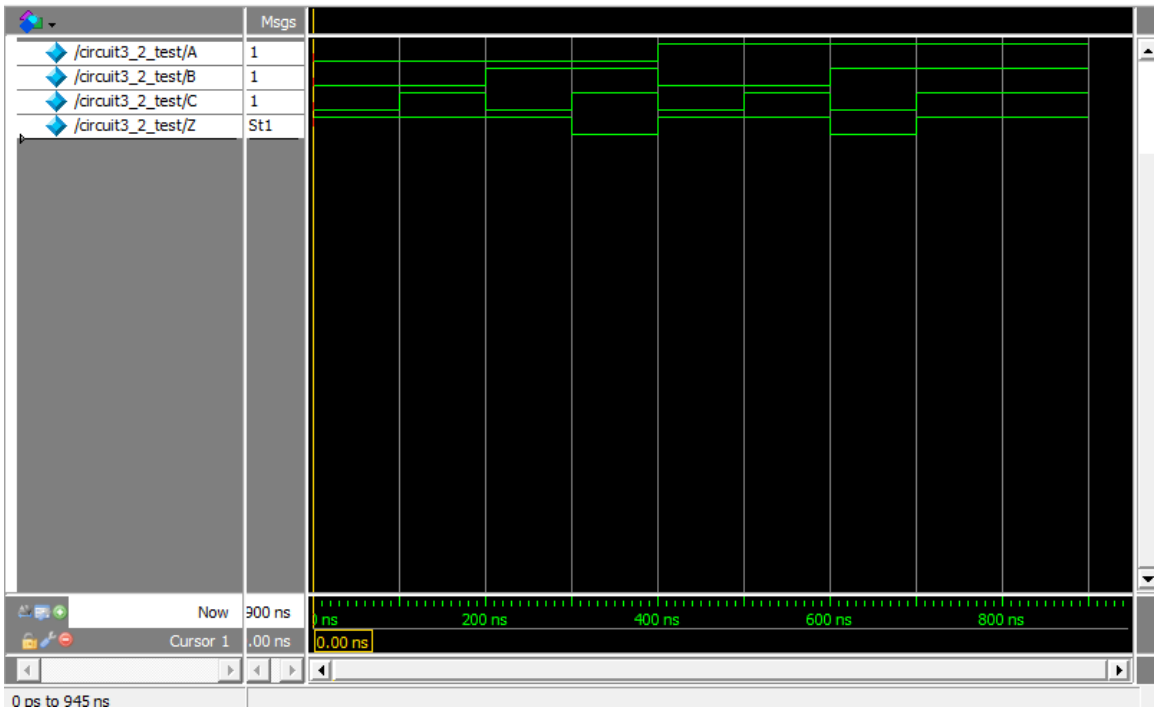
- AND μεταξύ των A και B
- NAND μεταξύ των B και C
- XOR μεταξύ των ενδιάμεσων σημάτων

Οι σχέσεις των σημάτων είναι:

$$K = A \cdot B$$
$$L = \overline{B \cdot C}$$
$$Z = K \oplus L$$

Εξομοίωση

Οι είσοδοι A, B και C μεταβάλλονται διαδοχικά ώστε να καλύπτονται όλοι οι δυνατοί συνδυασμοί.



Σχήμα 3.2: Κυματομορφές εξομοίωσης του κυκλώματος circuit3_2

Η έξοδος Z συμφωνεί με τη θεωρητική λογική συνάρτηση του κυκλώματος.

3.1.3 Συμπέρασμα

Η δομική περιγραφή στη Verilog επιτρέπει την απευθείας αντιστοίχιση ενός λογικού κυκλώματος σε κώδικα χρησιμοποιώντας πρωταρχικές συναρτήσεις πυλών. Τα αποτελέσματα της εξομοίωσης επιβεβαιώνουν τη σωστή λειτουργία και αναδεικνύουν φαινόμενα όπως οι αιχμές, που προκύπτουν από τις καθυστερήσεις διάδοσης.

Κεφάλαιο 4

Άσκηση 4 — Εναλλακτικοί τρόποι περιγραφής

4.1 Άσκηση 4 — Εναλλακτικοί τρόποι περιγραφής συνδυαστικών κυκλωμάτων

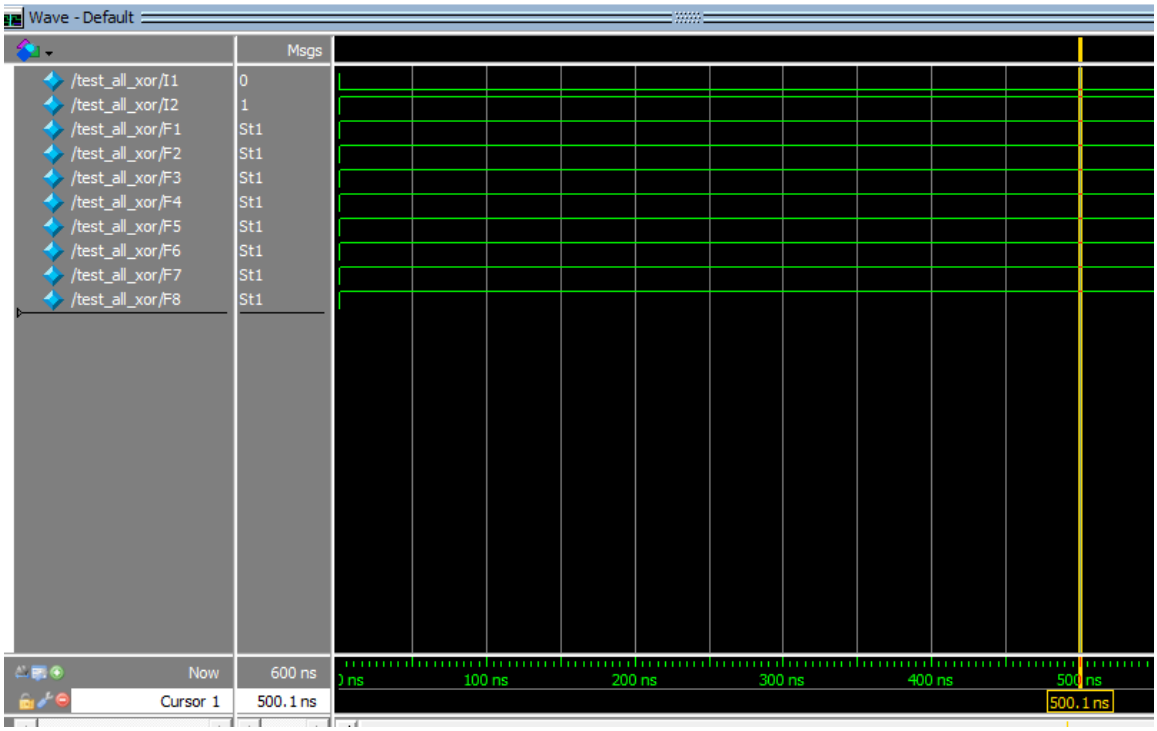
Σκοπός της άσκησης είναι η κατανόηση των διαφορετικών επιπέδων αφαίρεσης που υποστηρίζει η Verilog για την περιγραφή συνδυαστικών κυκλωμάτων:

- Δομική περιγραφή (structural)
- Περιγραφή με λογικές εξισώσεις (dataflow)
- Περιγραφή βάσει συμπεριφοράς (behavioral)

4.1.1 Παραδοτέο 1 — Πύλη XOR

Υλοποιήθηκαν πολλαπλές εκδόσεις της πύλης XOR χρησιμοποιώντας διαφορετικά επίπεδα αφαίρεσης.

