Πανεπιστήμιο Πατρών

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

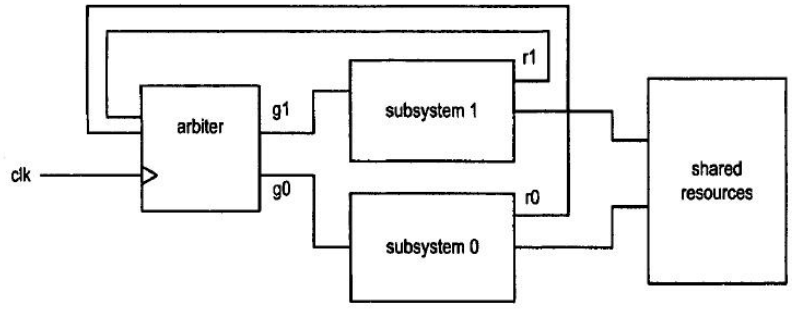
Εργαστήριο Σχεδίασης Ολοκληρωμένων Κυκλωμάτων

Σχεδιασμός Ολοκληρωμένων Συστημάτων με τεχνικές VLSI

Χειμερινό Εξάμηνο 2023

Άσκηση 1: Arbiter («Κύκλωμα Διαιτησίας»)

Σε μεγάλα συστήματα, κάποιοι πόροι (resources) είναι διαμοιραζόμενοι από πολλά υποσυστήματα. Για παράδειγμα, μερικοί επεξεργαστές μπορεί να μοιράζονται το ίδιο block μνήμης ενώ πολλές περιφερειακές συσκευές μπορεί να είναι συνδεδεμένες με το ίδιο bus. Ο arbiter είναι το κύκλωμα που επιλύει κάθε διένεξη (conflict) και συντονίζει την πρόσβαση στον εκάστοτε διαμοιραζόμενο πόρο (shared resource). Ας θεωρήσουμε τον arbiter του παρακάτω σχήματος.



Τα δύο υποσυστήματα επικοινωνούν με τον arbiter μέσω των σημάτων αίτησης (request signal) και των σημάτων παραχώρησης άδειας (grant signals), τα οποία ονομάζονται r(1) και g(1) για το υποσύστημα 1 και r(0) και g(0) για το υποσύστημα 0. Η λειτουργία του συνολικού συστήματος έχει ως εξής:

Όταν ένα υποσύστημα χρειάζεται έναν πόρο, δίνει σήμα αίτησης στον arbiter με ενεργοποίηση του σήματος r. Ο arbiter παρακολουθώντας τη χρήση των πόρων αλλά και των σημάτων αίτησης όλων των υποσυστημάτων, δίνει σήμα άδειας χρήσης στο εκάστοτε υποσύστημα, ενεργοποιώντας το αντίστοιχο σήμα g. Αφού ενεργοποιηθεί το σήμα άδειας για ένα υποσύστημα, τότε και μόνον τότε έχει δικαίωμα χρήσης του πόρου. Μετά το πέρας της χρήσης του πόρου από το υποσύστημα, το τελευταίο απενεργοποιεί το σήμα αίτησης και ο arbiter ελευθερώνει τον πόρο.

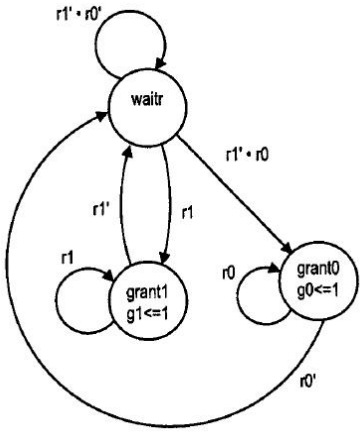
Συνεπώς, αφού ο arbiter αποφασίζει εν μέρει από γεγονότα που συνέβησαν νωρίτερα (προηγούμενα σήματα άδειας και αίτησης), χρειάζεται εσωτερικές «καταστάσεις» για να καταγράφει τι έγινε στο παρελθόν. Έτσι, μπορεί εύκολα να κατασκευαστεί με χρήση Μηχανής Πεπερασμένης Καταστάσεων (FSM).

