2η Εργαστηριακή Άσκηση

Κοτσιµπού Φωτεινή ΑΜ 1059567 Σιγούρου ΄Αλκηστις-Αικατερίνη ΑΜ 1059661

2 Απριλίου 2021



Εργασία στα πλαίσια του µαθήµατος Σχεδιασμός Συστημάτων VLSI

Τμήμα Μηχ. Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Περιεχόμενα

1. [Ενότητα Α](#_bookmark0) 1
   1. [Α1 - Ανάπτυξη κώδικα Πολυπλέκτη](#_bookmark1) 1
   2. [Α2 - Μεταγλώττιση και προσομοίωση Πολυπλέκτη](#_bookmark2) 1
2. [Ενότητα Β](#_bookmark8) 5
   1. [Β1 - Υλοποίηση της Ζ συνάρτησης](#_bookmark9) 5
   2. [Β2-Προσεγγίσεις κώδικα](#_bookmark11) 5
      1. [Β2α-Προσέγγιση µε βάση διεργασία](#_bookmark12) 5
      2. [Β2β-Προσέγγιση µε συνδυαστικό κύκλωμά](#_bookmark13) 5
   3. [Β3- Μεταγλώττιση προσοµοίωση](#_bookmark17) 8
   4. [Β4-Σύγκριση προσεγγίσεων](#_bookmark20) 9

# Ενότητα Α

## Α1 - Ανάπτυξη κώδικα Πολυπλέκτη

Για την εκπλήρωση του ερωτήματος Α1 υλοποιήσαμε έναν πολυπλέκτη µε δύο διαφορετικούς κώδικες. Αρχικά, θέσαμε όλες µας τις μεταβλητές στην αρχή του ENTITY ως std logic vector,8 ψηφιών, για της δύο εισόδους a, b και την έξοδο x ενώ για το σήμα ελέγχου sel προτιμήσαμε έναν std logic vector, 2 ψηφίων. Η διαφορά στους δύο κώδικες βρίσκεται στις βασικές εντολές που υπάρχουν στο architecture.

Στην πρώτη εναλλακτική, στο Σχήμα [1,](#_bookmark4) δημιουργήσαμε µία διεργασία και κάναμε χρήση ενός βρόγχου elseif για κάθε πιθανή κατάσταση του πολυπλέκτη που µας είχε δοθεί από την εκφώνηση. Στην προσπάθεια µας να σιγουρευτούμε για την ορθότητα του κώδικα, επιλέξαμε να προσθέσουμε ακόμη µία κατάσταση η οποία θα ελέγχει όλες τις υπόλοιπες ενδιάμεσες καταστάσεις. Ο λόγος που το κάνουμε αυτό είναι γιατί χρησιμοποιούμε κατά την δήλωση των μεταβλητών, std logic vector,, ένας τύπος µε 9 δυνατές καταστάσεις γεγονός που καθιστά των κώδικά µας εκτεθειμένο να βρεθεί σε µία από τις ¨ανεπιθύμητες’ καταστάσεις. Το πρόβλημα αυτό λύνουμε εισάγοντας ένα τελευταίο else που θα δίνει στην έξοδό µας τη τιμή ”xxxxxxxx”.

Στη δεύτερη εναλλακτική, στο Σχήμα [2,](#_bookmark5) κάνουμε χρήση, ενός use case statement µε όλες τις πιθανές καταστάσεις-cases στις οποίες µπορεί να βρεθεί ο πολυπλέκτης µας ακολουθώντας τη λογική που χρησιμοποιήσαμε και παραπάνω στην πρώτη υλοποίηση.