Στο testbench εφαρμόστηκαν κοινά διανύσματα εισόδου στις μεταβλητές I_1 και I_2 και παρατηρήθηκαν οι έξοδοι F_1 έως F_8.



Σχήμα 4.1: Κυματομορφές εξομοίωσης όλων των εκδόσεων της πύλης XOR

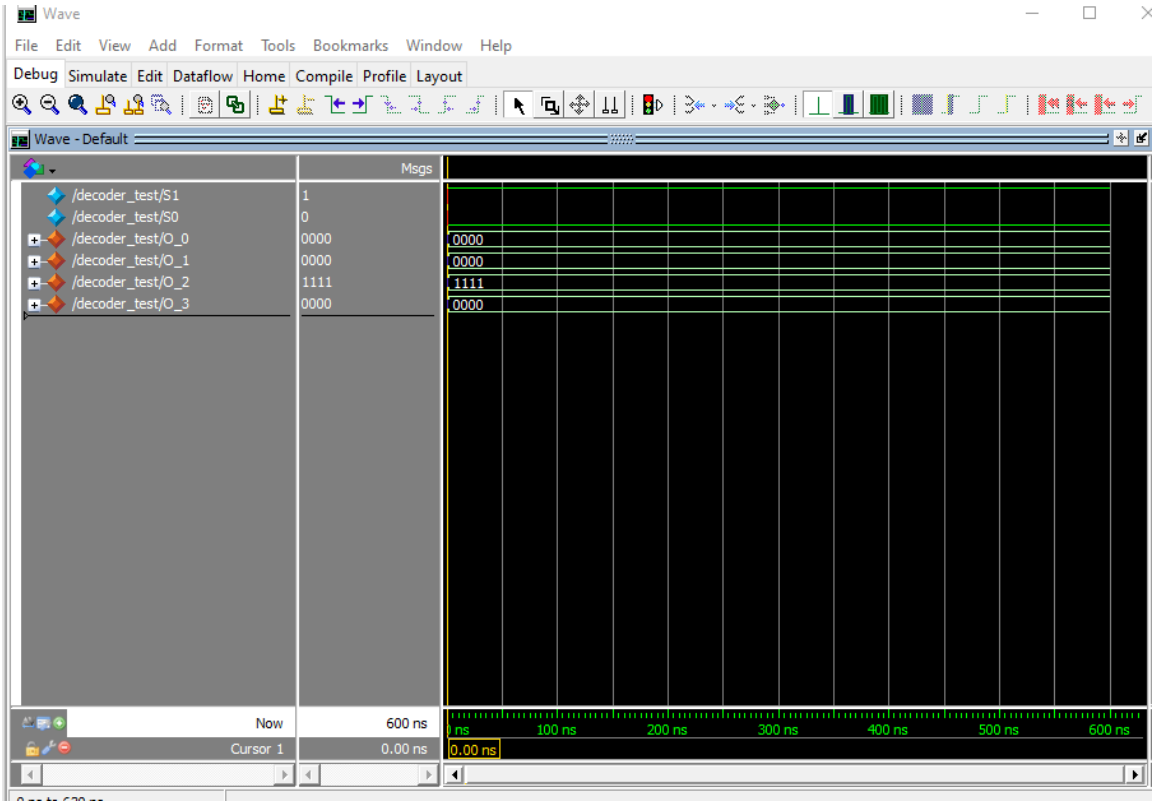
Παρατηρείται ότι όλες οι έξοδοι είναι ταυτόσημες για κάθε συνδυασμό εισόδων, γεγονός που επιβεβαιώνει ότι οι διαφορετικές περιγραφές αντιστοιχούν σε ισοδύναμα κυκλώματα.

4.1.2 Παραδοτέο 2 — Αποκωδικοποιητής 2 σε 4

Υλοποιήθηκε αποκωδικοποιητής 2→4 με τέσσερις διαφορετικές προσεγγίσεις:

- Δομική περιγραφή
- Περιγραφή με λογικές εξισώσεις
- Περιγραφή με ternary operator
- Behavioral περιγραφή

Ο αποκωδικοποιητής ενεργοποιεί μία από τις τέσσερις εξόδους ανάλογα με τη δυαδική τιμή των εισόδων S_1 και S_0.



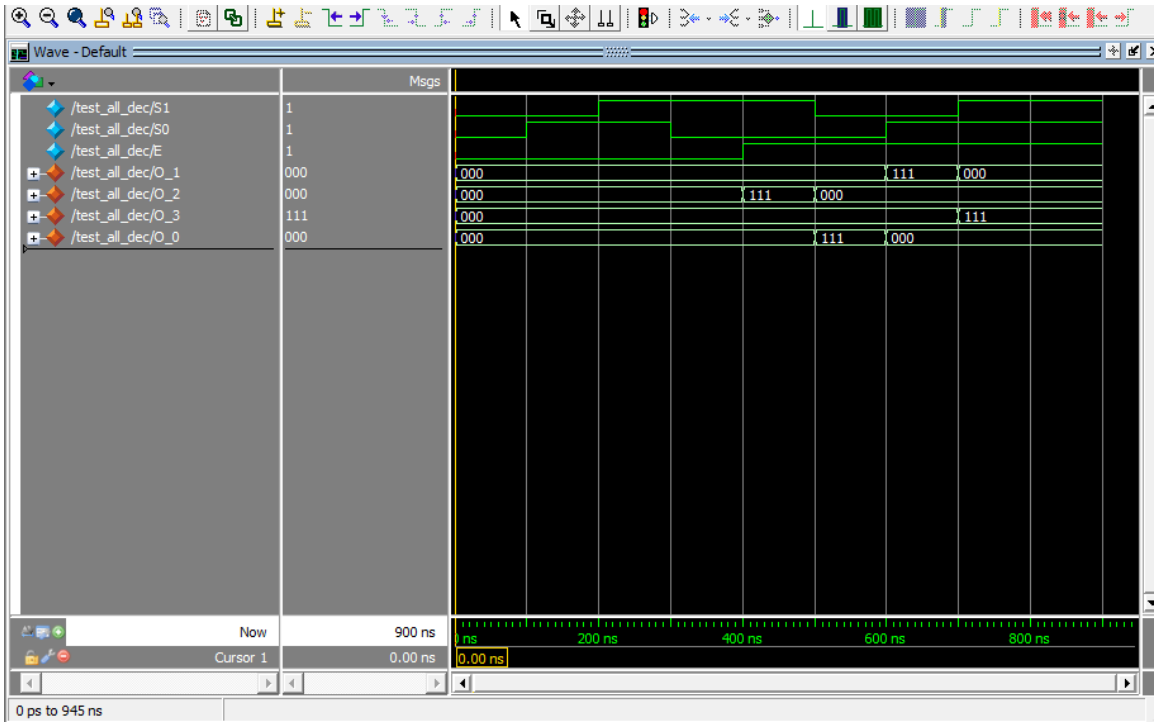
Σχήμα 4.2: Κυματομορφές εξομοίωσης αποκωδικοποιητή 2→4

Οι έξοδοι όλων των υλοποιήσεων συμπίπτουν, αποδεικνύοντας την ισοδυναμία των διαφορετικών τρόπων περιγραφής.

4.1.3 Παραδοτέο 3 — Αποπλέκτης 2 σε 4 με είσοδο επίτρεψης

Υλοποιήθηκε αποπλέκτης 2→4 με είσοδο επίτρεψης E.
Η λειτουργία του είναι:

- Αν $E = 0$ τότε όλες οι έξοδοι είναι μηδενικές
- Αν $E = 1$ τότε ενεργοποιείται μία έξοδος ανάλογα με S_1, S_0



Σχήμα 4.3: Κυματομορφές εξομοίωσης αποπλέκτη 2→4 με είσοδο επίτρεψης

Η εξομοίωση επιβεβαιώνει την ορθή λειτουργία του κυκλώματος.