Ερώτημα Α)

Να περιγραφεί σε VHDL ο arbiter με βάση το παρακάτω διάγραμμα FSM τύπου Moore**.** Το διάγραμμα αποτελείται από 3 καταστάσεις, waitr, grant1 και grant0. Η πρώτη κατάσταση καταδεικνύει ότι οι πόροι είναι ελεύθεροι και ο arbiter περιμένει αίτηση. Οι άλλες δυο καταδεικνύουν ότι είτε το υποσύστημα 1 είτε το 0 έχει δεσμεύσει τον πόρο.

Αρχικά ο arbiter είναι στην κατάσταση waitr. Αν ενεργοποιηθεί το σήμα r(1) σε ανιούσα ακμή ρολογιού, τότε δίνει άδεια στο υποσύστημα 1 μεταβαίνοντας στην κατάσταση grant1 ,όπου το σήμα g(1) γίνεται '1'. Με το πέρας της χρήσης του πόρου, το σήμα r(1) απενεργοποιείται και ο arbiter επιστρέφει στην κατάσταση αναμονής. Αντίστοιχη είναι η λειτουργία για την ικανοποίηση του υποσυστήματος 2.

Ένα κρίσιμο ζήτημα στη σχεδίαση του arbiter είναι η διαχείριση ταυτόχρονων αιτημάτων από τα 2 υποσυστήματα. Για τις ανάγκες του ερωτήματος αυτού, ο arbiter δίνει προτεραιότητα στο υποσύστημα 1.



Ερώτημα Β)

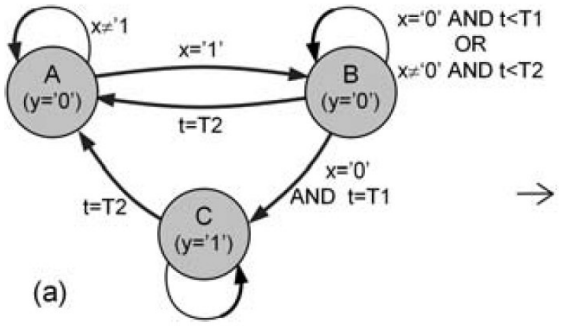
Στο παραπάνω ερώτημα ο arbiter δίνει προτεραιότητα στο υποσύστημα 1. Αυτό όμως ενδέχεται να δημιουργήσει πρόβλημα σε περίπτωση συνεχόμενων αιτήσεων του υποσυστήματος 1. Το πρόβλημα αυτό διορθώνεται με κατάλληλη αλλαγή του διαγράμματος της FSM ώστε να είναι περισσότερο δίκαιο. **Το νέο διάγραμμα θα πρέπει να «θυμάται» ποιο υποσύστημα είχε χρησιμοποιήσει τελευταία τον πόρο και θα δίνει προτεραιότητα στο άλλο υποσύστημα σε ενδεχόμενη ταυτόχρονη αίτηση και των δύο.**

Σχεδιάστε αρχικά το διάγραμμα της FSM «στο χαρτί» με πλήρως λειτουργικές καταστάσεις ώστε να ικανοποιούνται οι παραπάνω προϋποθέσεις. Στη συνέχεια, με βάση το νέο διάγραμμα της FSM, περιγράψτε σε VHDL ένα «δικαιότερο» arbiter.

ΠΡΟΣΟΧΗ: ΔΕΝ ΖΗΤΕΙΤΑΙ υλοποίηση των υποσυστημάτων και των πόρων καθώς και η συνδεσμολογία τους με τον arbiter. Το αρχικό block diagram δίνεται καθαρά για λόγους κατανόησης της λειτουργίας του arbiter. Για τις ανάγκες τις άσκησης αυτής οι πόροι και τα υποσυστήματα θεωρούνται black boxes. **ΖΗΤΕΙΤΑΙ ΜΟΝΟ** η υλοποίηση της FSM, δηλαδή του arbiter. Το κύκλωμα θα έχει ως **εισόδους** τα: **clk**, **reset**, **r** και ως **έξοδο** το **g**. Θεωρείστε τα σήματα εισόδου / εξόδου **r** και **g** ως **std_logic_vector**.

