

E-CAD

Αναφορά ασκήσεων μαθήματος και εργασίας εξαμήνου

Ανδρέας Καρατζάς

AM: 1054336

Τμήμα Μηχανικών η/υ και Πληροφορικής

ankaratzas@ceid.upatras.gr

Αικατερίνη - Μαρία Παντελεάκη

AM: 5862 (NEO: 1041574)

Τμήμα Μηχανικών η/υ και Πληροφορικής

panteleaki@ceid.upatras.gr

Περιεχόμενα

1 Πρώτη εργαστηριακή άσκηση	5
1.1 Ανάλυση θεωρίας και προσομοιώσεις στο OrCAD	5
1.2 Ζήτημα άσκησης	10
Βιβλιογραφία	13
2 Δεύτερη εργαστηριακή άσκηση	15
2.1 Top-down design	15
2.2 Bottom-up design	18
Βιβλιογραφία	21
3 Τρίτη εργαστηριακή άσκηση	23
3.1 Σχεδιαστικά κύτταρα	23
3.2 Η πλακέτα	24
4 Τέταρτη εργαστηριακή άσκηση	27
4.1 Το κύκλωμα hasp	27
4.2 Εξομοίωση του σχεδιασμού	28
4.3 Φυσική Υλοποίηση του σχεδιασμού	30
Βιβλιογραφία	31
5 Πέμπτη εργαστηριακή άσκηση	33
5.1 Εισαγωγή	33
5.2 Υλοποίηση μετρητή των 4 bits	34
5.3 Υλοποίηση μετρητή των 8 bits	34
5.4 Δημιουργία συνόλου διανυσμάτων εξομοίωσης	35
5.5 Επαλήθευση της λογικής λειτουργίας του μετρητή των 8 bits	36
5.6 Μετρητής 8 bits με σύγχρονη είσοδο καθαρισμού	37
5.7 Μέγιστη συχνότητα λειτουργίας του μετρητή των 8 bits	39
6 Έκτη εργαστηριακή άσκηση	41
6.1 Εισαγωγή	41
6.2 Κύκλωμα απεικόνισης δυαδικού αριθμού από τα dip switches στα led bars	41
6.3 Κύκλωμα Multiplier 2 αριθμών	42
6.3.1 Οι αριθμοί να είναι μη προσημασμένοι	42
6.3.2 Οι αριθμοί να είναι σε παράσταση συμπληρώματος του 2	44

7 Έβδομη εργαστηριακή άσκηση	47
7.1 Εισαγωγή	47
7.2 Λειτουργία stop-watch	47
7.3 Μέτρηση δεκάτων	48
8 Όγδοη εργαστηριακή άσκηση	53
8.1 Εισαγωγή	53
8.2 Εφαρμογή ροής σύνθεσης και υλοποίησης του κυκλώματος	53
8.3 Λειτουργία προηγούμενου πλήκτρου	53
8.4 Λειτουργία calculator	57
9 Project	63
9.1 Εισαγωγή	63
9.2 Το πληκτρολόγιο	63
9.3 Η οιδόνη	67
9.4 Η λειτουργία <i>Περιστροφή</i>	73
9.5 Το φαινόμενο <i>Explode</i>	73
9.6 Παράρτημα	73

Πρώτη εργαστηριακή άσκηση

1.1 Ανάλυση θεωρίας και προσομοιώσεις στο OrCAD

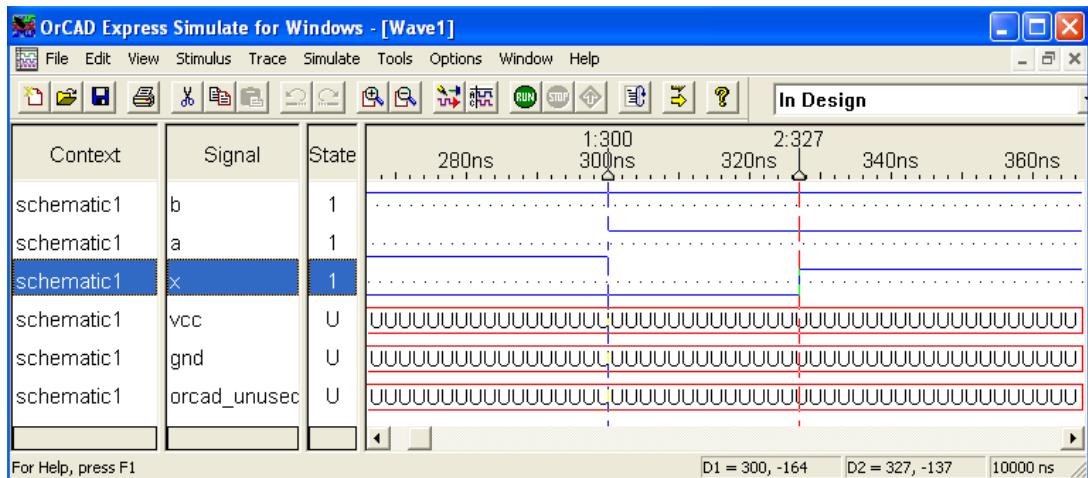
Στην άσκηση σχεδιάστηκαν 5 διαφορετικά κυκλώματα. Σκοπός ήταν ο υπολογισμός των χρόνων καθυστέρησης διάδοσης (*propagation delay*) για τις οικογένειες ολοκληρωμένων:

- Standard TTL: Πύλη AND 2 εισόδων 7408
- ALS : Πύλη AND 2 εισόδων 74ALS08
- AS : Πύλη AND 2 εισόδων 74AS08
- S : Πύλη AND 2 εισόδων 74S08
- LS TTL : Πύλη AND 2 εισόδων 74LS08

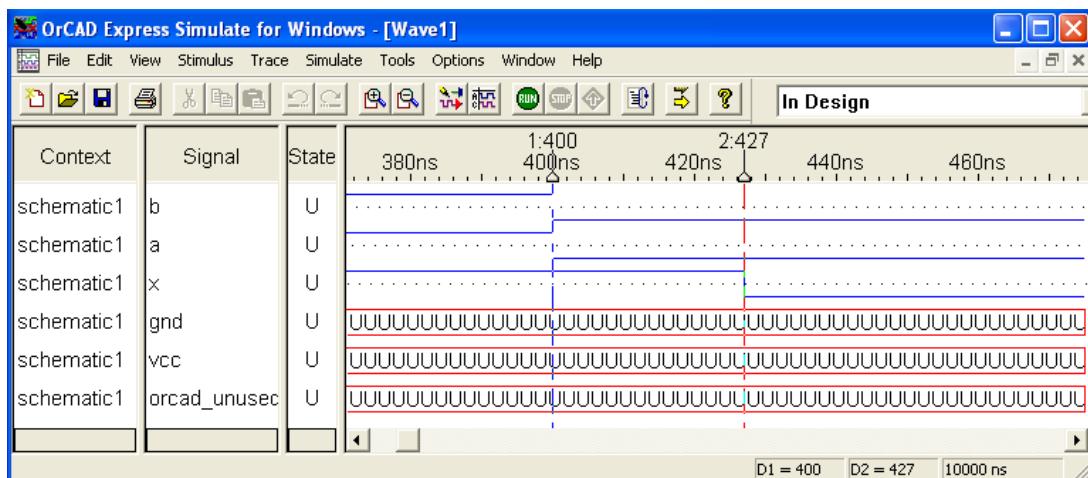
Παρακάτω υπάρχουν τα στιγμιότυπα οθόνης (*screenshots*), στα οποία βασίστηκε ο υπολογισμός των ζητούμενων χρόνων.

Πύλη AND 2 εισόδων 7408 Στο Σχήμα 1.1 φαίνεται η μετάβαση εξόδου από χαμηλό δυναμικό (λογικό 0), σε υψηλό δυναμικό (λογικό 1) και οι αντίστοιχες χρονικές στιγμές μέσα στην εξομοιώση¹. Στο Σχήμα 1.2 υπάρχει όμοια η μετάβαση εξόδου από υψηλό σε χαμηλό δυναμικό μαζί με τις αντίστοιχες χρονικές στιγμές. Παρατηρείται πως και στις 2 μεταβάσεις η καθυστέρηση διάδοσης είναι ίδια και ίση με 27 nsec. Στα πλαίσια της άσκησης, ο χρόνος καθυστέρησης μετάδοσης αρχικοποιήθηκε σε τυπική καθυστέρηση. Στην AND της οικογένειας standard TTL. Ωστόσο, καθυστέρηση μετάδοσης στα 27 nsec είναι η χειρότερη περίπτωση της AND που κατασκευάζει η Fairchild [1].

¹Οι χρονικές στιγμές έχουν σημειωθεί από τον *marker* όπως υποδεικνύεται στην εκφώνηση



$$\Sigmaχήμα 1.1: \Delta t = 327 \text{ nsec} - 300 \text{ nsec} = 27 \text{ nsec}$$

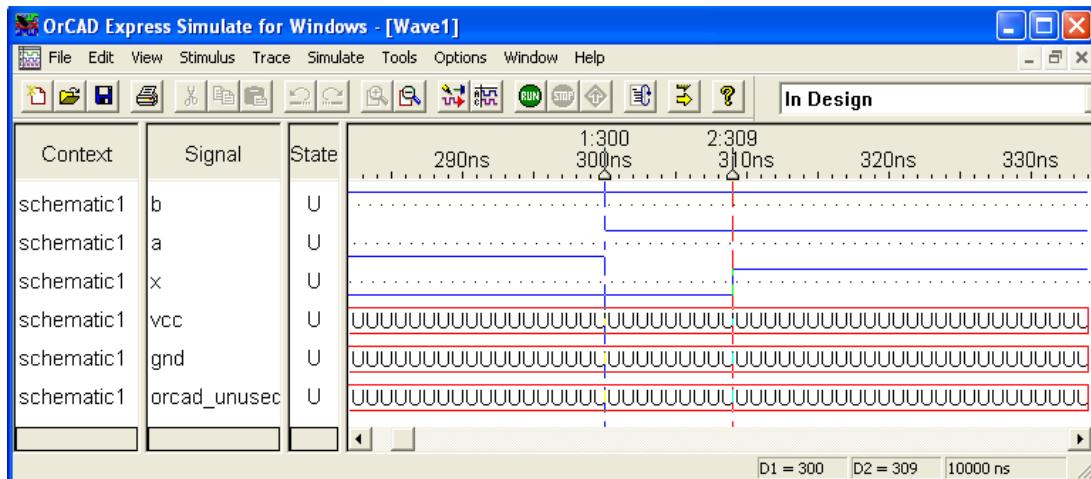


$$\Sigmaχήμα 1.2: \Delta t = 427 \text{ nsec} - 400 \text{ nsec} = 27 \text{ nsec}$$