4.1.4 Συμπέρασμα

Η άσκηση δείχνει ότι το ίδιο λογικό κύκλωμα μπορεί να περιγραφεί σε διαφορετικά επίπεδα αφαίρεσης στη Verilog. Οι περιγραφές υψηλού επιπέδου (dataflow και behavioral) είναι συνήθως πιο σύντομες, ευανάγνωστες και αποδοτικές για μεγάλα συστήματα, ενώ η δομική περιγραφή χρησιμοποιείται όταν απαιτείται συγκεκριμένη υλοποίηση.

Κεφάλαιο 5

Άσκηση 5 — Ιεραρχικός σχεδιασμός

5.1 Άσκηση 5 — Ιεραρχικός σχεδιασμός και επαλήθευση ισοδυναμίας

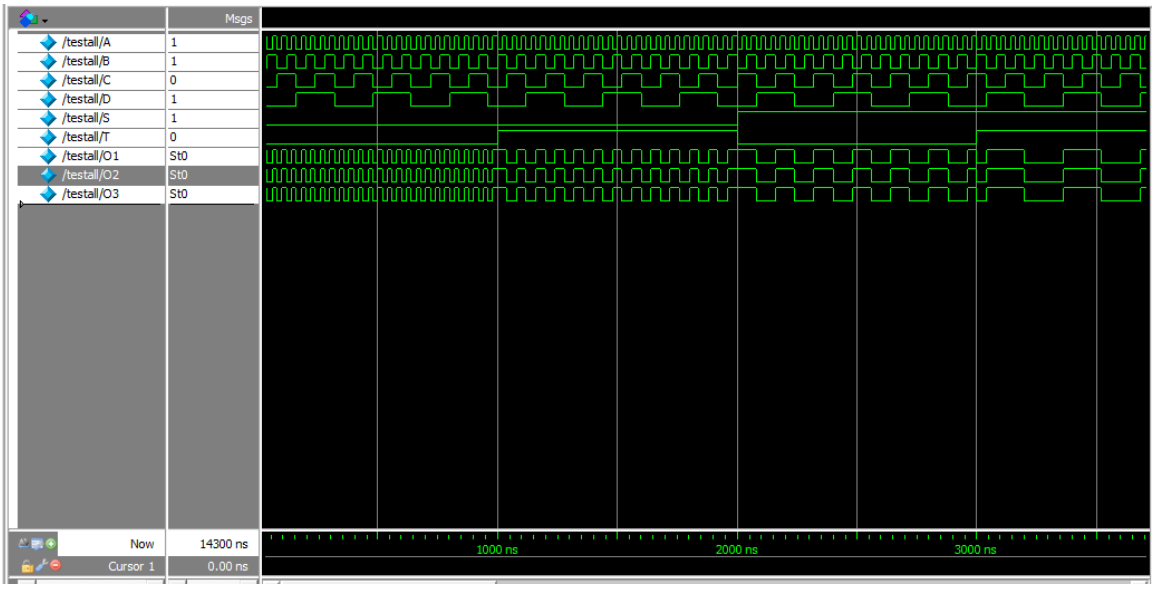
Η πέμπτη άσκηση εστιάζει στην έννοια της ιεραρχίας στη Verilog, δηλαδή στη δημιουργία σχεδιασμών που αποτελούνται από υποσχεδιασμούς (modules) πολλαπλών επιπέδων. Η προσέγγιση αυτή επιτρέπει την κατάτμηση της πολυπλοκότητας και την επαναχρησιμοποίηση κώδικα.

5.1.1 Παραδοτέο 1 — Πολυπλέκτης 4→1 σε διαφορετικές περιγραφές

Υλοποιήθηκε πολυπλέκτης 4→1 με τρεις διαφορετικές προσεγγίσεις: (α) δομική περιγραφή με πύλες, (β) dataflow με εξισώσεις και (γ) behavioral περιγραφή με nested ternary operator. Η λειτουργία του πολυπλέκτη δίνεται από τη σχέση:

$$O = \overline{S}\,\overline{T}A + \overline{S}TB + S\overline{T}C + STD$$

Στο testbench εφαρμόστηκαν περιοδικά σήματα στις εισόδους δεδομένων και τα σήματα επιλογής S, T άλλαζαν ανά 2000 ns, ώστε να επιλέγεται διαδοχικά κάθε πηγή.

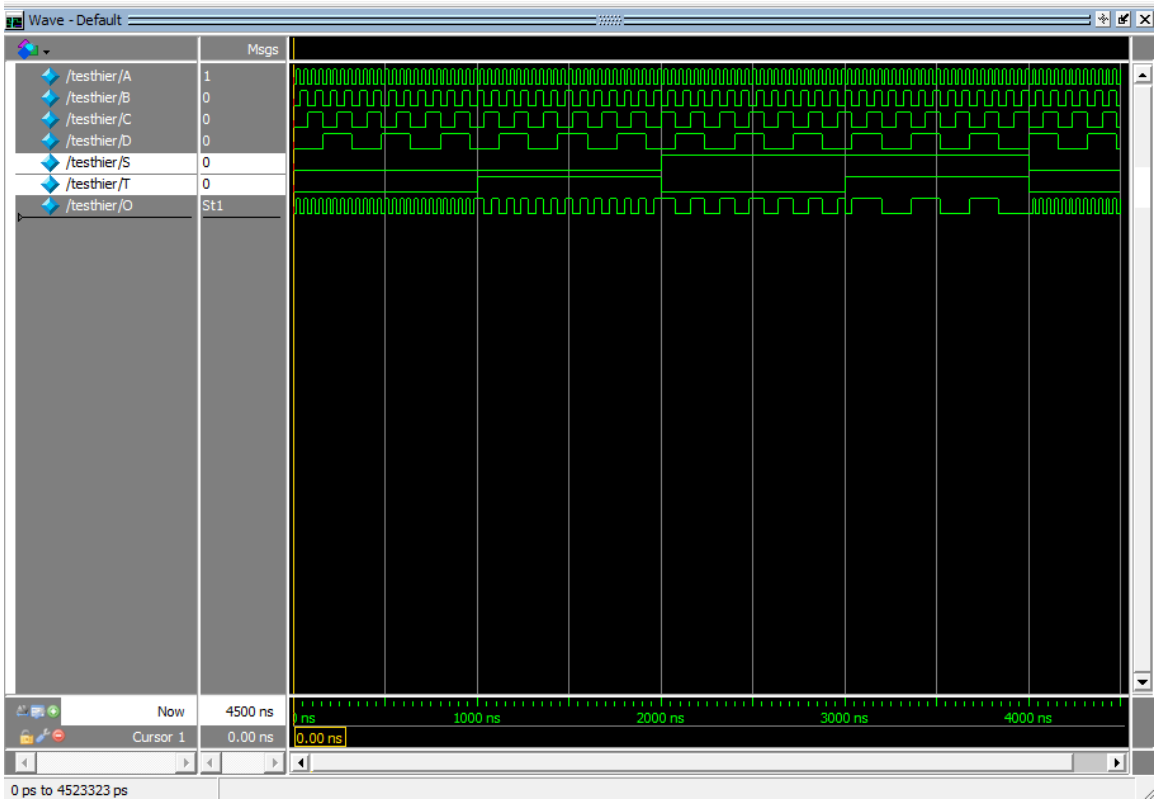


Σχήμα 5.1: Παραδοτέο 1: Σύγκριση εξόδων O1, O2, O3 (structural/equations/behavioral)

Παρατηρείται ότι οι έξοδοι των τριών υλοποιήσεων συμπίπτουν για όλα τα χρονικά διαστήματα, επιβεβαιώνοντας τη λειτουργική ισοδυναμία.