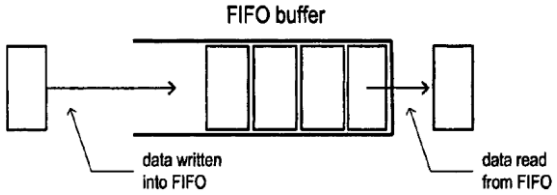
Άσκηση 2: FSM με ενσωματωμένους timers

Στο επόμενο σχήμα δείχνεται το διάγραμμα μίας μηχανής πεπερασμένων καταστάσεων, όπου x, y είναι η εξωτερική είσοδος και έξοδος, αντίστοιχα. Επιπλέον, τα T1 και T2 αντιστοιχούν σε κύκλους ρολογιού όπου T1 < T2. Περιγράψτε την FSM και επιβεβαιώστε την ορθή λειτουργία της. Οι τιμές T1 και T2 θα πρέπει να είναι παράμετροι στον κώδικα. Προφανώς η FSM είναι συγχρονισμένη με το ρολόι του συστήματος.

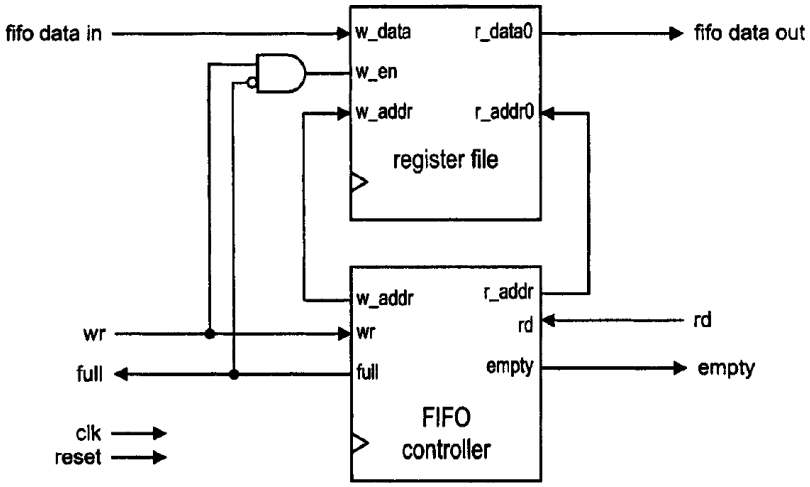


Άσκηση 3: Register-based Synchronous FIFO buffer

Ένας σύγχρονος FIFO buffer είναι ένα κύκλωμα όπου τα δεδομένα εισέρχονται από μία είσοδο και διαβάζονται από μία έξοδο κάτω από τη λογική First-In First-Out σε συγχρονισμό με το ρολόι του συστήματος. Δηλαδή, τα δεδομένα αποθηκεύονται διαδοχικά στις θέσεις μνήμης και διαβάζονται με τη σειρά που αποθηκεύτηκαν, όπως φαίνεται στο παρακάτω σχήμα.



Ένας τρόπος υλοποίησης αυτού του κυκλώματος είναι η χρήση ενός κυκλώματος ελέγχου σε ένα γενικό σύστημα μνήμης (π,χ ένα register file, όπως δείχνεται στο επόμενο σχήμα).



