Πανεπιστήμιο Πατρών

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Εργαστήριο Σχεδίασης Ολοκληρωμένων Κυκλωμάτων

Σχεδιασμός Ολοκληρωμένων Ψηφιακών Συστημάτων

Χειμερινό Εξάμηνο 2023

Εργαστήριο 4
Sequential VHDL coding

Άσκηση 1: LED time-multiplexing

Θεωρείστε ότι πρέπει να οδηγήσετε τέσσερα seven-segment LED displays. Για να μειωθούν τα I/O εξωτερικά σήματα χρησιμοποιείται χρονική πολύπλεξη.

Συγκεκριμένα, τα τέσσερα displays έχουν τα δικά τους σήματα ενεργοποίησης (an0, …, an3), αλλά μοιράζονται (με πολύπλεξη στο χρόνο) 8 κοινές γραμμές εισόδου που χρησιμοποιούνται για την απεικόνιση της τιμής στα segments. Δε ζητείται μετατροπή ώστε τα displays να αναπαριστούν hex μορφή. Συγκεκριμένα, η εκάστοτε είσοδος χρησιμοποιείται όπως είναι για να ανάψει το εκάστοτε LED.

Τα σήματα (an0, …, an3) ενεργοποιούν τα segments και είναι ενεργά σε χαμηλή στάθμη.

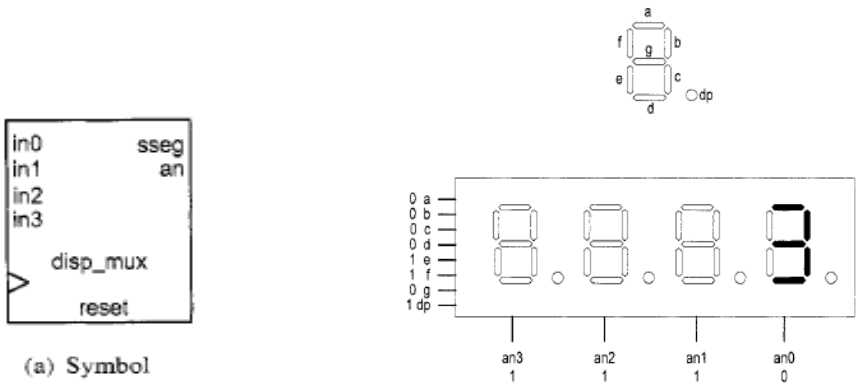
Το κύκλωμα που πρέπει να σχεδιάσετε έχει την ακόλουθη διεπαφή

Είσοδοι: in0, in1, in2, in3 (όλες 8 bit), clk, reset (asynchronous). Το ρολόι του κυκλώματος έχει περίοδο 50 MHz.

Έξοδοι: an (4 bit), sseg (8 bit). Για το sseg τα 7 ψηφία χρησιμοποιούνται για την αναπαράσταση του αριθμού και ένα για την τελεία (dot point -dp) του κάθε αριθμού.

Το ζητούμενο είναι ότι τα displays πρέπει να αναβοσβήνουν με τέτοιο ρυθμό ώστε να φαίνονται συνεχώς αναμμένα από το ανθρώπινο μάτι. Για το λόγο αυτό το κάθε σήμα (an0, …, an3), πρέπει να ενεργοποιείται με μία συχνότητα περίπου ίση με 800 Hz ($50MHz/2^{16}$).

Στα ακόλουθα σχήματα παρουσιάζονται το σύμβολο του κυκλώματος και το αντίστοιχο χρονικό διάγραμμα. *Συνθέστε και προσομοιώστε τη λύση σας*



(a) Symbol

Figure 4.5 Time-multiplexed seven-segment LED display.

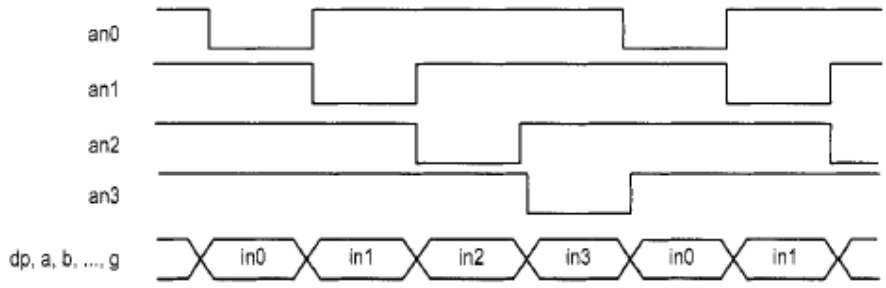


Figure 4.6 Timing diagram of a time-multiplexed seven-segment LED display.