5.1.2 Παραδοτέο 2 — Ιεραρχικός πολυπλέκτης 4→1 με decoder

Στο δεύτερο παραδοτέο υλοποιήθηκε πολυπλέκτης 4→1 μέσω ιεραρχίας δύο επιπέδων: ο αποκωδικοποιητής 2→4 (decoder2_2_4) παράγει τα σήματα επιλογής και στη συνέχεια αυτά "μάσκουν" τις εισόδους μέσω AND και OR πυλών.



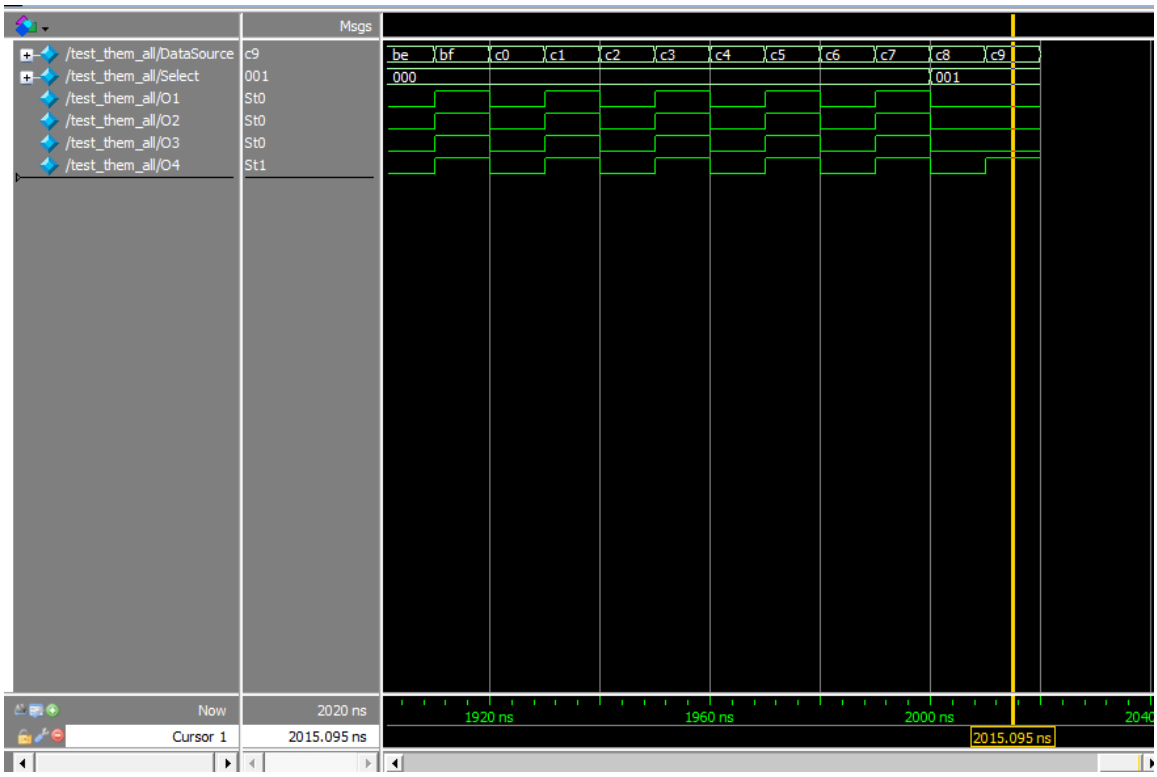
Σχήμα 5.2: Παραδοτέο 2: Εξομοίωση ιεραρχικού πολυπλέκτη 4→1 (testhier)

Η εξομοίωση επιβεβαιώνει ότι η έξοδος ακολουθεί τη σωστή πηγή δεδομένων σύμφωνα με τα σήματα επιλογής.

5.1.3 Παραδοτέο 3 — Πολυπλέκτης 8→1 και εντοπισμός σφάλματος

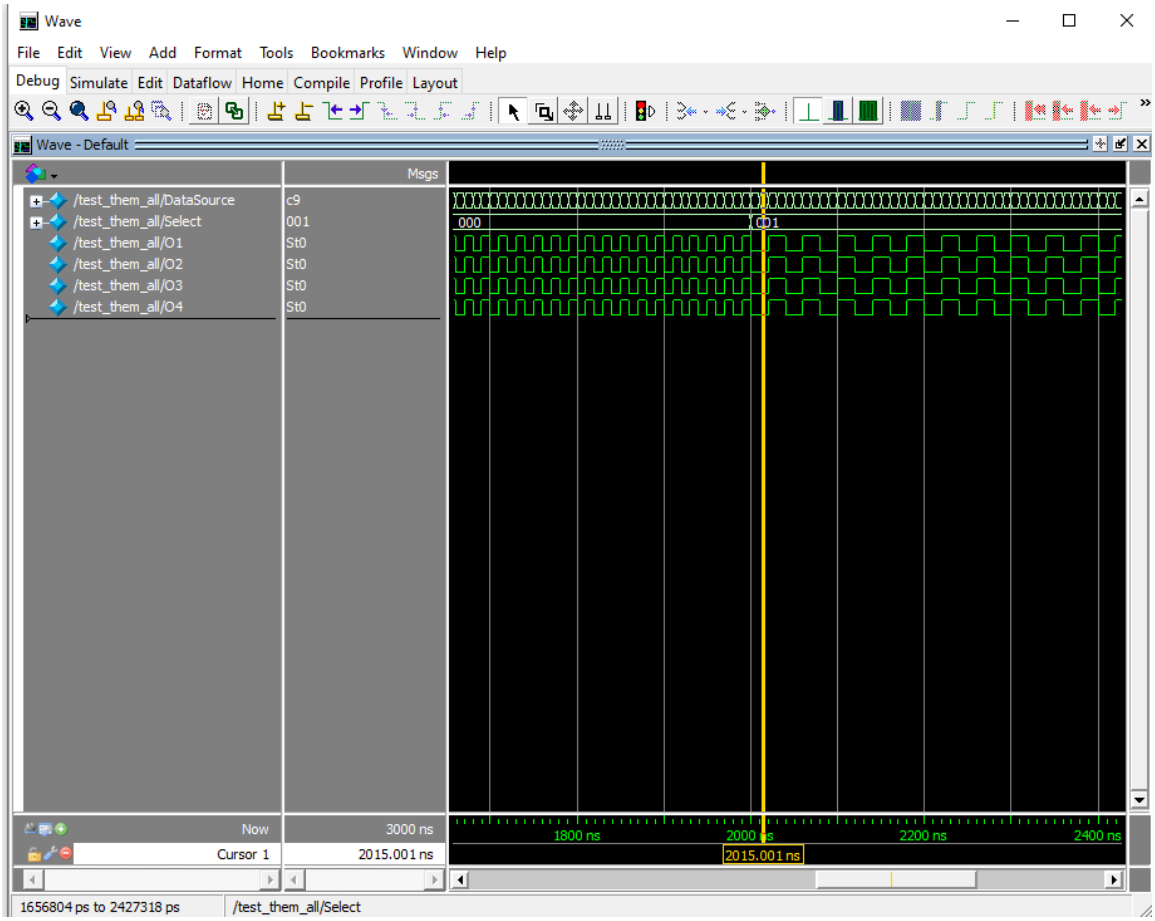
Υλοποιήθηκε πολυπλέκτης 8→1 σε τέσσερις διαφορετικές μορφές (CUT1–CUT4) και δημιουργήθηκε testbench που συγκρίνει τις εξόδους τους. Η εξομοίωση σταματά με $stop() όταν υπάρξει ασυμφωνία.