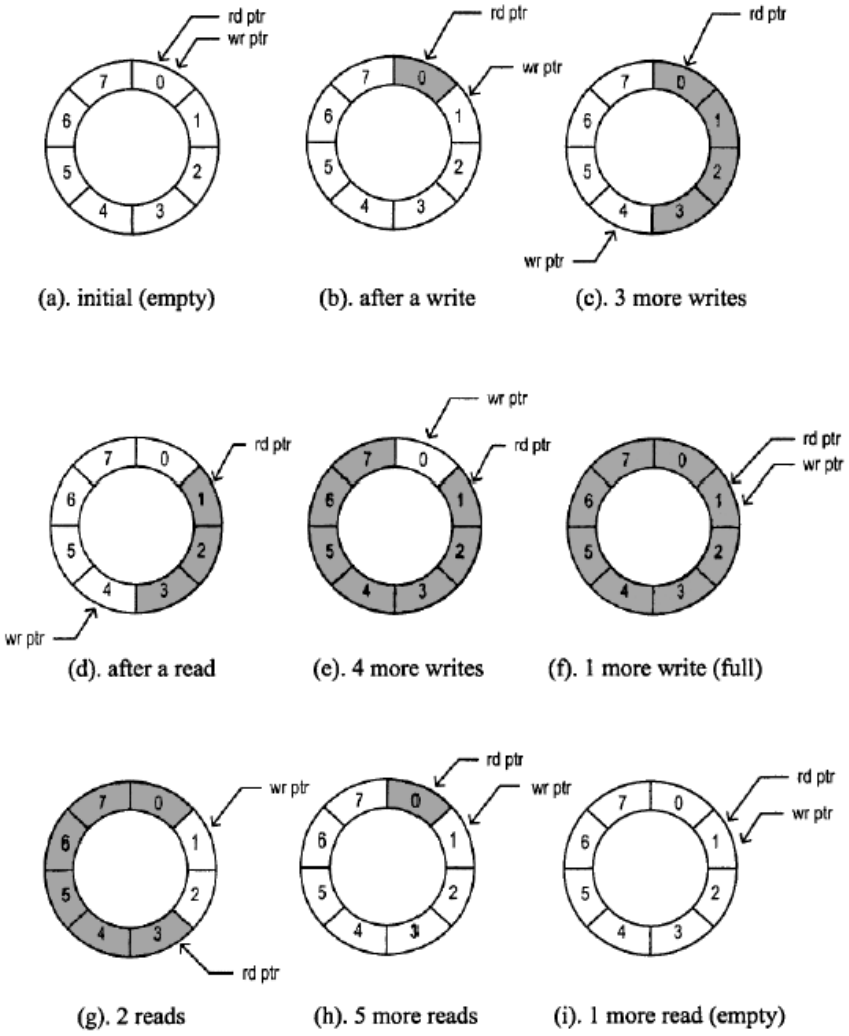
Όπως φαίνεται, οι είσοδοι είναι:
α) **fifo_data_in**: δεδομένα εγγραφής
β) **wr, rd**: σήματα εγγραφής και ανάγνωσης
γ) clk, reset: χρησιμοποιούνται και από τη μνήμη και το FIFO controller
ενώ οι έξοδοι είναι:
α) **fifo_data_out**: δεδομένα ανάγνωσης
β) **full, empty**: ο buffer είναι γεμάτος / άδειος. Όταν είναι γεμάτος δε μπορεί να δεχτεί επιπλέον δεδομένα για εγγραφή, ενώ όταν είναι άδειος δε μπορεί να εξάγει δεδομένα ανάγνωσης.

Για την αποφυγή εγγραφών όταν ο buffer είναι γεμάτος, το register file έχει ένα σήμα εισόδου w_en (write enable) το οποίο παράγεται με κατάλληλη λογική όπως φαίνεται στο σχήμα.

Επιπλέον, η μονάδα FIFO controller παράγει και δύο εσωτερικά σήματα w_addr (write_address) και rd_addr (read_address), που χρησιμοποιούνται για τη διευθυνσιοδότηση του καταχωρητή εγγραφής και ανάγνωση, αντίστοιχα.

Στην ουσία τα σήματα αυτά λειτουργούν ως δείκτες (pointers) για τον καθορισμό της θέσης όπου θα γραφτεί το νέο εισερχόμενο δεδομένο (wr_ptr) και της θέσης από όπου θα διαβαστεί το δεδομένο εξόδου (rd_ptr). Η λειτουργία των δεικτών αυτών φαίνεται στο ακόλουθο σχήμα μαζί την παραγωγή των σημάτων full & empty, όταν ο buffer είναι γεμάτος ή άδειος.

Περιγράψτε σε VHDL έναν FIFO buffer 4 θέσεων, όπου σε κάθε θέση έχει εύρος 8 ψηφία.