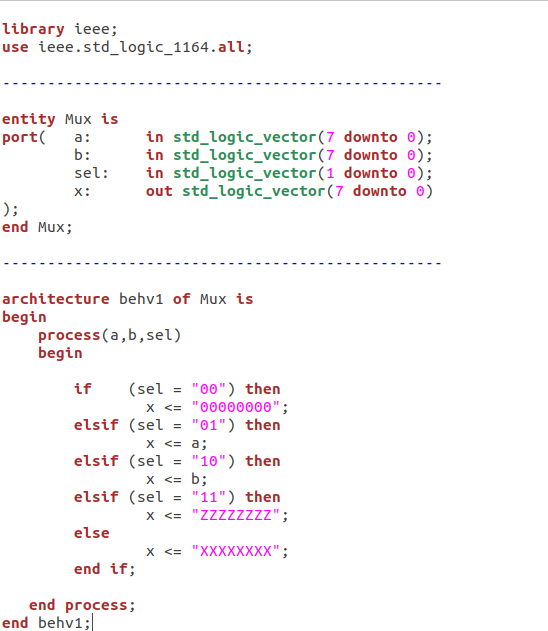
## Α2 - Μεταγλώττιση και προσοµοίωση Πολυπλέκτη

Για την προσομοίωση και των δύο εκδοχών χρησιμοποιήσαμε την εντολή force ώστε να εισάγουμε στον κάθε κύκλο τις κατάλληλες τιμές στα inputs που βρίσκουμε και στον πίνακα αληθείας [1.](#_bookmark3)

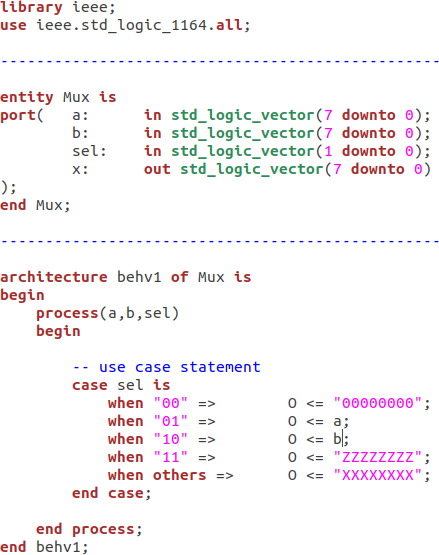
|  |  |
| --- | --- |
| sel | x |
| ”00” | ”00000000” |
| ”01” | ”a” |
| ”10” | ”b” |
| ”11” | ”ZZZZZZZ” |

Πίνακας 1: Πίνακας Αληθείας Πολυπλέκτης

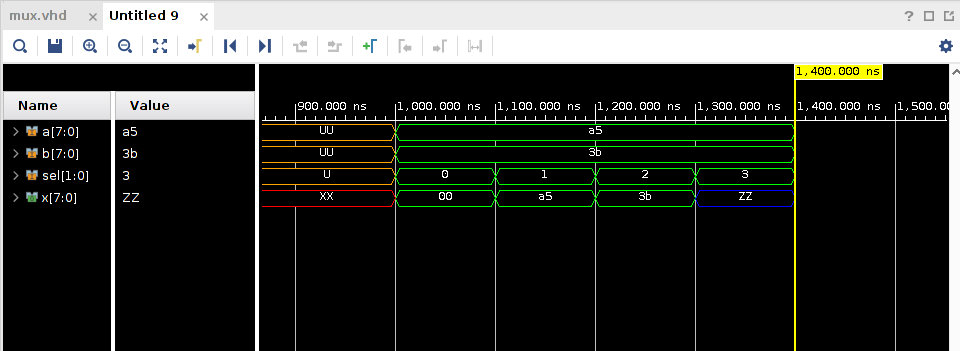
Τα αποτελέσματα για τις 2 εκδοχές φαίνονται στα Σχήματα [3](#_bookmark6) και [4](#_bookmark7) και παρατηρείται ότι παίρνουμε τα σωστά αποτελέσματα και στις 2 εξόδους.