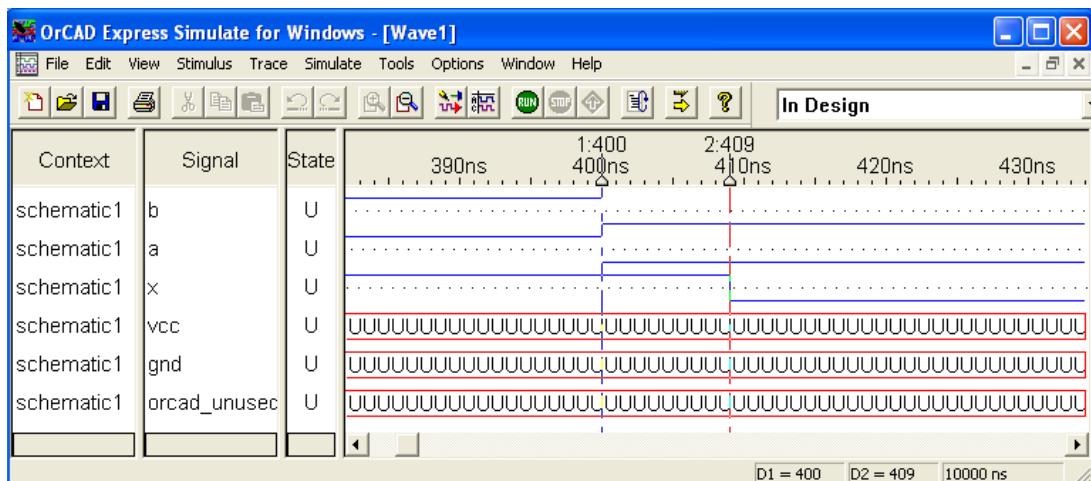
Πύλη AND 2 εισόδων 74ALS08 Στο Σχήμα 1.3 φαίνεται η μετάβαση εξόδου από χαμηλό δυναμικό (λογικό 0), σε υψηλό δυναμικό (λογικό 1) και οι αντίστοιχες χρονικές στιγμές μέσα στην εξομοίωση. Στο Σχήμα 1.4 υπάρχει όμοια η μετάβαση εξόδου από υψηλό σε χαμηλό δυναμικό μαζί με τις αντίστοιχες χρονικές στιγμές. Παρατηρείται πως και στις 2 μεταβάσεις η καθυστέρηση διάδοσης είναι ίδια και ίση με 9 nsec. Η οικογένεια *Advanced Low-Power Schottky* είναι μια από της οικογένειες διάδοχοι της *TTL Schottky*². Όπως και στην *Schottky TTL*, η κατανάλωση είναι χαμηλή και το *gain* σχετικά μεγαλύτερο [3]. Ετσι, ο χρόνος καθυστέρησης διάδοσης είναι χαμηλότερος των *standard TTL*. Η συγκεκριμένη

²H *TTL Schottky* έχει πλέον ξεπεραστεί

πύλη έχει και καλύτερες επιδόσεις. Υπάρχουν κατασκευαστές [2] που έχουν φτάσει χρόνους $t_{TLH} = t_{THL} = 2 \text{ nsec}$ ³.



Σχήμα 1.3: $\Delta t = 309 \text{ nsec} - 300 \text{ nsec} = 9 \text{ nsec}$

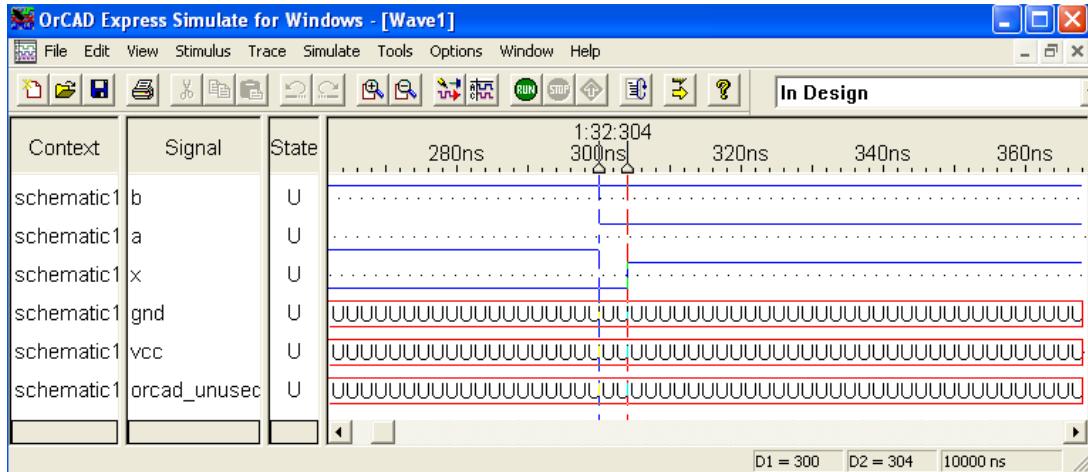


Σχήμα 1.4: $\Delta t = 409 \text{ nsec} - 400 \text{ nsec} = 9 \text{ nsec}$

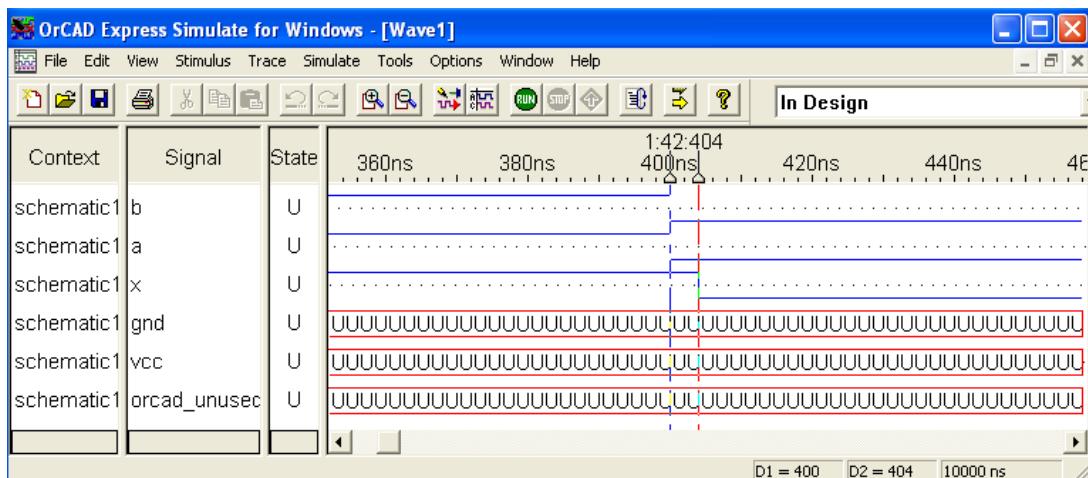
Πύλη AND 2 εισόδων 74AS08 Στο Σχήμα 1.5 φαίνεται η μετάβαση εξόδου από χαμηλό δυναμικό (λογικό 0), σε υψηλό δυναμικό (λογικό 1) και οι αντίστοιχες χρονικές στιγμές μέσα στην εξομοίωση. Στο Σχήμα 1.6 υπάρχει όμοια η μετάβαση εξόδου από υψηλό σε χαμηλό δυναμικό μαζί με τις αντίστοιχες χρονικές στιγμές. Η καθυστέρηση διάδοσης και στις 2 περιπτώσεις είναι η ίδια και ίση με 4 nsec. Παρατηρείται πως η οικογένεια των Advanced Schottky TTL chips προσφέρουν ακόμα χαμηλότερη καθυστέρηση καθώς ο σχεδιασμός τους

³Ο t_{TLH} και ο t_{THL} είναι ο χρόνος καθυστέρησης διάδοσης για τη μετάβαση από λογικό 0 σε λογικό 1, και ο χρόνος καθυστέρησης διάδοσης για τη μετάβαση από λογικό 1 σε λογικό 0 αντίστοιχα [4]

είναι βελτιστοποιημένος για να δίνει πολύ υψηλή ταχύτητα [3]. Ωστόσο, για την επίτευξη υψηλών ταχυτήτων, οι πύλες στη συγκεκριμένη οικογένεια έχουν μεγαλύτερη κατανάλωση από τις υπόλοιπες *Schottky TTL* οικογένειες.



Σχήμα 1.5: $\Delta t = 304 \text{ nsec} - 300 \text{ nsec} = 4 \text{ nsec}$

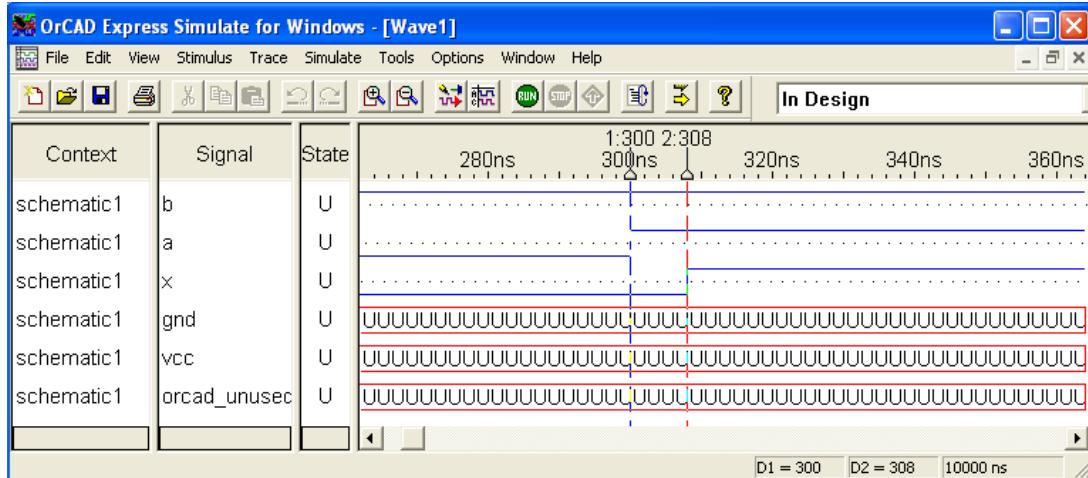


Σχήμα 1.6: $\Delta t = 404 \text{ nsec} - 400 \text{ nsec} = 4 \text{ nsec}$

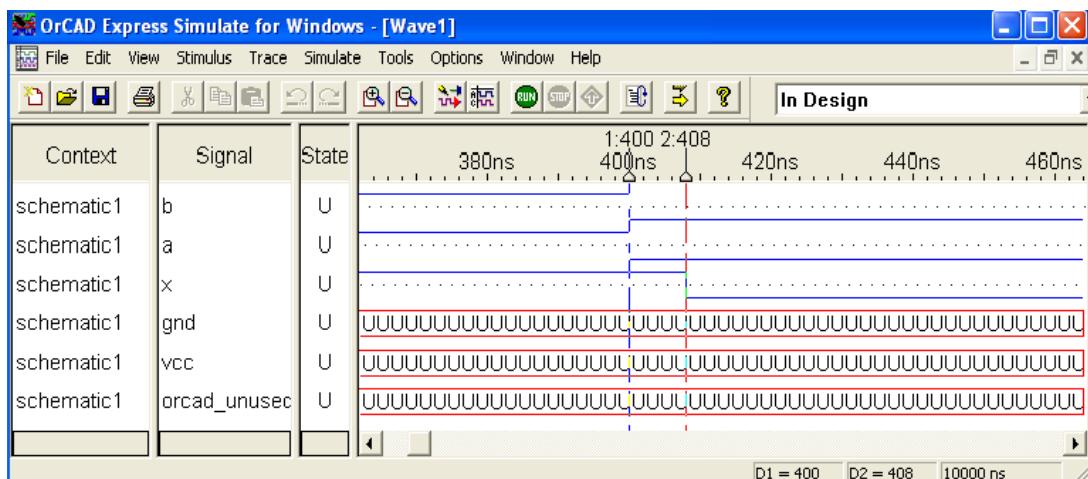
Πύλη AND 2 εισόδων 74S08 Στο Σχήμα 1.7 φαίνεται η μετάβαση εξόδου από χαμηλό δυναμικό (λογικό 0), σε υψηλό δυναμικό (λογικό 1) και οι αντίστοιχες χρονικές στιγμές μέσα στην εξομοίωση. Στο Σχήμα 1.8 υπάρχει όμοια η μετάβαση εξόδου από υψηλό σε χαμηλό δυναμικό μαζί με τις αντίστοιχες χρονικές στιγμές. Η καθυστέρηση διάδοσης και στις 2 περιπτώσεις είναι η ίδια και ίση με 8 nsec. Αν και ξεπερασμένη οικογένεια, η *Schottky TTL* έχει σαφώς χαμηλότερους χρόνους t_{TLH} και t_{TTL} από την οικογένεια πυλών *standard TTL*. Μάλιστα, όταν έγινε το λανσάρισμα της διόδου Schottky ⁴, οι ταχύτητες μνήμης

⁴Η διόδος Schottky είναι το θεμελιώδες στοιχείο των οικογενειών Schottky

διπλασιάστηκαν [6]. Επίσης, έχει χαρακτηριστικά που κληρονόμησαν έπειτα και οι οικογένειες LS, ALS και AS, όπως η συμβατότητα με τα περισσότερα TTL ολοκληρωμένα και την ανοχή λειτουργίας σε υψηλότερες θερμοκρασίες [5].