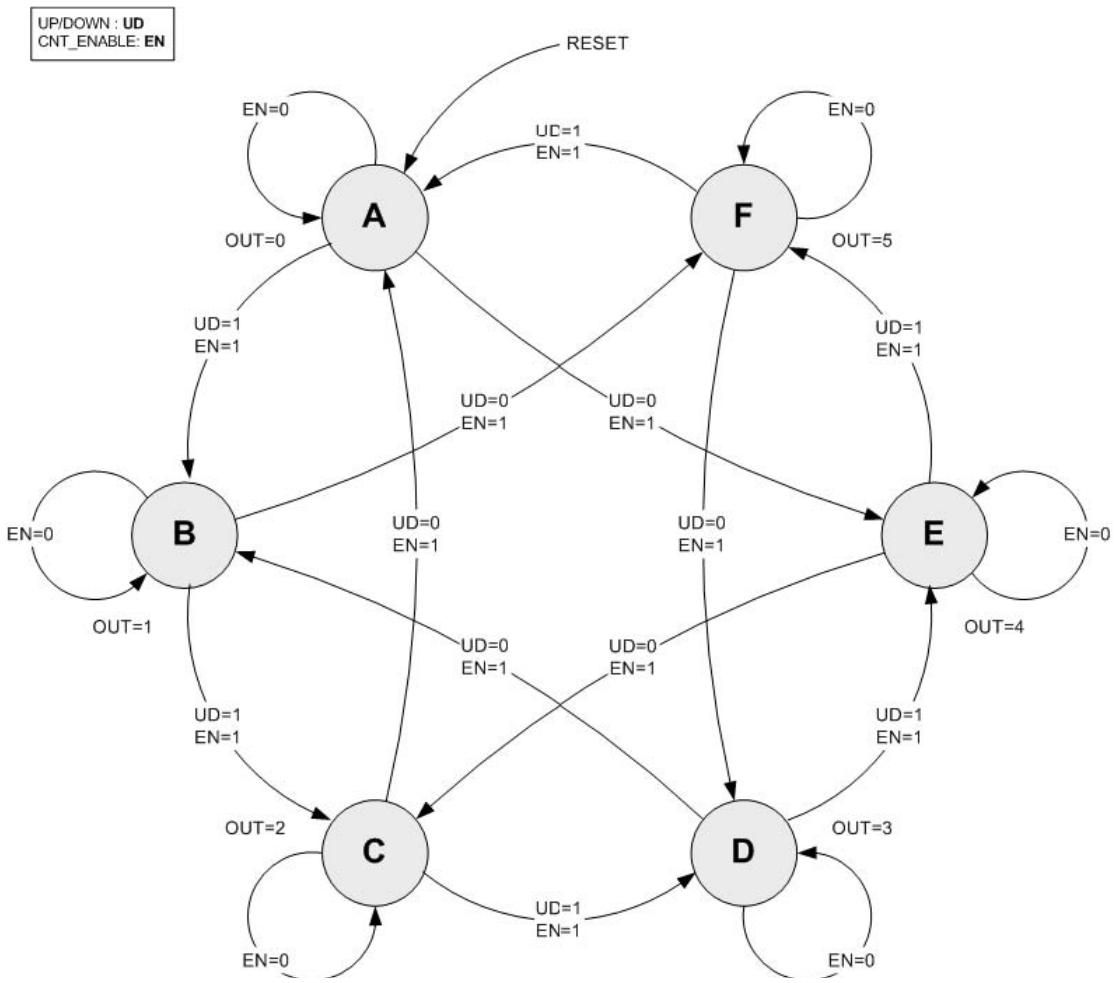


Άσκηση 4: Υλοποίηση FSM

Στόχος της άσκησης είναι να υλοποιηθεί μια FSM σε γλώσσα VHDL όταν είναι γνωστό το διάγραμμα καταστάσεων της. Το κύκλωμα θα έχει τις ακόλουθα σήματα εισόδου/εξόδου:

- Ένα σήμα CLOCK (ρολόι)
- Ένα σήμα RST. Όταν το RST είναι ’1’ η έξοδος θα μηδενίζεται.
- Ένα σήμα Up Down (U_D) ανάλογα με το οποίο η FSM θα δίνει ως έξοδο μέτρηση προς τα «πάνω» ανά 1, όταν το U_D είναι ‘1’, και μέτρηση προς τα «κάτω» ανά 2, όταν το U_D είναι ‘0’, σε κάθε κύκλο του ρολογιού (Clock).
- Ένα σήμα Count Enable (EN) ανάλογα με το οποίο η FSM θα δίνει ως έξοδο μέτρηση προς τα “πάνω” ή προς τα “κάτω” (όταν EN = ‘1’) ή η έξοδος της FSM θα παραμένει σταθερή όταν EN = ‘0’, σε κάθε κύκλο του ρολογιού (Clock).
- Η έξοδος είναι των 3 bits, η αρχικοποίηση της FSM γίνεται στην τιμή 0 (“000”) και η μέγιστη τιμή της εξόδου είναι 5 (“101”).

Το διάγραμμα καταστάσεων για το κύκλωμα είναι το παρακάτω:


UP/DOWN : UD
CNT_ENABLE: EN
RESET
A
B
C
D
E
F
OUT=0
OUT=1
OUT=2
OUT=3
OUT=4
OUT=5
EN=0
UD=1
EN=1
UD=0
EN=1