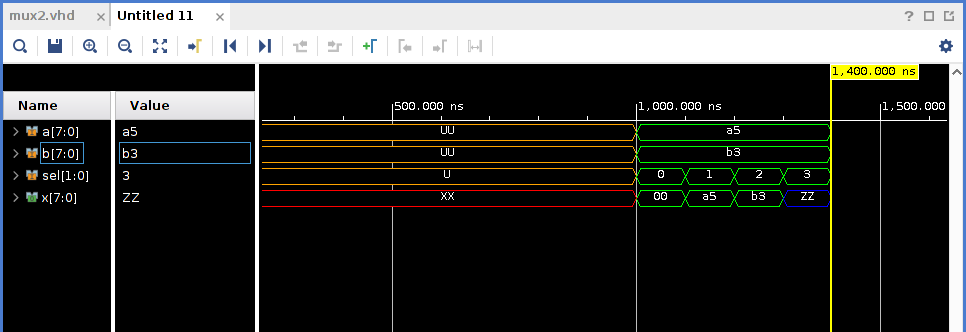
Σχήµα 1: Multiplexer v.1



Σχήµα 2: Multiplexer v.2



Σχήµα 3: Multiplexer v.1 results



Σχήµα 4: Multiplexer v.2 results

# Ενότητα Β

## Β1 - Υλοποίηση της Ζ συνάρτησης

Στο ερώτημα αυτό, κληθήκαμε να υλοποιήσουμε την συνάρτηση Z= ABC’ + AB + C και να την προσομοιώσουμε, ώστε να καταλήξουμε στα αποτελέσματα του Πίνακα Αληθείας [2.](#_bookmark10)

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | Z |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Πίνακας 2: Πίνακας Αληθείας Ζ συνάρτησης

## Β2-Προσεγγίσεις κώδικα

Χρειάστηκε να προσεγγίσουμε µε 2 διαφορετικές τεχνικές. Η πρώτη ήταν προσέγγιση µε βάση διεργασία και η δεύτερη µε structural αρχιτεκτονική.

* + 1. Β2α-Προσέγγιση µ.ε βάση διεργασία

Στην πρώτη εκδοχή, δουλέψαμε κάνοντας χρήση διεργασίας που αποτελεί bevavioral υλοποίηση. Στη διεργασία αυτή δηλώσαμε τα σήματα εισόδου µας και στην συνέχεια συµβουλευόµενες τον πίνακα αληθείας που κατασκευάσαμε για το δοθέν κύκλωμά, εντοπίσαμε τις τιμές των μεταβλητών εισόδου για τις οποίες η έξοδός µας γίνεται µονάδα. Τις καταστάσεις αυτές τις συµπεριλάβαµε σε έναν elseif βρόγχο ο οποίος δίνει την τιμή ΄1΄ στην έξοδό µας, Ζ, για όλες τις περιπτώσεις που προκύπτουν από τον πίνακα αληθείας που θα βρείτε παραπάνω. Στις εισόδους και εξόδους του κυκλώματος έχουμε δώσει τον τύπο µεταβλητης std logic ο οποίος δίνει στις µμεταβλητές µας µόνο τις λογικές τιμές ¨0’ και

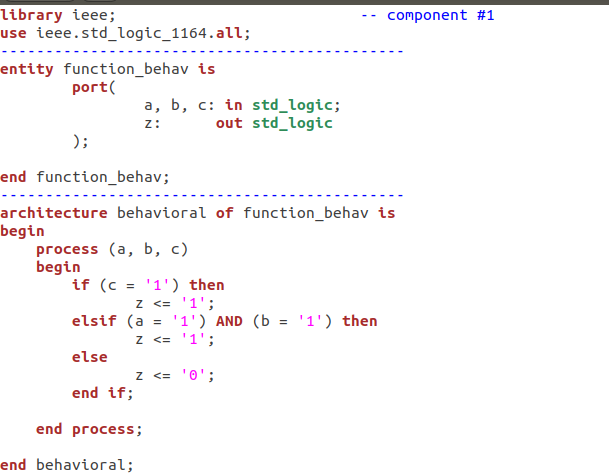
¨1’ και µας εξασφαλίζει πως το κύκλωμά µας δεν θα εισέλθει σε καμία ανεπιθύμητη περίπτωση. Για τον λόγο αυτό δεν κάνουμε χρήση τους προστατευτικού else στο τέλος, όπως κάναμε στο ερώτημα της ενότητας Α.