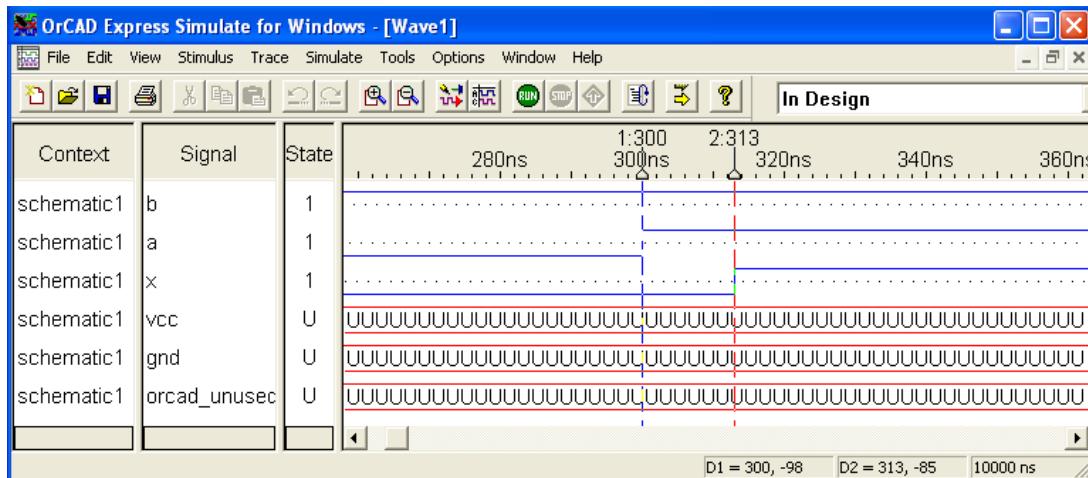


$$\Sigmaχήμα 1.7: \Delta t = 308 \text{ nsec} - 300 \text{ nsec} = 8 \text{ nsec}$$

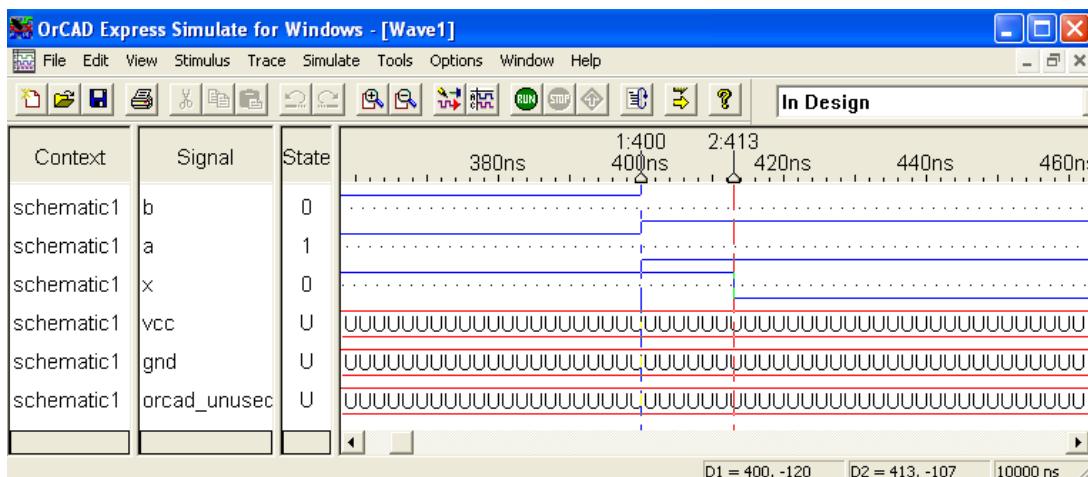


$$\Sigmaχήμα 1.8: \Delta t = 408 \text{ nsec} - 400 \text{ nsec} = 8 \text{ nsec}$$

Πύλη AND 2 εισόδων 74LS08 Στο Σχήμα 1.9 φαίνεται η μετάβαση εξόδου από χαμηλό δυναμικό (λογικό 0), σε υψηλό δυναμικό (λογικό 1) και οι αντίστοιχες χρονικές στιγμές μέσα στην εξομοίωση. Στο Σχήμα 1.10 υπάρχει όμοια η μετάβαση εξόδου από υψηλό σε χαμηλό δυναμικό μαζί με τις αντίστοιχες χρονικές στιγμές. Η καθυστέρηση διάδοσης και στις 2 περιπτώσεις είναι η ίδια και ίση με 13 nsec. Η οικογένεια *Low-Power Schottky TTL* είναι ο πρόγονος της ALS. Η LS έχει λίγο χειρότερο χρόνο από καθυστέρησης διάδοσης από την S. Ωστόσο, η κατανάλωση είναι λιγότερη σε σχέση με τις πύλες S.



$$\Sigmaχήμα 1.9: \Delta t = 313 \text{ nsec} - 300 \text{ nsec} = 13 \text{ nsec}$$



$$\Sigmaχήμα 1.10: \Delta t = 413 \text{ nsec} - 400 \text{ nsec} = 13 \text{ nsec}$$

1.2 Ζήτημα άσκησης

Από την παραπάνω ανάλυση, οι τιμές που αντιστοιχούν στον πίνακα που κλήθηκε να συμπληρώσει η ομάδα φαίνονται στον Πίνακα 1.1.

	7408	74ALS08	74AS08	74S08	74LS08
Χρόνος καθυστέρησης μετάβασης στο λογικό 0 (σε nsec)	27	9	4	8	13
Χρόνος καθυστέρησης μετάβασης στο λογικό 1 (σε nsec)	27	9	4	8	13

Πίνακας 1.1: Χρόνοι καθυστέρησης διάδοσης μερικών γνωστών TTL οικογενειών, στο πρόγραμμα OrCAD

Βιβλιογραφία

- [1] ElectroSchematics
DM7408
Quad 2-Input AND Gates
<https://www.electroschematics.com/wp-content/uploads/2013/07/7408-datasheet.pdf>
- [2] Philips Semiconductors
IC05 Data Handbook
<http://pdf.datasheetcatalog.com/datasheet/philiips/74ALS08N.pdf>
- [3] Nuts and Volts
Understanding digital logic ICS — part 2
By Ray Marston
https://www.nutsvolts.com/magazine/article/understanding_digital-logic_ics_part_2
- [4] Toshiba Electronic Devices & Storage Corporation
C²MOSTM LOGIC IC Product Guide STD Series
2018-09-01
<https://toshiba.semicon-storage.com/info/docget.jsp?did=63520>
- [5] Texas Instruments
Logic Guide
2017
<http://www.ti.com/lit/sg/sdyu001ab/sdyu001ab.pdf>
- [6] Computer History Museum
1969: SCHOTTKY-BARRIER DIODE DOUBLES THE SPEED OF TTL MEMORY & LOGIC
<https://www.computerhistory.org/siliconengine/schottky-barrier-diode-doubles-the-speed-of-ttl-memory-logic/>

Δεύτερη εργαστηριακή άσκηση

Ένας σχεδιαστής θα πρέπει να γνωρίζει τις 2 βασικότερες μεθοδολογίες σχεδιασμού ολοκληρωμένων [1] .

Top-down Η πρώτη μέθοδος είναι η μέθοδος *top-down* κατά την οποία ο σχεδιαστής ξεκινάει από ένα μαύρο κουτί, το οποίο συμβολίζει το ολοκληρωμένο που θέλει να φτιάξει. Το μαύρο αυτό κουτί ο σχεδιαστής μπορεί να το περιγράψει με γλώσσες υλικού, όπως η [Verilog](#). Ωστόσο, μπορεί να ξεκινήσει κι από πιο χαμηλά, όπως συμβαίνει και στην παρούσα άσκηση. Από όποιο επίπεδο σχεδιασμού και να ξεκινήσει όμως, αν ακολουθεί τη μεθοδολογία σχεδιασμού *top-bottom*, μπορεί μόνο να πηγαίνει σε χαμηλότερα επίπεδα σχεδιασμού.

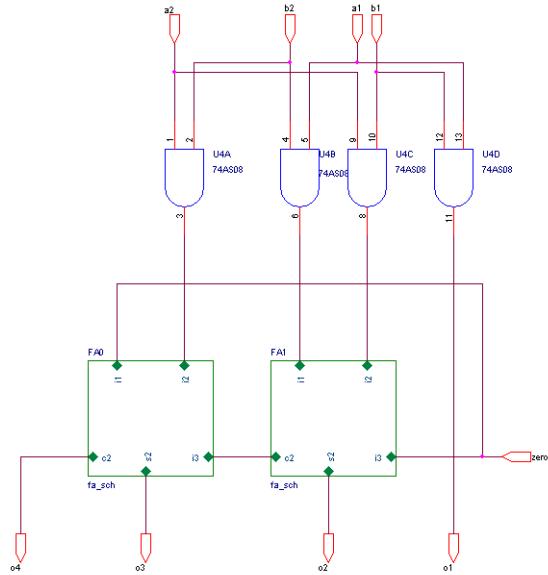
Bottom-up Το αντίθετο συμβαίνει στη μεθοδολογία σχεδιασμού *bottom-up*. Εδώ, ο σχεδιαστής ξεκινάει από το επίπεδο των *transistors*, συνεχίζοντας σε υψηλότερα επίπεδα. Όμοια με πριν, ο σχεδιαστής μπορεί να ξεκινήσει και από κάποιο ανώτερο επίπεδο, όπως συμβαίνει και στην άσκηση, όπου το ανώτερο επίπεδο είναι οι πύλες. Χρησιμοποιώντας τις πύλες προκύπτει από την άσκηση ένα σύστημα: *to d flip-flop*.

