Ολοκληρωμένα Κυκλώματα

Εργαστηριακή άσκηση: Επιβεβαίωση της ορθής λειτουργίας ενός κυκλώματος διαιτησίας τύπου Round Robin

Δημήτρης Κωνσταντίνου, Γιώργος Δημητρακόπουλος

Σκοπός ενός κυκλώματος arbiter (διαιτητή) είναι η επιλογή στην έξοδό του, το πολύ μίας εκ των εισόδων του. Οι είσοδοι μπορούν να θεωρηθούν ως requests (αιτήματα), περιμένοντας grant (επιβεβαίωση) ώστε να χρησιμοποιήσουν κάποιον πιθανώς κοινό πόρο.

Ένα κύκλωμα arbiter θα πρέπει να παράγει αυστηρά ένα grant, σε περίπτωση που υπάρχει τουλάχιστον ένα request και μόνο σε αντίθετη περίπτωση δεν παράγεται grant. Ο τρόπος ο οποίος παράγονται τα grants σε κάθε κύκλο, αναλόγως τα requests είναι και η ειδοποιός διαφορά μεταξύ υλοποιήσεων arbiter.

Στο εργαστήριο αυτό θα επικεντρωθούμε στον round robin arbiter. Σε αυτή την περίπτωση, σε κάθε κύκλο ρολογιού μία είσοδος ορίζεται ως η είσοδος με τη μεγαλύτερη προτεραιότητα, ενώ η προτεραιότητα των επόμενων εισόδων φθίνει κυκλικά προς μία σταθερή κατεύθυνση. Το πρώτο ενεργό request με την υψηλότερη προτεραιότητα είναι αυτό που κερδίζει το grant. Η προτεραιότητα μεταβάλετε ώστε το request το οποίο κέρδισε το τελευταίο grant μέχρι και αυτόν τον κύκλο, να έχει την μικρότερη δυνατή προτεραιότητα από τον επόμενο κύκλο ρολογιού.

Στην περίπτωσή μας, μετά το reset η είσοδος με τη μεγαλύτερη προτεραιότητα είναι η #0 (req[0]). Αυτό σημαίνει πως για τις 4 εισόδους έχουμε τις εξής **προτεραιότητες**: $Pri[0] > Pri[1] > Pri[2] > Pri[3]$. Αντίστοιχα, σε κάποιο επόμενο κύκλο όπου τυχαίνει η είσοδος#2 να έχει υψηλότερη προτεραιότητα, οι **προτεραιότητες** θα ήταν: $Pri[2] > Pri[3] > Pri[0] > Pri[1]$.

Για παράδειγμα στον κύκλο 1 του πίνακα η θέση με την υψηλότερη προτεραιότητα είναι η θέση 0 ενώ ενεργά requests έχουν οι θέσεις 0 και 3. Έτσι το grant θα δοθεί στη θέση 0 και η προτεραιότητα στον επόμενα κύκλο ρολογιού (Piorioty_Nxt) θα δοθεί στη θέση 1. Στον κύκλο 1 ξεκινώντας και πάλι από τη θέση με την υψηλότερη προτεραιότητα (θέση 1) θα δοθεί το grant στη θέση 3 καθώς είναι η πρώτη με ενεργό request στον κύκλο εξέτασης των θέσεων 1->2->3->0.

Cycle#	Requests[3:0]	Priority	Grants[3:0]	Priority_Nxt
1	1001	0	0001	1
2	1001	1	1000	0
3	0101	0	0001	1

Για το εργαστήριο

Σας έχει δοθεί το αρχείο **rr_arbiter_buggy.svp** το οποίο υλοποιεί έναν round robin arbiter και υλοποιεί το interface που φαίνεται παρακάτω.

```
module rr_arbiter_buggy #(
  parameter int REQS         = 4,  // The number of requests (4 is enough)
  parameter int BUG          = 0   // 1, 2, 4, 8 to insert bug.(0 means no bug)
) (
 //Input-Output List
 input  logic clk,
 input  logic rst,

 input  logic[REQS-1:0] reqs_i,     // The requests to choose from
 output logic[REQS-1:0] grants_o,   // The winner (if any)
 output logic           any_grant_o // Asserted when there is at least one grant
);
```

Όμως το αρχείο **rr_arbiter_buggy.svp** του arbiter έχει δύο ιδιαιτερότητες:

1) Περιέχει την παράμετρο «**BUG**» και ανάλογα την τιμή της, εισάγει κάποιο σφάλμα το οποίο θα πρέπει να εντοπίσετε με ένα testbench που θα υλοποιήσετε.
2) Η πραγματική υλοποίηση του arbiter είναι κρυπτογραφημένη ώστε να εντοπίσετε τα σφάλματα **μόνο** με την βοήθεια του testbench. Ωστόσο, το αρχείο περιέχει την πλήρης υλοποίηση και συμπεριφέρεται με τον ίδιο τρόπο όπως κάθε .sv (π.χ. για compile “vlog rr_arbiter_buggy.svp”)

Σκοπός του εργαστηρίου είναι η ανάπτυξη ενός testbench το οποίο θα οδηγεί το module rr_arbiter_buggy, τόσο σε κανονικές όσο και σε ακραίες περιπτώσεις λειτουργίας ώστε να ανακαλυφθούν πιθανά λάθη σχεδιασμού που παραβιάζουν το πρωτόκολλο λειτουργίας.

Έτσι, θα πρέπει να έχετε υλοποιήσει ένα testbench το οποίο θα είναι ικανό να εντοπίζει τα σφάλματα για τιμές BUG = {1, 2, 4, 8} (για BUG=0 δεν εισάγεται σφάλμα). Για το εργαστήριο θα πρέπει να έχετε καταγράψει τις εσφαλμένες συμπεριφορές και φυσικά θα πρέπει να μπορείτε να τις επαναλάβετε. Δεν χρειάζεται (και δεν μπορείτε) να εντοπίσετε την αιτία του bug. Μπορείτε μόνο να το ενεργοποιήσετε να συμβεί και να περιγράψετε ποιο είναι το λάθος.