Ο κώδικας παρατίθεται στο Σχήμα [9](#_bookmark18)

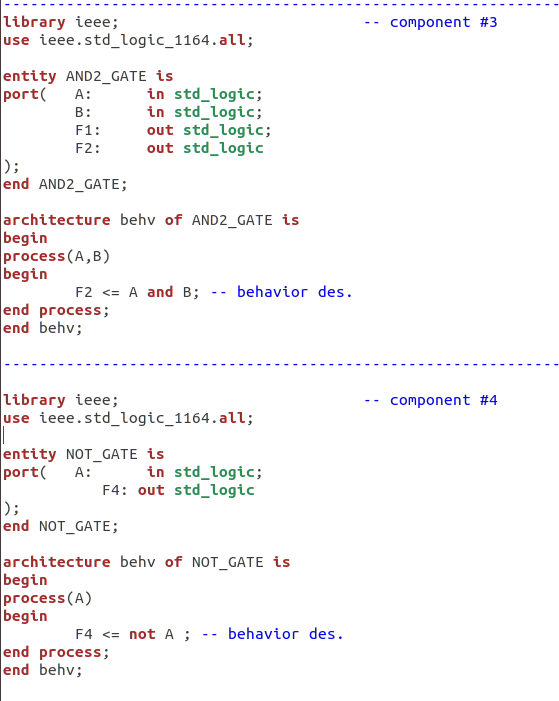
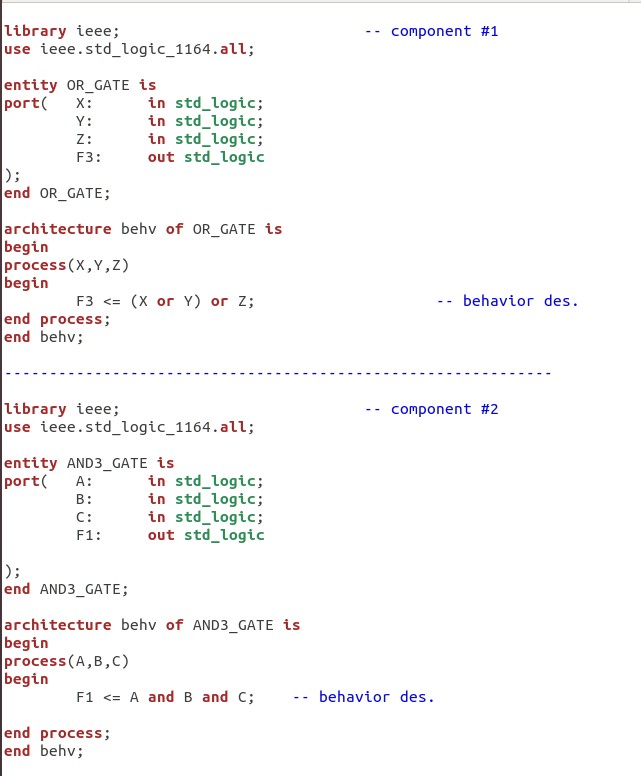
* + 1. Β2β-Προσέγγιση µ.ε συνδυαστικό κύκλωμά

Στην δεύτερη εκδοχή, δουλέψαμε µε components ή αλλιώς structural αρχιτεκτονική. Δημιουργήσαμε 4 πύλες µε την εντολή ENTITY ( NOT, AND-2, AND-3, OR-3) και συνθέσαμε την συνάρτηση Ζ ως καινούργια οντότητα µε αυτές χρησιμοποιώντας ως components τις πύλες που δημιουργήσαμε. Στην συνέχεια προσθέσαμε 3 καλώδια µε την μορφή signals ώστε να µην χρειαστεί να επαναλαμβάνουμε ολόκληρη την πύλη στην σύνθεση που ακολουθεί. Τέλος ορίζουμε ποιο σήμα έχουν για input και output οι πύλες µας, ώστε να µμπορέσουμε να εξομοιώσουμε στην συνέχεια.