Πλεονεκτήματα και μειονεκτήματα Είναι απλό για ένα σχεδιαστή να αποφασίσει μια από τις δύο μεθόδους. Στην *top-down* ο σχεδιαστής μπορεί να ολοκληρώσει πιο γρήγορα το σχεδιασμό του. Ωστόσο, υπάρχει ο περιορισμός των εργαλείων που του παρέχονται. Επίσης, επιλέγει ο ίδιος αν επιθυμεί να αλλάξει τη δομή ενός ολοκληρωμένου ή μιας πύλης. Οπότε, η μέθοδος αυτή μπορεί να χρησιμοποιηθεί για "*rapid*" *designing* όταν ο σχεδιαστής καλείται να φτιάξει ένα μεγάλης κλίμακας κύκλωμα. Αντίθετα, η μέθοδος *bottom-up* χρησιμοποιείται για μικρής κλίμακας σχέδια, καθώς είναι μια αρκετά αργή μέθοδος. Ο σχεδιαστής προτιμά τη συγκεκριμένη μέθοδο όταν τα ολοκληρωμένα που υπάρχουν στο εμπόριο δε μπορούν να καλύψουν τις ανάγκες του, ή όταν ο ίδιος έχει μια καινοτόμα ιδέα όπου θα πρέπει να ορίσει από την αρχή μια βάση.

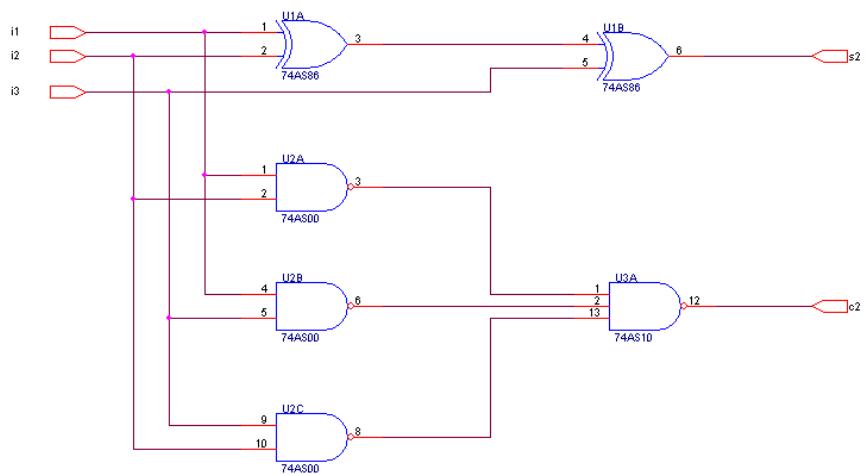
2.1 Top-down design

Για το πρώτο μέρος της άσκησης, σχεδιάστηκε ένας *carry-save* πολλαπλασιαστής δύο bits, όπως φαίνεται στο Σχήμα 2.1. Το μαύρο κουτί εδώ είναι ο full adder. Ακολουθώντας τις οδηγίες της εκφώνησης, σχεδιάστηκε ο *full adder*, όπως φαίνεται στο Σχήμα 2.2. Για την επαλήθευση της ορθότητας του σχεδιασμού του *carry-save* πολλαπλασιαστή, το κύκλωμα εξομοιώθηκε στο OrCAD σύμφωνα με το *stimulus file* που φαίνεται στον Πίνακα 2.1. Πριν την εξομοίωση ωστόσο, θα πρέπει να οριστεί ο Πίνακας 2.2, που είναι ο πίνακας αληθείας του *carry-save* πολλαπλασιαστή. Αφού ορίστηκε κι ο πίνακας αληθείας μπορεί να επαληθευτεί η ορθότητα του σχεδιασμού. Τα αποτελέσματα της εξομοίωσης βρίσκονται στο Σχήμα 2.3.

Παρατηρείται πως οι κυματομορφές στην εξομοίωση που έγινε είναι σωστές σύμφωνα με τον Πίνακα 2.2. Άρα, ο σχεδιασμός του *carry-save* πολλαπλασιαστή είναι ορθός.



Σχήμα 2.1: 2-Input Carry-save multiplier

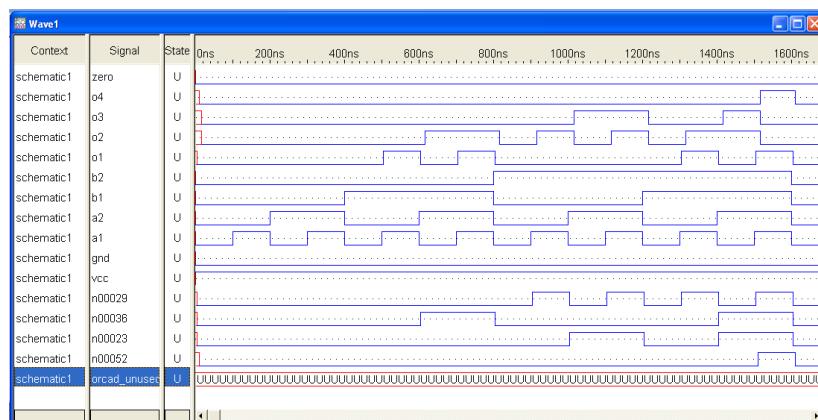


Σχήμα 2.2: Full adder design

Σήμα	Τύπος	Χρονισμός
GND	absolute	set to 0 at time 0
VCC	absolute	set to 1 at time 0
zero	absolute	set to 0 at time 0
a1	clock	set to 0 for 100ns; set to 1 for 100ns
a2	clock	set to 0 for 200ns; set to 1 for 200ns
b1	clock	set to 0 for 400ns; set to 1 for 400ns
b2	clock	set to 0 for 800ns; set to 1 for 800ns

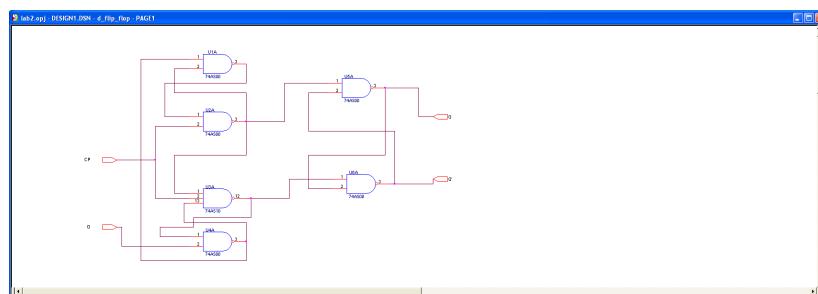
Table 2.1: *Stimulus file* για την εξομοίωση του *carry-save* πολλαπλασιαστή

b2	b1	a2	a1	o4	o3	o2	o1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

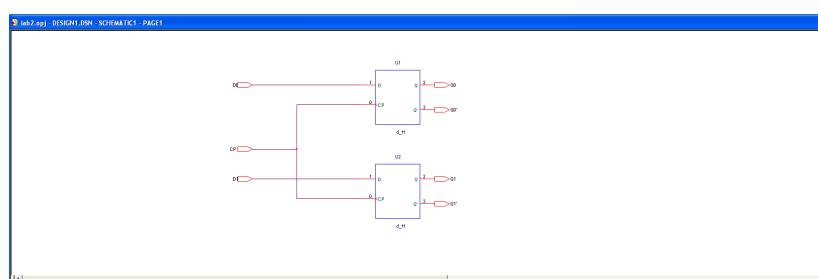
Table 2.2: Πίνακας αληθείας του *carry-save* πολλαπλασιαστήΣχήμα 2.3: Εξομοίωση του *carry-save* πολλαπλασιαστή

2.2 Bottom-up design

Για το δεύτερο μέρος της άσκησης, σχεδιάστηκε ένα *D flip-flop*, όπως φαίνεται στο Σχήμα 2.4. Χρησιμοποιώντας λογικές πύλες έγινε ο custom σχεδιασμός του *D flip-flop* που ζητήθηκε στην άσκηση. Το ανώτερο επίπεδο στην περίπτωση της άσκησης, είναι η χρήση του *D flip-flop* για τη σχεδίαση ενός καταχωρητή των 2 bits. Χρησιμοποιώντας 2 *D flip-flop* οπότε, υλοποιείται ο ζητούμενος καταχωρητής των 2 bits, όπως φαίνεται στο Σχήμα 2.5. Για να επαληθευτεί η ορθότητα του σχεδιασμού έγινε εξομοίωση του καταχωρητή. Το *stimulus file* που δημιουργήθηκε για την εξομοίωση παρουσιάζεται στον Πίνακα 2.3. Ο πίνακας αληθείας του *D flip-flop* είναι στον Πίνακα 2.4. Λόγω του απλού σχεδιασμού του ζητούμενου καταχωρητή, δεν έγινε πίνακας αληθείας για τον καταχωρητή.



Σχήμα 2.4: Ο σχεδιασμός του *D flip-flop*



Σχήμα 2.5: Ο σχεδιασμός του καταχωρητή χωρητικότητας 2 bit

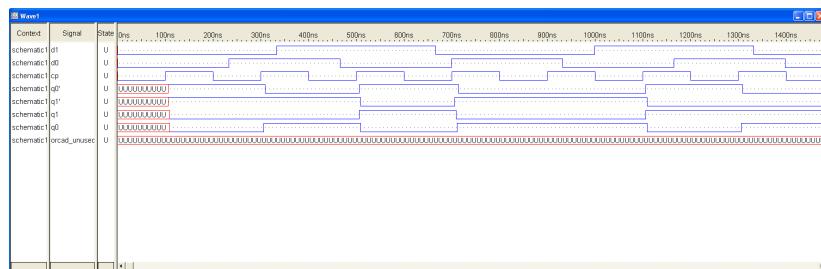
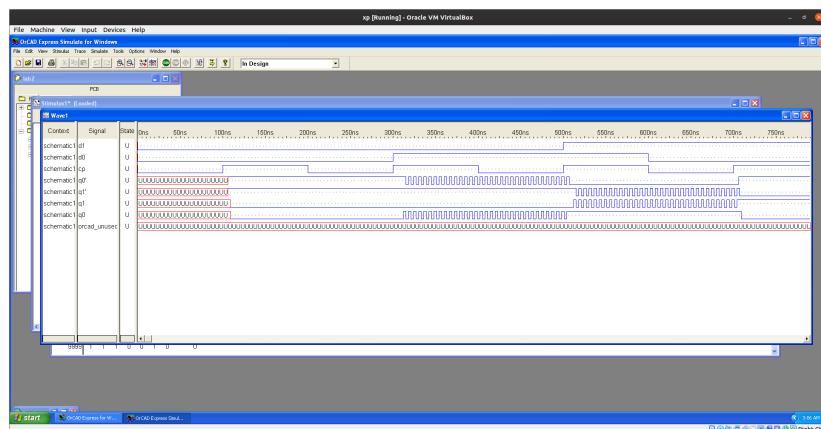
Σήμα	Τύπος	Χρονισμός
CP	clock	At 0, set to 0 for 100ns; set to 1 for 100ns, repeat forever
D0	clock	At 0, set to 0 for 233ns; set to 1 for 233ns, repeat forever
D1	clock	At 0, set to 0 for 333ns; set to 1 for 333ns, repeat forever

Table 2.3: *Stimulus file* για την εξομοίωση του καταχωρητή 2 bits

CP	D	Q	Q'	Description
0	X	Q	Q'	Memory no change
1	0	0	1	Reset Q $\leftarrow 0$
1	1	1	0	Set Q $\leftarrow 0$

Table 2.4: Πίνακας αληθείας του *D flip-flop* [2]

Το αποτέλεσμα της εξομοίωσης φαίνεται στο Σχήμα 2.6. Οι χρόνοι στον Πίνακα 2.4 επιλέχθηκαν έτσι ώστε να τηρείται το T_{setup} και το T_{hold} [3]. Ο χρόνος T_{setup} είναι ο ελάχιστος χρόνος που ψά πρέπει το σήμα *D* να παραμείνει σταθερό πριν το σήμα *CP*, δηλαδή το ρολόι του *D flip-flop*, αλλάξει κατάσταση. Ο χρόνος T_{hold} είναι ο ελάχιστος χρόνος που ψά πρέπει το σήμα *D* να μην αλλάξει κατάσταση μετά από αλλαγή κατάστασης στο σήμα *CP*. Σε περίπτωση που οι χρόνοι αυτοί δεν τηρηθούν, το κύκλωμα εισέρχεται σε κατάσταση αστάθειας, όπως φαίνεται στο Σχήμα 2.7.