Άσκηση 2: Κύκλωμα Αλγορίθμου Merge Sort Τεσσάρων αριθμών

Περιγράψτε σε VHDL ένα κύκλωμα που υλοποιεί τον αλγόριθμο Merge Sort για 4 αριθμούς. Το κύκλωμα έχει την ακόλουθη διεπαφή:

Είσοδοι:

Clk, reset (ασύγχρονη αρχικοποίηση)**, en,**

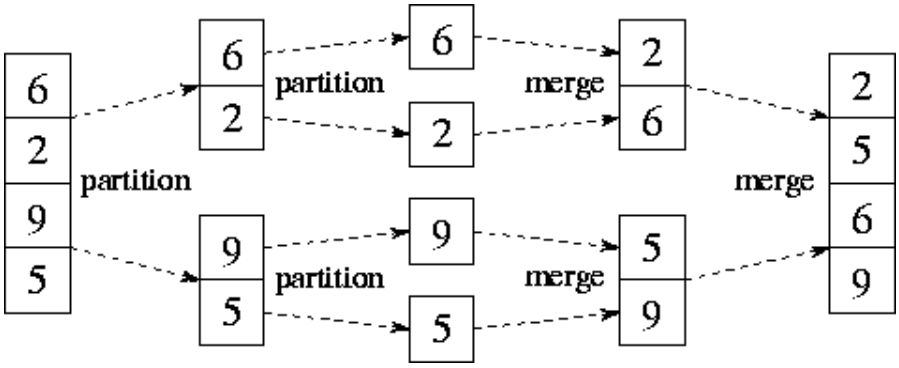
In_a, In_b, In_c, In_d: 16-bit μη-προσημασμένοι αριθμοί

Έξοδος

Sorted_Out_a, Sorted_Out_b, Sorted_Out_c, Sorted_Out_d: οι αριθμοί στην σωστή (αύξουσα) σειρά

Το κύκλωμα λειτουργεί ως εξής: Δέχεται τις εισόδους παράλληλα εισάγοντας αυτές σε καταχωρητές εισόδου. Κατόπιν, από τους καταχωρητές αυτούς τα σήματα εισέρχονται στη λογική για το αρχικό split-sort-merge (προφανώς, το split είναι περιττό αφού ήδη τα σήματα είναι ξεχωριστά). Κατόπιν, τέσσερα σήματα (ανά δύο sorted) εισέρχονται σε νέους καταχωρητές (ενδιάμεσους). Στο επόμενο παλμό εισέρχονται στη λογική που είναι υπεύθυνη για το τελικό sorting. Τέλος, στην ορθή σειρά πλέον, εισέρχονται σε τελικούς καταχωρητές (εξόδου) από όπου και προκύπτουν οι έξοδοι. Συνολικά, για τον πλήρη υπολογισμό χρειάζονται 3 κύκλοι.

(Hint: Μπορείτε να χρησιμοποιήστε processes τόσο για τη λογική του sorting -και στο πρώτο και στο δεύτερο επίπεδο- αλλά και για την υλοποίηση των registers. Μην ξεχνάτε ότι το εσωτερικό των processes «εκτελείται» σειριακά ενώ οι processes μεταξύ τους ως οντότητες παράλληλα.)



Άσκηση 3: Κύκλωμα Υπολογισμού Μέσου Όρου

Να σχεδιάσετε και να υλοποιήσετε κύκλωμα υπολογισμού του μέσου όρου 8 μη προσημασμένων αριθμών. Το σύστημα θα έχει τα ακόλουθα σήματα εισόδου και εξόδου: α) ρολόι (clk), β) ασύγχρονο σήμα αρχικοποίησης (reset), γ) είσοδο και δ) έξοδο (μέσος όρος).

Η έξοδος θα είναι 16-bit, και κάθε αριθμός (εκ των 8) της εισόδου θα είναι και αυτός 16-bit.