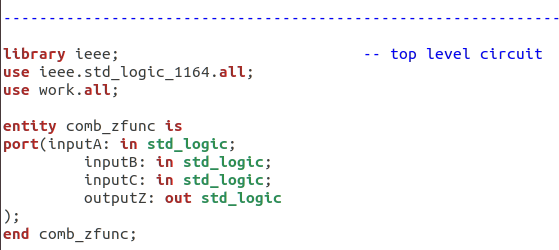
Ο κώδικας παρατίθεται στα Σχήματα [6,](#_bookmark14) [7](#_bookmark15) και [8](#_bookmark16)



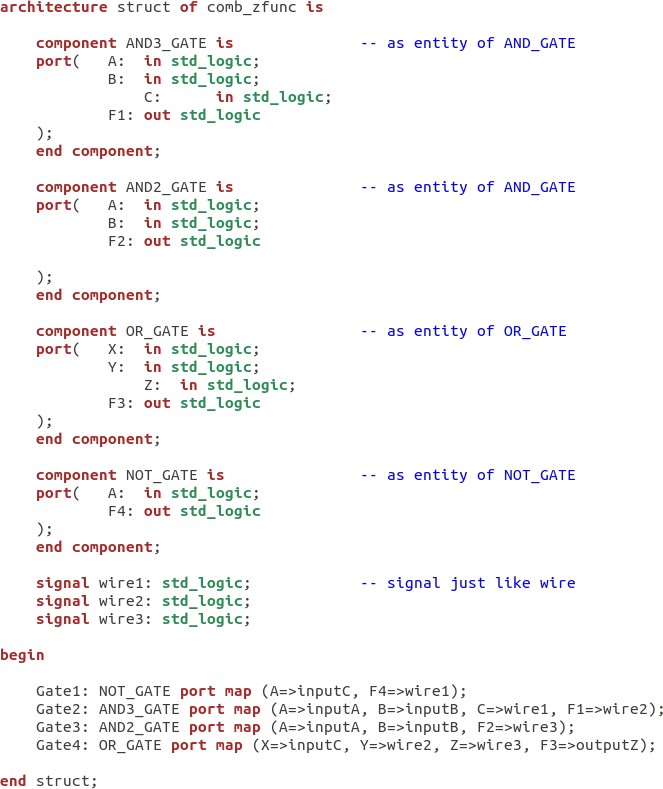
Σχήµα 5: Behavioral version of Z function - code