Σχήμα 2.6: Οι κυματομορφές μετά από την εξομοίωση του *D flip-flop*Σχήμα 2.7: Οι κυματομορφές μετά από λάθος εξομοίωση του *D flip-flop*

Βιβλιογραφία

[1] EE Times

A Formal Top-Down Design Process for Mixed-Signal Circuits

By Ken Kundert, Fellow, Cadence Design Systems, San Jose, Calif

10-04-2000

<https://www.eetimes.com/a-formal-top-down-design-process-for-mixed-signal-circuits/>

[2] Electronics Tutorials

The D-type Flip Flop

https://www.electronics-tutorials.ws/sequential/seq_4.html

[3] VLSI Expert

Setup and Hold Time: Static Time Analysis (STA) basic (Part 3a)

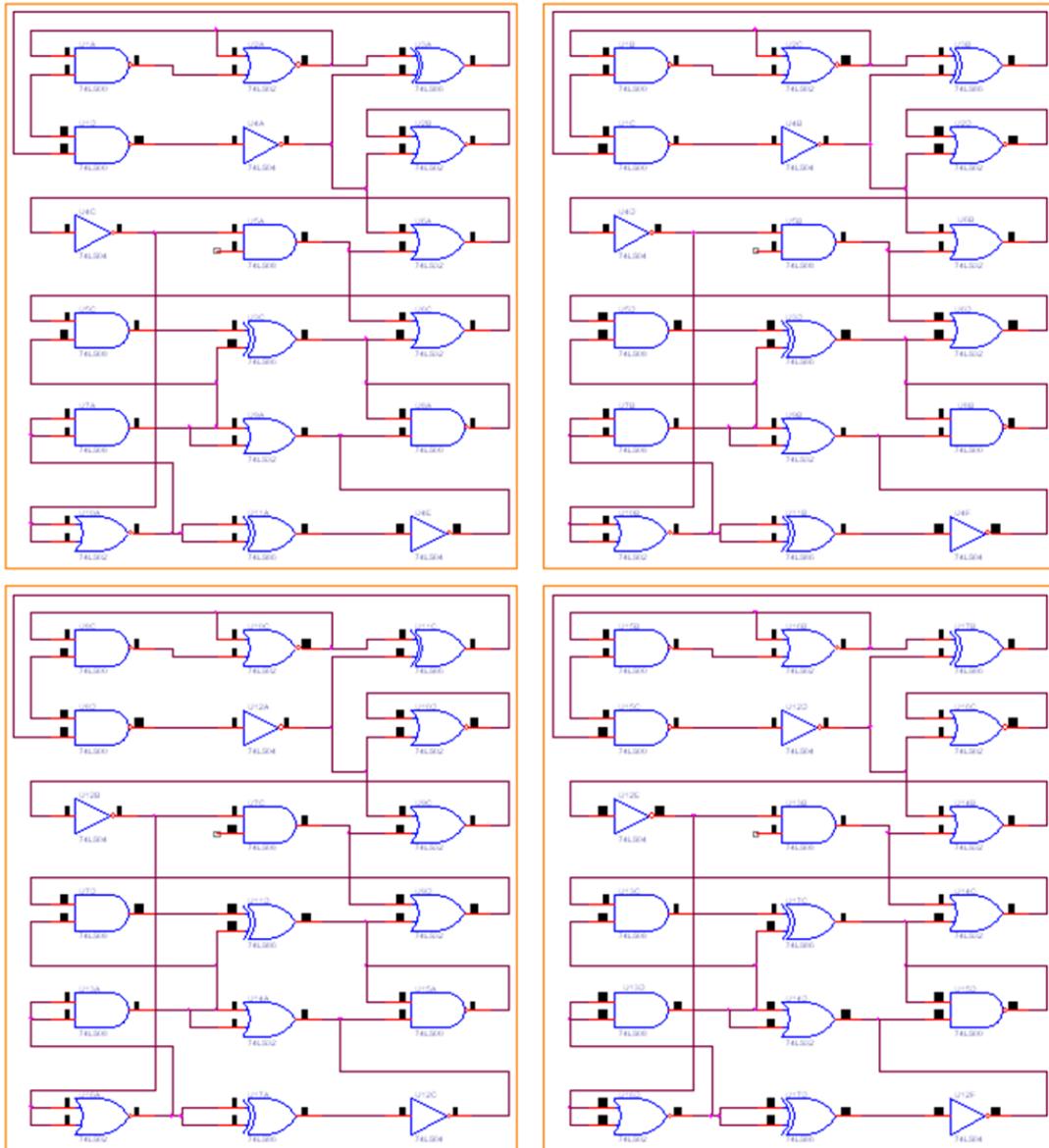
07-04-2011

<http://www.vlsi-expert.com/2011/04/static-timing-analysis-sta-basic-part3a.html>

Τρίτη εργαστηριακή άσκηση

3.1 Σχεδιαστικά κύτταρα

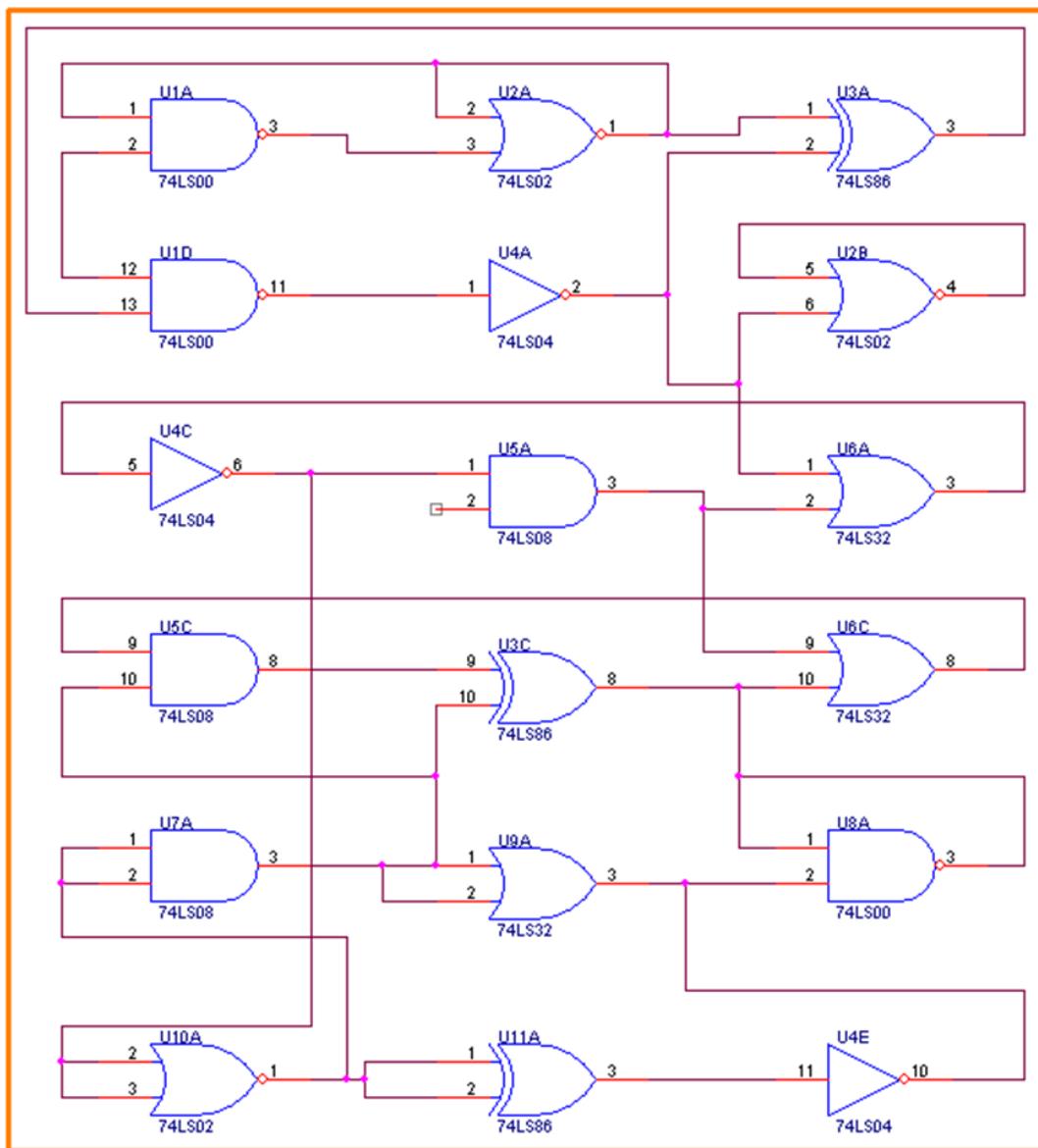
Τα τέσσερα σχεδιαστικά κύτταρα για το κύκλωμα της άσκησης φαίνονται στην εικόνα 3.1.



Σχήμα 3.1: Αρχιτεκτονική κυκλώματος

Το κάθε σχεδιαστικό κύτταρο έχει τη μορφή που φαίνεται στην εικόνα 3.2.