Στο αρχικό (buggy) αρχείο παρατηρήθηκε πρώτη ασυμφωνία γύρω στη χρονική στιγμή 2015 ns, όπου η έξοδος του CUT4 (O4) διέφερε από τις υπόλοιπες.



Σχήμα 5.3: Παραδοτέο 3: Εντοπισμός ασυμφωνίας (bug) — διαφορά στην έξοδο O4

Με χρήση ιεραρχικής πλοήγησης στο ModelSim (structure/signals) εντοπίστηκε το λάθος στο υποσύστημα του CUT4, διορθώθηκε ο κώδικας και επαναλήφθηκε η εξομοίωση.



Σχήμα 5.4: Παραδοτέο 3: Μετά τη διόρθωση, οι έξοδοι O1–O4 συμπίπτουν (2000–7000 ns)

5.1.4 Συμπέρασμα

Η άσκηση ανέδειξε τα πλεονεκτήματα του ιεραρχικού σχεδιασμού (επαναχρησιμοποίηση και ευκολότερο debugging) και έδειξε μια πρακτική μεθοδολογία επαλήθευσης ισοδυναμίας πολλαπλών υλοποιήσεων μέσω testbench.

Κεφάλαιο 6

Άσκηση 6 — Συγκριτές και καθυστερήσεις

6.1 Άσκηση 6 — Συγκριτές δυαδικών αριθμών και καθυστερήσεις

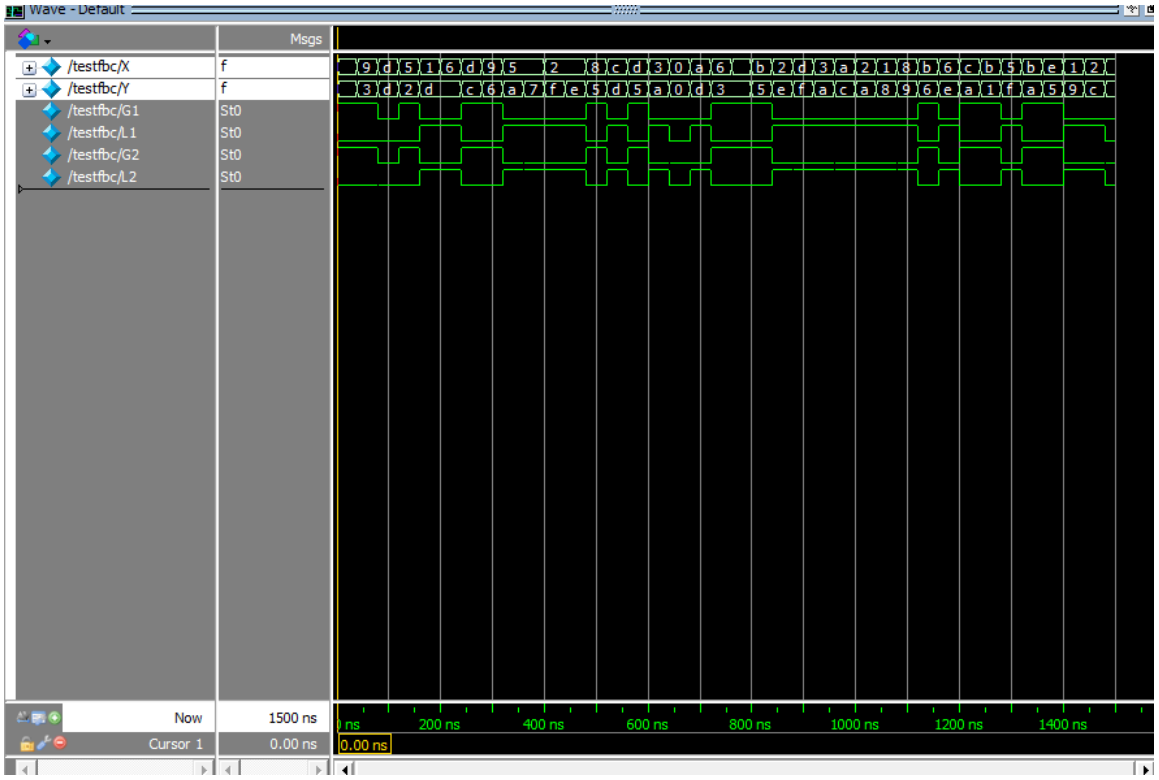
Σκοπός της άσκησης είναι η μελέτη της χρήσης αρτηριών στη Verilog και η εκτίμηση της χρονικής καθυστέρησης συνδυαστικών κυκλωμάτων, μέσω σύγκρισης διαφορετικών αρχιτεκτονικών συγκριτών.

6.1.1 Παραδοτέο 1 — Σύγκριση behavioral και σειριακού συγκριτή

Υλοποιήθηκαν δύο συγκριτές 4 δυαδικών ψηφίων:

- Behavioral συγκριτής
- Σειριακός συγκριτής βασισμένος σε κύτταρα 2 bit

Το testbench εφαρμόζει τυχαίες τιμές στις αρτηρίες εισόδου X και Y.



Σχήμα 6.1: Παραδοτέο 1: Σύγκριση εξόδων των δύο συγκριτών

Οι έξοδοι των δύο υλοποιήσεων συμπίπτουν πλήρως, γεγονός που επιβεβαιώνει τη λειτουργική ισοδυναμία τους.

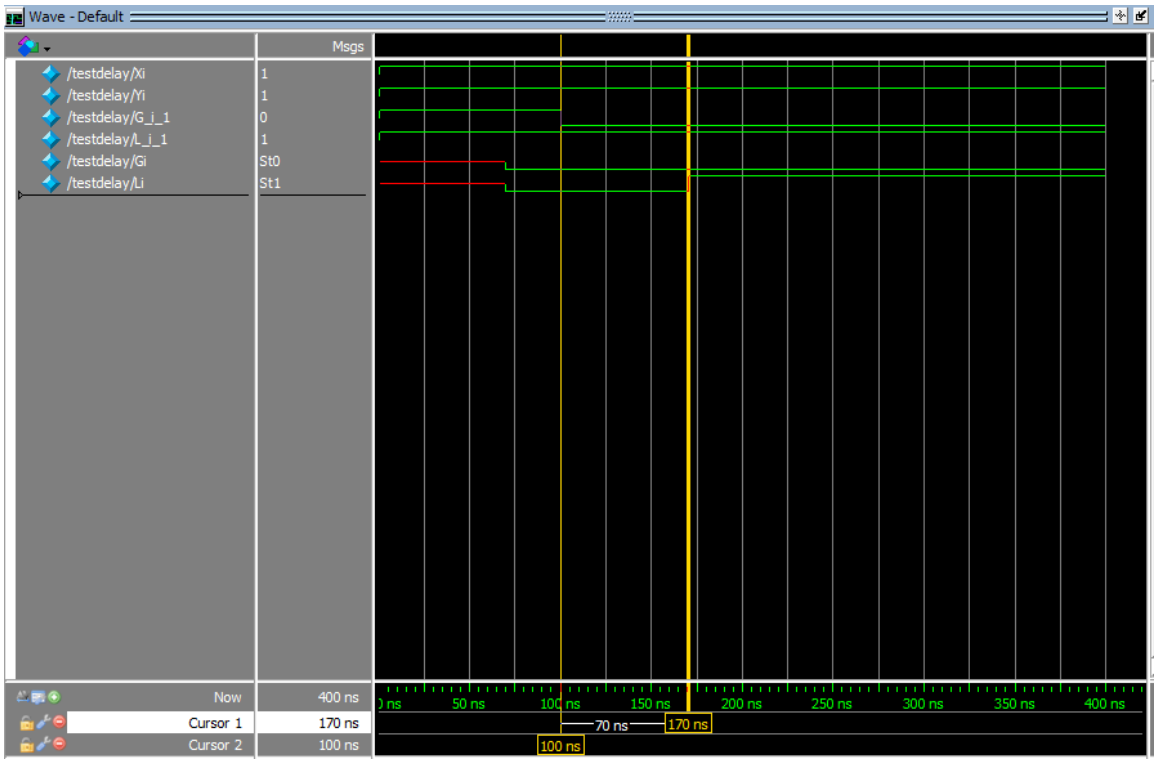
6.1.2 Παραδοτέο 2 — Μέτρηση καθυστέρησης συγκριτή 2 bit