Σχήµα 6: Ορισµός πυλών - Structural Version



Σχήµα 7: Ορισµός Zfunc component - Structural Version

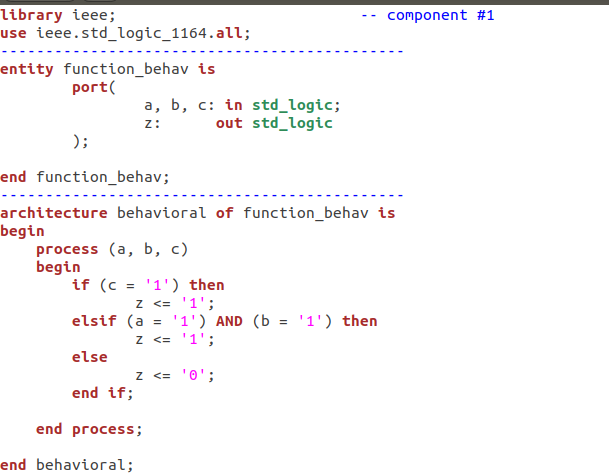


Σχήµα 8: Ορισµός Zfunc architecture - Structural Version

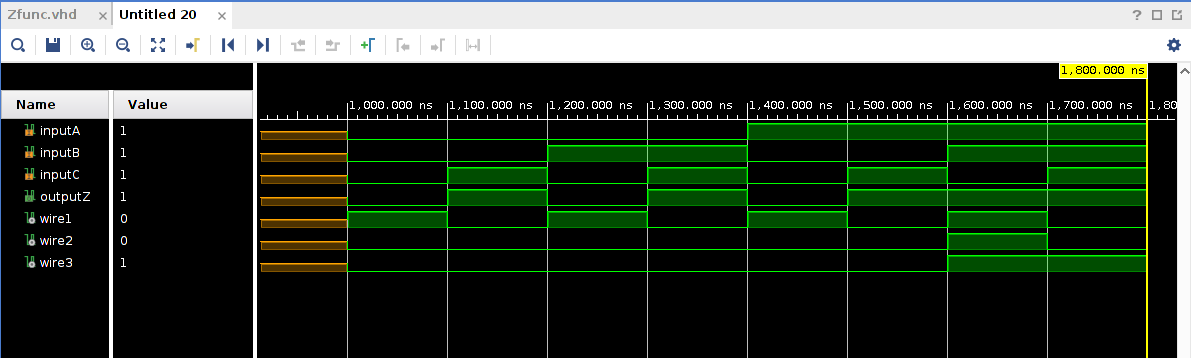
## Β3- Μεταγλώττιση προσοµοίωση

Για την προσομοίωση και των δύο εκδοχών χρησιμοποιήσαμε την εντολή force ώστε να εισάγουμε στον κάθε κύκλο τις κατάλληλες τιμές στα inputs που βρίσκουμε και στον πίνακα αληθείας [2.](#_bookmark10)

Τα αποτελέσματα για τις 2 εκδοχές φαίνονται στα Σχήματα [9](#_bookmark18) και [10](#_bookmark19) και παρατηρείται ότι παίρνουμε τα σωστά αποτελέσματα και στις 2 εξόδους.



Σχήµα 9: Behavioral version of Z function - code



Σχήµα 10: Structural version of Z function - results

## Β4-Σύγκριση προσεγγίσεων

Για την υλοποίηση του ερωτήματος Β1, ακολουθήσαμε δύο διαφορετικές προσεγγίσεις, µία µε βάση την διεργασία (Behavioral) και µία structural. Παρατηρήσαμε πως οι 2 αυτές υλοποιήσεις, παρότι µας δίνουν το ίδιο αποτέλεσμα κατά την προσομοίωση, είναι πολύ διαφορετικές ως προς τον κώδικα που απαιτούν. Για την υλοποίηση µέσω διεργασίας, χρειάστηκε να κατασκευάσουμε τον πίνακα αληθείας του κυκλώματος και στη συνέχεια να εντοπίσουμε για ποιες τιμές των εισόδων, η έξοδός µας, Ζ, γίνεται µονάδα.

Στη συνέχεια αρκούσε απλά να χρησιμοποιήσουμε έναν elseif βρόγχο (ή use cases όπως χρησιμοποιήσαμε και στο ερώτημα της ενότητας Α) όπου θα καθορίζουμε την τιμή της εξόδου, µέσω των τιμών που λαμβάνουν οι είσοδοι. Η υλοποίηση αυτή µας βοηθάει στην περίπτωση που δεν γνωρίζουμε το ακριβές κύκλωμά που µας οδηγεί σε έναν πίνακα αληθείας και γνωρίζουμε µόνο το πως συμπεριφέρεται η έξοδος για κάθε τιμή των εισόδων του.

Σε αντίθετη περίπτωση, ο κώδικας για την structural υλοποίηση απαιτεί τη γνώση του ακριβούς κυκλώματος που µας δίνεται ώστε να γίνει η υλοποίηση κάθε πύλης µέχρι να καταλήξουμε στην έξοδο Ζ.