Το αρχείο netlist που δημιουργήθηκε είναι το DESIGN_ASK3.MNL και με βάση αυτό έγινε η



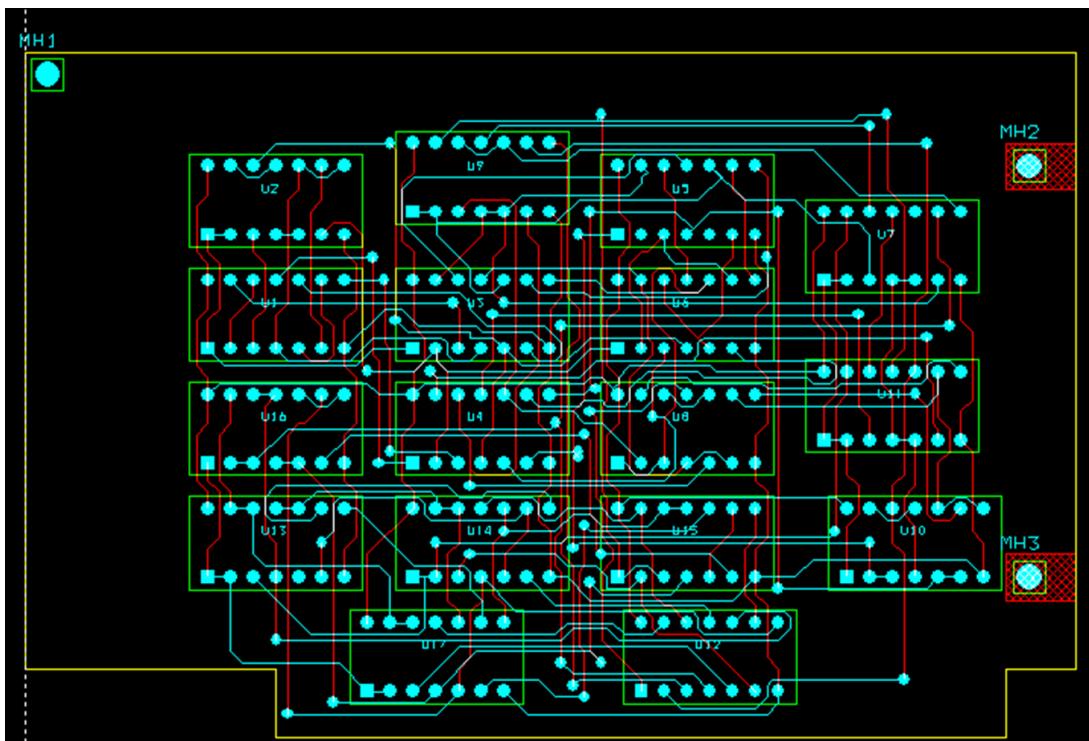
Σχήμα 3.2: Αρχιτεκτονική κυττάρου

διάταξη των ολοκληρωμένων κυκλωμάτων στο χώρο με βάση τη βιβλιοθήκη tutor.tch.

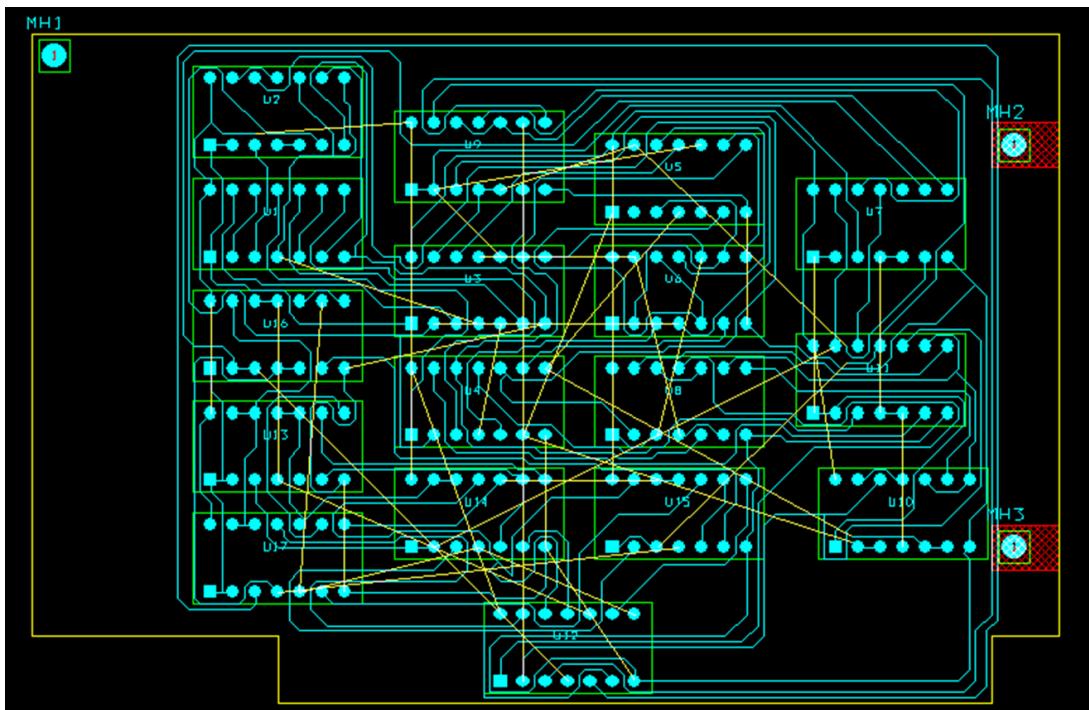
3.2 Η πλακέτα

Η πλακέτα των δύο επιπέδων φαίνεται στην Εικόνα 3.3, όπου με κόκκινο χρώμα φαίνονται τα καλώδια στο επίπεδο BOTTOM, ενώ με μπλε τα καλώδια του TOP.

Για να γίνει διασύνδεση των στοιχείων μόνο στο ένα επίπεδο της πλακέτας, επιλέχθηκε το Bottom Layer να μη χρησιμοποιείται για routing. Η διαδρόμηση ενός μόνο επιπέδου παρουσιάζεται στην Εικόνα 3.4. Με γαλάζιο χρώμα φαίνονται οι έγκυρες διασυνδέσεις, ενώ με κίτρινο είναι οι συνδέσεις που δεν μπόρεσαν να πραγματοποιηθούν.



Σχήμα 3.3: Η πλακέτα

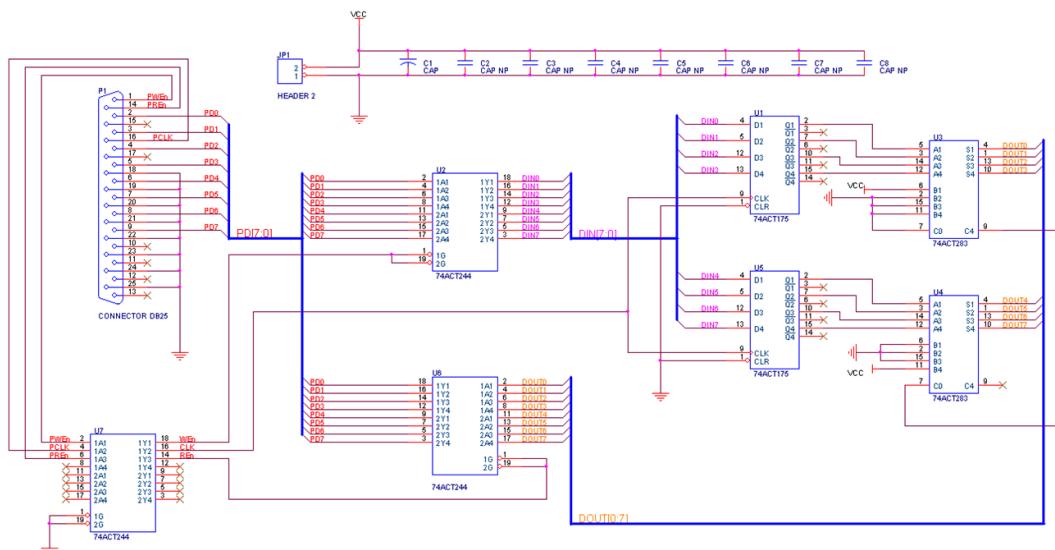


Σχήμα 3.4: Διασύνδεση στοιχείων

Τέταρτη εργαστηριακή άσκηση

4.1 Το κύκλωμα hasp

Σε αυτή την άσκηση κατασκευάστηκε το σχηματικό ενός κυκλώματος hasp, όπως φαίνεται στην εικόνα 4.1.



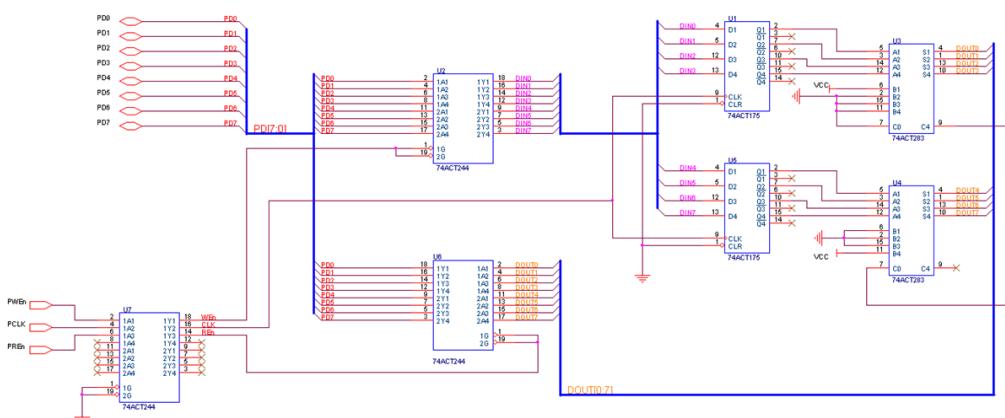
Σχήμα 4.1: Το κύκλωμα hasp

Για το γραφικό σχεδιασμό του κυκλώματος χρησιμοποιήθηκαν οι βιβλιοθήκες TTL για τους απομονωτές 74ACT244 και η DEVICE για τους πυκνωτές (CAPACITOR, CAPACITOR NON-POL) καθώς και για τους connectors (CONNECTOR DB25, HEADER 2). Τα σήματα γείωσης (GND) και τάσης (VCC) προέρχονται από τη βιβλιοθήκη CAPSYM. Όταν το λογισμικό τοποθετήσει το byte στην παράλληλη θύρα, με την ενεργοποίηση των κατάλληλων σημάτων ελέγχου το byte αποθηκεύεται στους 4-bit καταχωρητές 74ACT175 και μετά περνά στους 4-bit adders 74ACT283. Εκεί, ουσιαστικά γίνεται πρόσθεση με τον αριθμό $100000012 = -12710$ σε αναπαράσταση 2's complement. Αυτό συμβαίνει γιατί σε αναπαράσταση Excess-127, ο αναπαριστάμενος αριθμός είναι (αρχικός αριθμός) - 127. Στο σχηματικό έχει προστεθεί ένας πυκνωτής τανταλίου και 7 decoupling πυκνωτές, οι οποίοι συνδέονται με τροφοδοσία και γείωση μέσω της υποδοχής HEADER 2. Οι πυκνωτές τανταλίου είναι πολωμένοι και διακρίνονται για τη μεγάλη χωρητικότητα που έχουν σε μικρό μέγεθος διαστάσεων καθώς και για τη σταθερότητα που παρουσιάζουν στη διατήρηση της χωρητικότητάς τους σε υψηλές θερμοκρασίες και τάσεις. Γι' αυτό εδώ ο πυκνωτής τανταλίου σε συνεργασία με τους 7 σταθερούς κεραμικούς πυκνωτές λειτουργούν ως decoupling

capacitors. Συγκεκριμένα, τους συνδέουμε παράλληλα, στο ένα άκρο με τη γείωση (GND) και στο άλλο με την τροφοδοσία (VCC) ενώ τους τοποθετούμε σε χοντινή απόσταση από το κύκλωμα που σχεδιάζουμε. Όταν θέτουμε σε λειτουργία το κύκλωμα, δηλαδή εφαρμόζουμε $VCC = 5V$ ή $3.3V$, οι πυκνωτές φορτίζονται οπότε σε περίπτωση που κάποια στιγμή υπάρξει αλλαγή στην τάση, τότε οι decoupling capacitors θα διασφαλίσουν ότι το κύκλωμα θα διατηρήσει τη σταθερότητά του. Συγκεκριμένα, σε περίπτωση που η τάση εισόδου μειωθεί, ο πυκνωτής τανταλίου αποφορτίζεται ώστε να παρέχει την απαιτούμενη ηλεκτρική ενέργεια στο κύκλωμα. Όταν όμως η τροφοδοσία αυξηθεί, τότε οι κεραμικοί σταθεροί πυκνωτές απορροφούν το πλεόνασμα ηλεκτρικής ενέργειας που διαπερνά το κύκλωμα ώστε η τάση να παραμένει σταθερή. Έτσι, επιτυγχάνεται εξομάλυνση της τάσης εισόδου του κυκλώματος, χωρίς διακοπές.