Θα σχεδιαστούν 2 εκδόσεις του κυκλώματος:

Ι. Σειριακή Είσοδος. Τα δεδομένα (αριθμοί) θα εισέρχονται σειριακά (ένα-ένα) από μια είσοδο 16-bit ανά κύκλο ρολογιού. Μετά από 8 κύκλους (στον 9ο) θα παράγεται το αποτέλεσμα του μέσου όρου και θα δίνεται στην έξοδο. Η διαδικασία θα συνεχίζεται για επόμενη οκτάδα.

(*Σημείωση: Όταν θα υπολογίζεται ο μέσος όρος μιας οκτάδας, θα πρέπει το κύκλωμα να είναι έτοιμο να δεχθεί έναν αριθμό από την επόμενη οκτάδα. Ο υπολογισμός θα γίνεται σε burst-mode, δηλαδή δεν θα διακόπτεται όταν υπολογιστεί ο μέσος όρος της πρώτης οκτάδας. Το κύκλωμα θα αναμένει*

είσοδο άμεσα. Συνεπώς δεν απαιτείται ξεχωριστό σήμα έναρξης/λήξης υπολογισμού. Μπορείτε να αξιοποιήσετε το γνωστό κύκλωμα accumulator-adder για υπολογισμό και αποθήκευση.)

ΙΙ. Παράλληλη Είσοδος. Τα δεδομένα θα εισέρχονται παράλληλα (όλα μαζί) σε 1 κύκλο ρολογιού, στον επόμενο κύκλο (2ος κύκλος) θα παράγεται το αποτέλεσμα και στον μεθεπόμενο (3ος) θα δίνεται στην έξοδο.

Και στις δύο εκδόσεις, τη διαίρεση (/8) μπορείτε να την εκτελέσετε μέσω ολίσθησης (ακόμα και αν έχουμε απώλειες σε ακρίβεια). Σε κάθε περίπτωση όμως αν σκεφτείτε άλλο τρόπο, επιστημονικά τεκμηριωμένο, είναι αποδεκτός. Επίσης, θα πρέπει να υπάρχουν καταχωρητές (registers) τόσο στην είσοδο των κυκλωμάτων όσο και στην έξοδο.

Ποιό είναι το critical path της κάθε σχεδίασης? Ποιος από τους δύο είναι γρηγορότερος και γιατί? Ποιος είναι πιο αποδοτικός όσον αφορά την ρυθμαπόδοση (throughput)?

Άσκηση 4: Μελέτη κώδικα – Παραγωγή F/Fs

Α) Μελετήστε τους τρείς παρακάτω κώδικες και σχεδιάστε τα κυκλώματα που θεωρείτε ότι θα δημιουργηθούν. Πόσα F/Fs χρειάζονται σε κάθε περίπτωση?

Β) Μεταγλωττίστε τους κώδικες για να επιβεβαιώστε τις απαντήσεις σας

Γ) Είναι τα κυκλώματα λειτουργικά ισοδύναμα ? Ποιο θα διαλέγατε και γιατί?

```
1 ---------------------------------------
2 ENTITY test IS
3 PORT (clk: IN BIT;
4 x: IN BIT_VECTOR(7 DOWNTO 0);
5 sel: IN INTEGER RANGE 0 TO 7;
6 y: OUT BIT);
7 END ENTITY;
8 -----code 1:-------------------------
9 ARCHITECTURE circuit OF test IS
10 BEGIN
11 PROCESS (clk)
12 VARIABLE temp: BIT;
13 BEGIN
14 temp := x(sel);
15 IF clk'EVENT AND clk='1' THEN
16 y <= temp;
17 END IF;
18 END PROCESS;
19 END ARCHITECTURE;
-------------------------------------
-----code 2:----------------------------
11 PROCESS (clk)
12 VARIABLE temp: BIT_VECTOR(7 DOWNTO 0);
13 BEGIN
14 IF clk'EVENT AND clk='1' THEN
15 temp := x;
16 END IF;
17 y <= temp(sel);
18 END PROCESS;
-----------------------------------------
-----code 3:----------------------------
11 PROCESS (clk)
12 BEGIN
13 IF clk'EVENT AND clk='1' THEN
14 y <= x(sel);
15 END IF;
```

```
16 END PROCESS;
```

Οι ασκήσεις να υλοποιηθούν χρησιμοποιώντας concurrent και sequential κώδικα όχι όμως με δομικό κώδικα δηλαδή χρήση port maps.