Ο συγκριτής 2 bit υλοποιήθηκε με πύλες συγκεκριμένων καθυστερήσεων:

- Αντιστροφέας: 10 μονάδες
- Πύλη AND 2 εισόδων: 20 μονάδες
- Πύλη AND 3 εισόδων: 30 μονάδες
- Πύλη OR 3 εισόδων: 30 μονάδες

Το χειρότερο μονοπάτι διάδοσης περιλαμβάνει έναν αντιστροφέα, μια AND 3 εισόδων και μια OR 3 εισόδων, άρα η συνολική καθυστέρηση είναι:

$$T = 10 + 30 + 30 = 70$$



Σχήμα 6.2: Παραδοτέο 2: Μέτρηση καθυστέρησης με cursors

Η εξομοίωση επιβεβαιώνει τη θεωρητική τιμή.

6.1.3 Παραδοτέο 3 — Σειριακός και παράλληλος συγκριτής 8 bit

Υλοποιήθηκαν δύο αρχιτεκτονικές:

Σειριακός συγκριτής Αποτελείται από αλυσίδα συγκριτών 2 bit. Η καθυστέρηση αυξάνεται γραμμικά με το μήκος:

$$T_{serial} = 70 + (n-2) \cdot 60$$

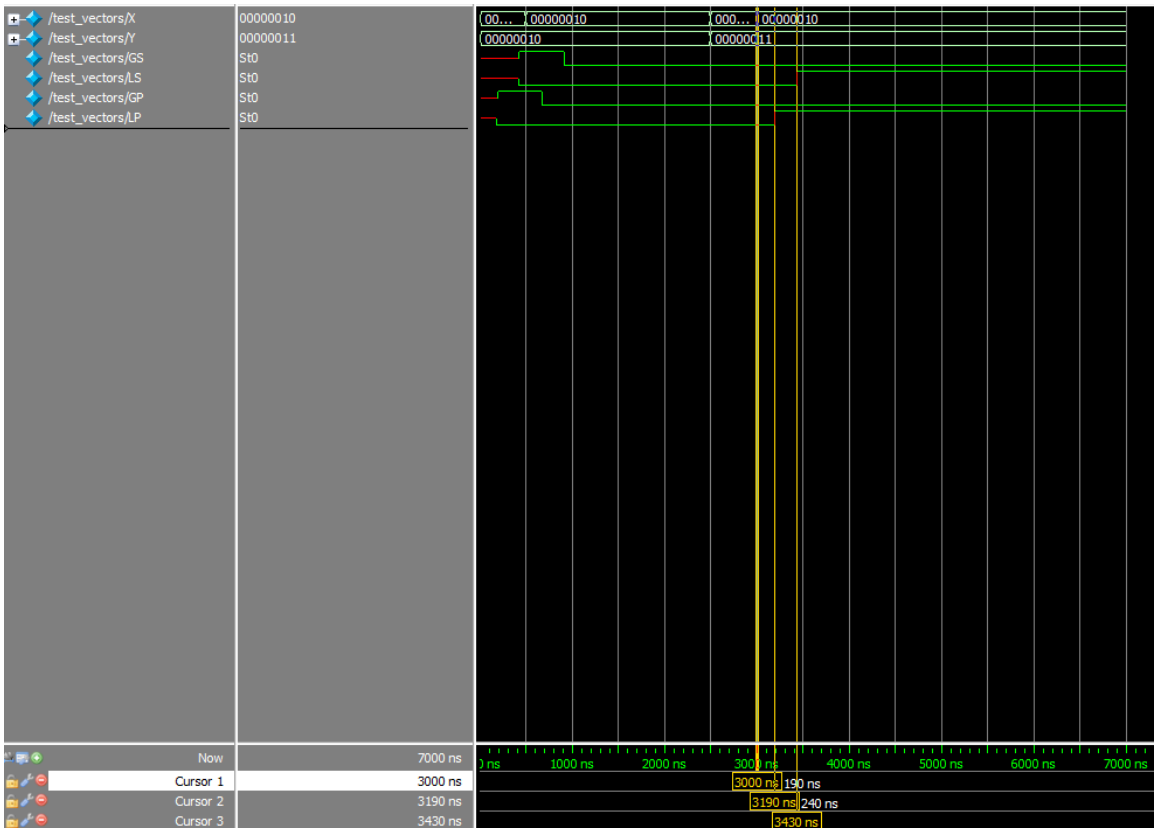
Για 8 bit προκύπτει:

$$T_{serial} = 430$$

Παράλληλος συγκριτής Βασίζεται σε δενδρική δομή πολλών επιπέδων, όπου τα συγκριτικά αποτελέσματα υπολογίζονται ταυτόχρονα.

Για 8 bit η καθυστέρηση είναι:

$$T_{parallel} = 70 + 2 \cdot 60 = 190$$



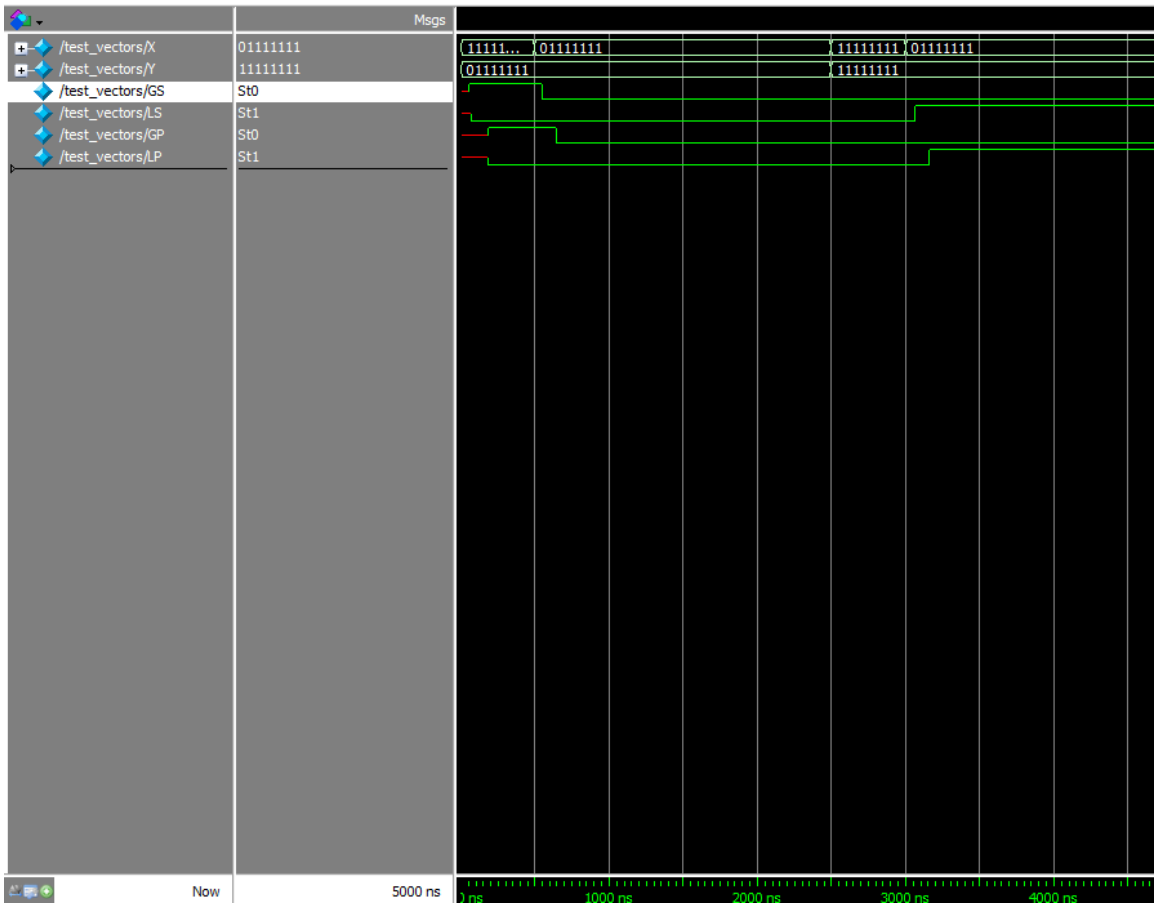
Σχήμα 6.3: Παράλληλος συγκριτής — μικρότερη καθυστέρηση