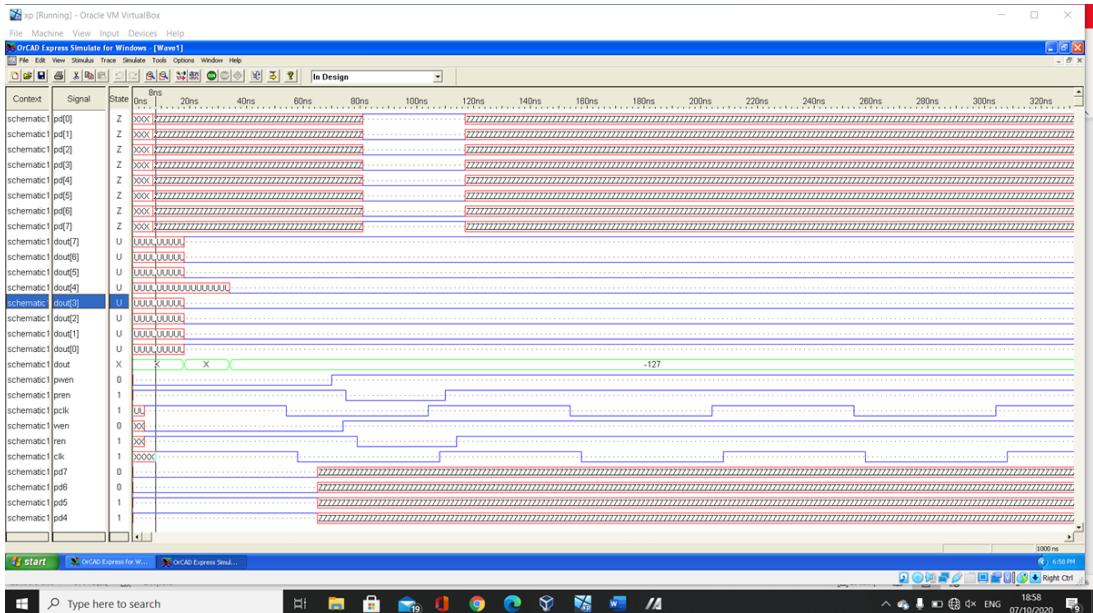
4.2 Εξομοίωση του σχεδιασμού

Για να πραγματοποιηθεί εξομοίωση του σχεδιασμού έγινε αντικατάσταση της παράλληλης θύρας με bidirectional hierarchical ports. Το νέο schematic φαίνεται στην εικόνα 4.2.

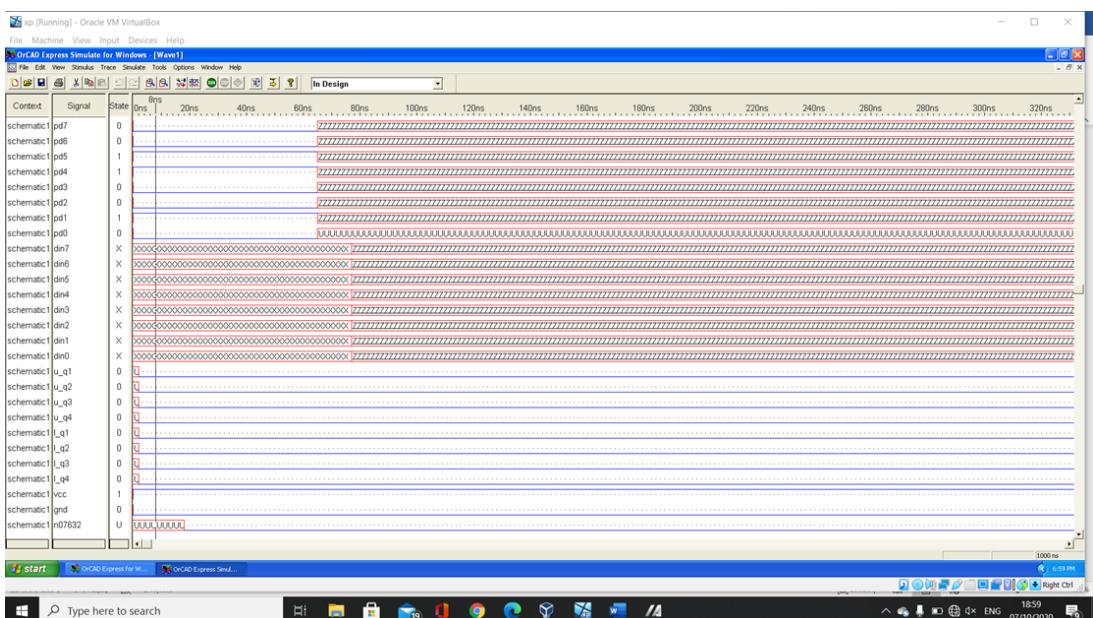


Σχήμα 4.2: Το νέο schematic

Στο αρχείο stimulus δόθηκαν τιμές στα σήματα PWE_n , PRE_n και $PCLK$ ως εξής. Το ρολόι $PCLK$ θα έχει περίοδο 100 ns (50 ns ON, 50 ns OFF), το PWE_n είναι ενεργοποιημένο για 65 ns από την αρχή και το PRE_n ενεργοποιείται στα 75 ns μέχρι τα 110 ns για να διαβαστούν τα δεδομένα από τα bidirectional ports. Επίσης, στα ports διπλής κατεύθυνσης δόθηκαν σήματα εισόδου τύπου absolute με τιμές $PD[7:0] = 00110010$ από την αρχή (0 ns) έως τα 65 ns . Από τα 65 ns και μετά τα σήματα $PD[7:0]$ τέθηκαν σε κατάσταση υψηλής εμπέδησης Z. Τέλος, στα σήματα vcc και gnd δόθηκαν τιμές 1 και 0 αντίστοιχα.



Σχήμα 4.3: Κυματομορφή εξομοίωσης



Σχήμα 4.4: Κυματομορφή εξομοίωσης - συνέχεια

Παρατηρήθηκε ότι η καθυστέρηση που προσθέτουν τα OK 74ACT244 είναι 4 ns καθώς τόσο αργούν να διαδοθούν τα σήματα PWE_n , PRE_n και $PCLK$ στις εξόδους του chip. Άρα, τη χρονική στιγμή 4 ns όπου ενεργοποιηθεί το 74ACT244 που θα προωθήσει τα δεδομένα προς τα D-ff. Αυτό το ολοκληρωμένο κύκλωμα με τη σειρά του θα καθυστερήσει κατά 4 ns τα δεδομένα προς την έξοδό του. Μέχρι στιγμής η συνολική καθυστέρηση είναι 8 ns και επειδή τα δεδομένα κλειδώνονται στα flip - flops με την ανοδική ακμή, αρκεί να θέσουμε μια θετική ακμή του ρολογιού τη χρονική στιγμή 8 ns . Για να γίνει αυτό θα πρέπει το σήμα PCLK να γίνει από 0 σε 1 τη στιγμή $8\text{ ns} - 4\text{ ns} = 4\text{ ns}$ καθώς και

αυτό θα επιβαρυνθεί από την καθυστέρηση του 74ACT244. Έπειτα, θα γίνει η πρόσθεση στα 74ACT283 τα οποία προσθέτουν τη δική τους καθυστέρηση και το αποτέλεσμα του αριθμού με κωδικοποίηση excess-127 θα οδηγηθεί στον tri-state buffer 74ACT244. Εκεί τα δεδομένα θα καθυστερήσουν κατά 4 ns και θα είναι διαθέσιμα στα bidirectional ports $PD[7 : 0]$ εφόσον το σήμα PWEn είναι απενεργοποιημένο και το PREn ενεργοποιημένο. Η μέγιστη συχνότητα λειτουργίας του κυκλώματος υπολογίζεται με βάση την ελάχιστη περίοδο λειτουργίας του. Η T_{min} είναι όσο και το άνθροισμα των επιμέρους καθυστερήσεων, δηλαδή:

$$(καθυστέρηση \ 74ACT244) + (καθυστέρηση \ 74ACT175) + \\ (καθυστέρηση \ 74ACT283) + (καθυστέρηση \ 74ACT244)$$

4.3 Φυσική Υλοποίηση του σχεδιασμού

Η φυσική υλοποίηση δεν προχώρησε καθώς δεν καταφέραμε σαν ομάδα να βρούμε πώς μπορεί να δημιουργηθεί netlist μέσω του εργαλείου OrCAD LAYOUT, μετά την εισαγωγή των footprints για τα ολοκληρωμένα κυκλώματα του σχεδιασμού.

Βιβλιογραφία

- [1] Autodesk
What Are Decoupling Capacitors?
Sam Sattel
<https://www.autodesk.com/products/eagle/blog/what-are-decoupling-capacitors/>
- [2] Arrow
The Ups and Downs of Tantalum Capacitors
07-12-2016
<https://www.arrow.com/en/research-and-events/articles/the-ups-and-downs-of-tantalum-capacitors>

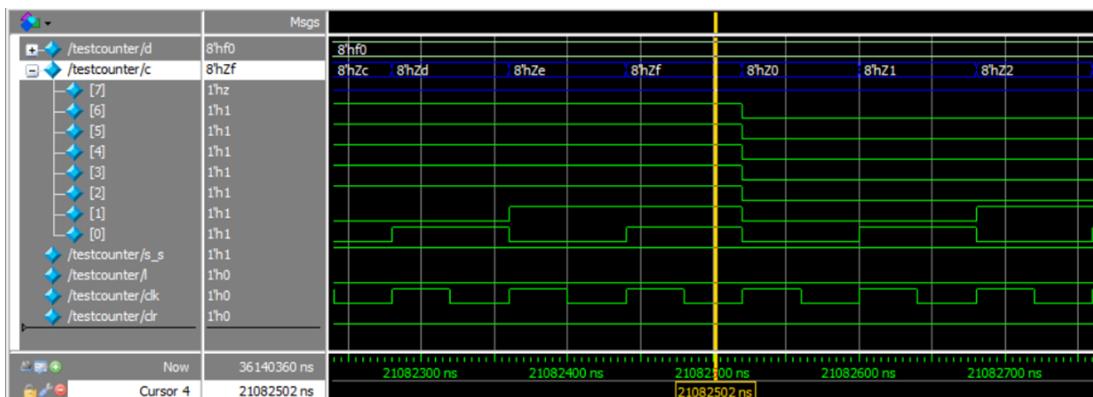
Πέμπτη εργαστηριακή άσκηση

5.1 Εισαγωγή

Για την πέμπτη άσκηση του μαθήματος χρησιμοποιήθηκε το εργαλείο *ModelSim PE Student Edition 10.4a*, όπως διατίθεται από την ιστοσελίδα της Mentor Graphics. Τα αρχεία που δημιουργήθηκαν είναι τα εξής:

- `counter.v` και `testcounter.v` για την εξοικείωση με το εργαλείο
- `counter_4.v` για το πρώτο ζητούμενο
- `counter_8.v` για το δεύτερο ζητούμενο
- `testcounter_8.v` για το τρίτο ζητούμενο
- `countersyn_4.v`, `countersyn_8.v` και `testcountersyn_8.v` για το πέμπτο ζητούμενο
- `counterdel_4.v`, `counterdel_8.v` και `testcounterdel_8.v` για το έκτο ζητούμενο

Αρχικά, έγινε εξομοίωση με τα δούλευτα αρχεία `counter.v` και `testcounter.v`. Στο test-bench, παρατηρήσαμε ότι τα δεδομένα εισόδου και εξόδου ήταν των 8 και όχι των 7 bits. Έτσι, στις κυματομορφές εξόδου το πιο σημαντικό bit στην έξοδο είναι σε κατάσταση υψηλής εμπέδησης. Ο μετρητής λειτουργεί σωστά, καθώς μετρά προς τα πάνω και εκτελεί ορθά τη διαδικασία της φόρτωσης δεδομένων μόλις του ζητηθεί. Αφ' ότου φτάσει την τιμή 127, δηλαδή τη μέγιστη δυνατή τιμή για αναπαράσταση 7 bits, μετά επανέρχεται στην τιμή 0 και συνεχίζει τη μέτρηση. Η συγκεκριμένη αλλαγή φαίνεται στην εικόνα 5.1.