Παρατηρείται ότι η παράλληλη αρχιτεκτονική είναι σημαντικά ταχύτερη.

6.1.4 Περίπτωση ταχύτερου σειριακού αποτελέσματος

Σε ορισμένα διανύσματα εισόδου το αποτέλεσμα μπορεί να προκύψει πριν διαδοθεί η μετάβαση σε όλο το κύκλωμα, οπότε ο σειριακός συγκριτής εμφανίζεται ταχύτερος.

Η εξομοίωση με κατάλληλα επιλεγμένο διάνυσμα επιβεβαιώνει το φαινόμενο.



Σχήμα 6.4: Σειριακός συγκριτής — μικρότερη καθυστέρηση

6.1.5 Συμπέρασμα

Η άσκηση αναδεικνύει ότι η αρχιτεκτονική ενός κυκλώματος επηρεάζει σημαντικά την καθυστέρησή του. Η σειριακή υλοποίηση παρουσιάζει γραμμική αύξηση καθυστέρησης, ενώ η παράλληλη υλοποίηση οδηγεί σε λογαριθμική αύξηση και συνεπώς σε πολύ υψηλότερη απόδοση.

Κεφάλαιο 7

Άσκηση 7 — BCD Αριθμητική

7.1 Άσκηση 7 — BCD Αριθμητική

Σκοπός της άσκησης είναι η μελέτη της πρόσθεσης αριθμών σε κώδικα BCD και η εκτίμηση της χρονικής καθυστέρησης των αντίστοιχων κυκλωμάτων.

7.1.1 Πλήρης αθροιστής με καθυστέρηση

Υλοποιήθηκε πλήρης αθροιστής με καθυστέρηση 80 μονάδων για το άθροισμα και 45 μονάδων για το κρατούμενο.

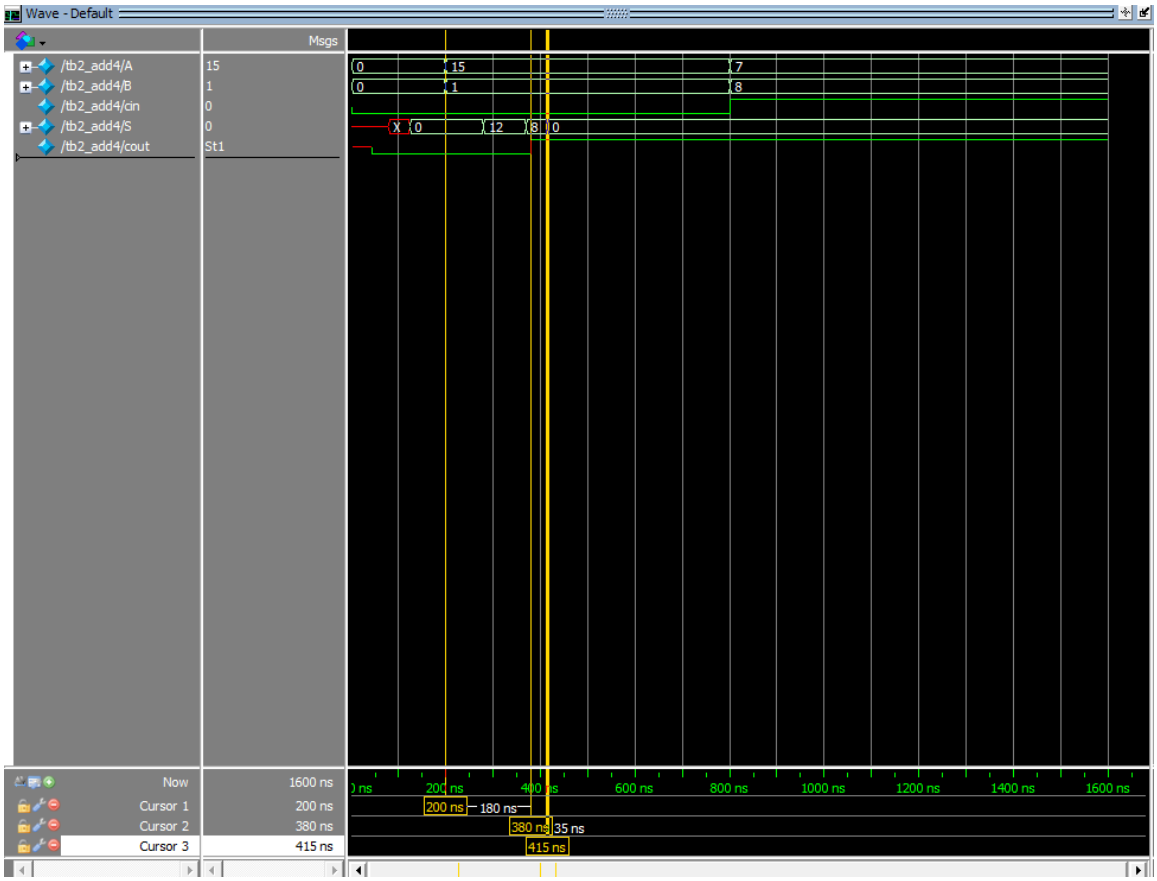
7.1.2 Ripple-carry adder 4 bit

Ο αθροιστής αποτελείται από τέσσερις πλήρεις αθροιστές σε σειρά. Η χειρότερη καθυστέρηση για το κρατούμενο είναι:

$$t_{cout} = 4 \cdot 45 = 180$$

Για το πιο σημαντικό bit του αθροίσματος:

$$t_{S3} \approx 215$$



Σχήμα 7.1: Ripple Carry Adder Simulation

7.1.3 BCD Full Adder

Ο BCD αθροιστής εκτελεί:

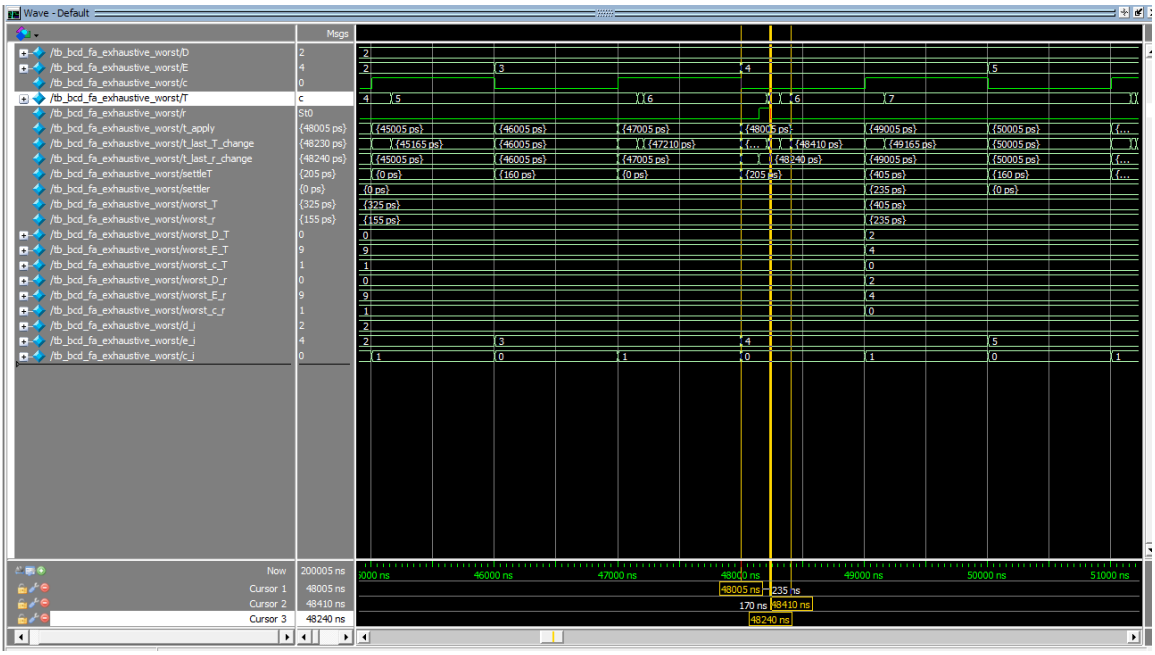
1. Δυαδική πρόσθεση D + E + c
2. Έλεγχο υπερχείλισης (αποτέλεσμα > 9)
3. Διόρθωση +6 όταν απαιτείται

Το κρατούμενο εξόδου προκύπτει από τη συνθήκη:

$$r = cout \vee (s_3 \wedge (s_2 \vee s_1))$$

Η χειρότερη καθυστέρηση του r είναι περίπου 235 μονάδες.
Η καθυστέρηση παραγωγής του διορθωμένου αποτελέσματος T είναι:

$$T = 405$$



Σχήμα 7.2: BCD Full Adder Simulation

7.1.4 BCD παράλληλος αθροιστής 3 ψηφίων

Ο αθροιστής αποτελείται από τρία κύτταρα BCD FA σε διάταξη διάδοσης κρατουμένου. Η καθυστέρηση του τελικού κρατουμένου είναι:

$$w = 3 \cdot 235 = 705$$

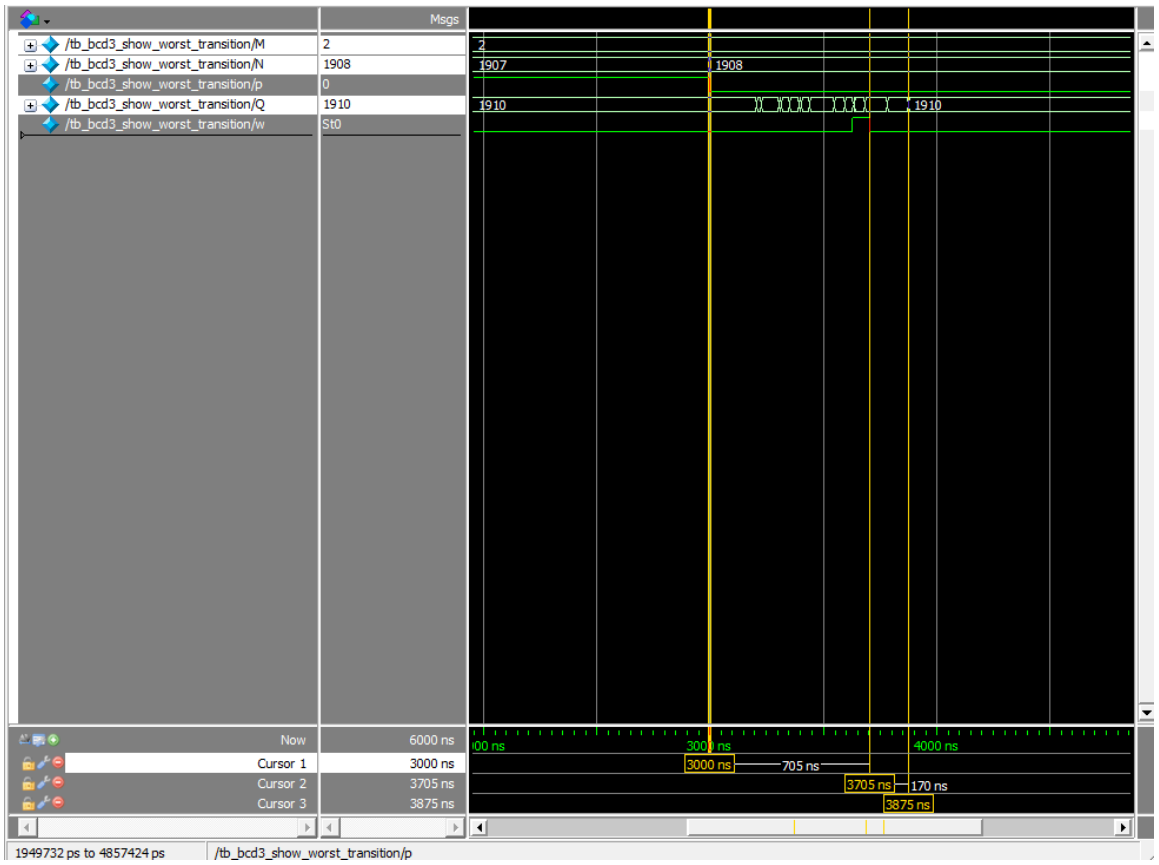
Για το πιο σημαντικό ψηφίο του αποτελέσματος:

$$Q_{MSD} = 2 \cdot 235 + 405 = 875$$

7.1.5 Πειραματική επιβεβαίωση

Με χρήση exhaustive testbench βρέθηκε ότι η χειρότερη καθυστέρηση είναι:

$$T_{worst} = 875$$



Σχήμα 7.3: BCD παράλληλος αθροιστής 3 ψηφίων Simulation

που συμφωνεί με τον θεωρητικό υπολογισμό.

7.1.6 Συμπέρασμα

Η BCD πρόσθεση απαιτεί σημαντικά μεγαλύτερη καθυστέρηση από τη δυαδική, λόγω του επιπλέον ελέγχου υπερχείλισης και της διόρθωσης κατά 6. Η διάταξη ripple carry οδηγεί σε γραμμική αύξηση της καθυστέρησης με τον αριθμό των ψηφίων.

Συμπεράσματα

Οι ασκήσεις παρείχαν εισαγωγή στη Verilog και στις βασικές τεχνικές σχεδίασης ψηφιακών κυκλωμάτων, από απλές λογικές πύλες έως σύνθετα ιεραρχικά συστήματα με ανάλυση καθυστερήσεων.

Η εξομοίωση αποτέλεσε βασικό εργαλείο επαλήθευσης της ορθότητας και της απόδοσης των κυκλωμάτων.