5.2 Υλοποίηση μετρητή των 4 bits

Η τροποποίηση που έγινε στο design του μετρητή είναι η αλλαγή του πλήθους των bits των διανυσμάτων data και count, ώστε να περιέχουν 4 bits αντί για 7 που είχαν πριν. Το νέο αρχείο counter_4.v είναι στον κώδικα 5.1.

Κώδικας 5.1

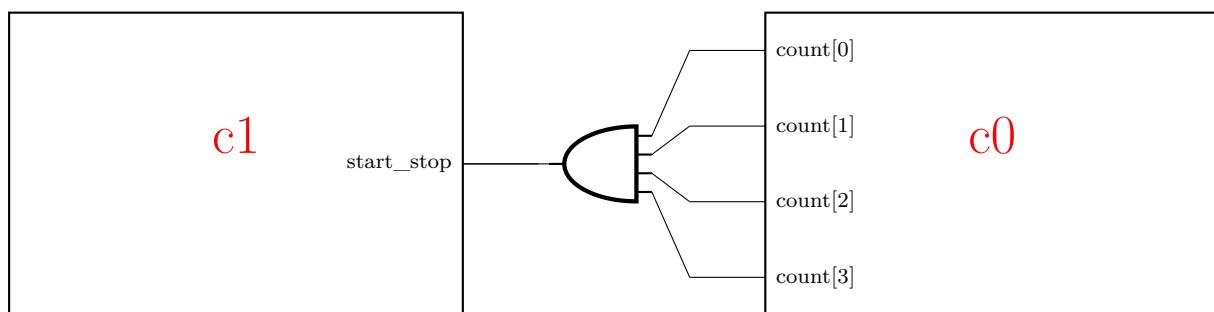
```

1 module counter4 (clear, clock, load, start_stop, count, data);
2   input [3:0] data;
3   output [3:0] count;
4   input start_stop;
5   input load;
6   input clock;
7   input clear;
8   reg [3:0] count;
9
10  always @ (posedge clock or posedge clear)
11    if (clear) count <= 0;
12    else if (load) count <= data;
13    else if (start_stop) count <= count + 1;
14 endmodule

```

5.3 Υλοποίηση μετρητή των 8 bits

Από τους δύο 4-bit counters που θα χρησιμοποιηθούν, ο ένας θα παράγει τα 4 λιγότερο σημαντικά bits (c0) και ο δεύτερος τα 4 περισσότερο σημαντικά (c1). Ο μετρητής counter1 θα πρέπει να ενεργοποιηθεί τη στιγμή που ο counter0 έχει σαν έξοδο την τετράδα 1111_b . Έτσι, αρκεί το σήμα επίτρεψης λειτουργίας του high order nibble να είναι μια πύλη AND τεσσάρων εισόδων, δηλαδή των τιμών count[0], count[1], count[2], count[3] του counter0. Αυτό το σήμα θα συνδεθεί στην είσοδο start_stop του counter1, ο οποίος θα αρχίσει να μετράει από την τιμή 0000_b . Αυτή η λογική φαίνεται σχηματικά στην εικόνα 5.2.



Σχήμα 5.2: Λογική που προστέθηκε για τον 8 - bit counter

Το λογισμικό για το design του μετρητή των 8 bits είναι στον κώδικα 5.2.

Κώδικας 5.2

```

1 module counter8 (clear, clock, load, start_stop, count, data);
2   input [7:0] data;
3   output [7:0] count;
4   input start_stop0;
5   input load;
6   input clock;
7   input clear;
8   wire start_stop1;
9
10  assign start_stop1 = (count[0] && count[1] && count[2] && count
11    ↪ [3]);
12
13 //Dhmiourgia instances twn metrhtwn 4-bit
14 counter4 c0(clear, clock, load, start_stop0, count[3:0], data[3
15    ↪ :0]);
16 counter4 c1(clear, clock, load, start_stop1, count[7:4], data[7
17    ↪ :4]);
18 endmodule

```

Εδώ το σήμα επίτρεψης start_stop1 του c1 είναι εσωτερικό wire. Επίσης δημιουργήθηκαν τα δύο στιγμιότυπα (c0 και c1) του 4-bit counter μέσω του module *counter4*.

5.4 Δημιουργία συνόλου διανυσμάτων εξομοίωσης

Για τη δημιουργία του συνόλου δοκιμής του 8-bit counter (*testcounter_8.v*) θα βασιστούμε στο αρχείο *testcounter.v*. Οι αλλαγές που έγιναν είναι η αλλαγή ονόματος του module του testbench και η δημιουργία στιγμιότυπου του module *counter8* αντί του *counter*. Επίσης, τα δεδομένα που προορίζονται για παράλληλη φόρτωση έχουν πλέον την τιμή 50_h , για λόγους διευκόλυνσης της επαλήθευσης των αποτελεσμάτων, καθώς παρατηρήθηκε ότι τη στιγμή που το σήμα *s_s* ενεργοποιείται, ο μετρητής έχει φτάσει στην τιμή 53_h . Το αρχείο *testcounter_8.v* φαίνεται στον κώδικα 5.3.

Κώδικας 5.3

```

1 module testcounter_8 ();
2   reg [7:0] d;
3   wire [7:0] c;
4   reg s_s, l, clk, clr;
5
6   counter8 inst0 (clr, clk, l, s_s, c, d);
7
8   initial
9     begin
10      # 5 clk <= 0; clr <= 0; l <= 0; s_s <= 0; d <= 8'h50;
11    end
12   always # 40 clk <= !clk;

```

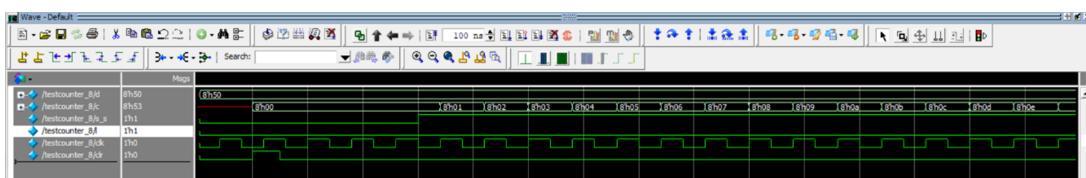
```

13 initial
14 begin
15     # 100 clr <= 1;
16     # 50 clr <= 0;
17 end
18 initial
19 begin
20     # 400 s_s <= 1;
21     # 3000 s_s <= 0;
22 end
23 initial
24 begin
25     # 3500 s_s <= 1;
26     # 3700 l <= 1;
27     # 200 l <= 0;
28 end
29 endmodule

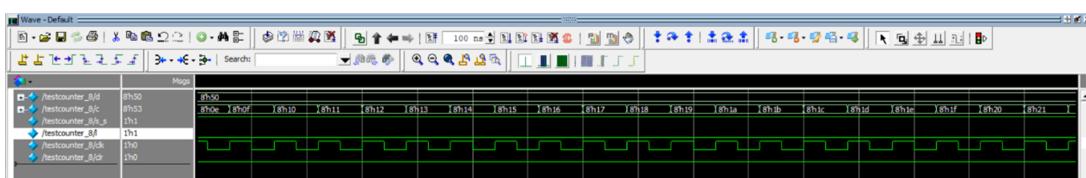
```

5.5 Επαλήθευση της λογικής λειτουργίας του μετρητή των 8 bits

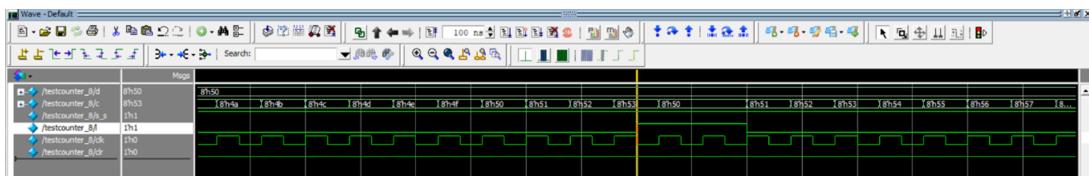
Πραγματοποιώντας εξομοίωση για τον 8-bit counter, παρατηρείται ότι το κύκλωμα λειτουργεί σωστά καθώς η μέτρηση και η φόρτωση πραγματοποιούνται όπως αναμενόταν. Κάποια ενδεικτικά screenshots των κυματομορφών, που δείχνουν τα αποτελέσματα της εξομοίωσης, φαίνονται στις εικόνες 5.3 έως 5.6.



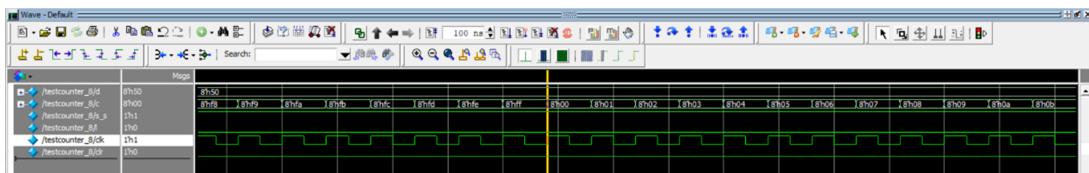
Σχήμα 5.3: Μέτρηση από 00_h έως $0E_h$



Σχήμα 5.4: Μέτρηση από $0E_h$ έως 21_h



Σχήμα 5.5: Στιγμή της παράλληλης φόρτωσης

Σχήμα 5.6: Στιγμή που ο μετρητής έφτασε την τιμή FF_h και επανέρχεται στο 00_h

5.6 Μετρητής 8 bits με σύγχρονη είσοδο καθαρισμού

Για το μετρητή με σύγχρονη είσοδο καθαρισμού, έγιναν αλλαγές στα αρχεία counter_4.v και counter_8.v οπότε προέκυψαν τα countersyn_4.v και countersyn_8.v, που φαίνονται στον κώδικα 5.4.

Κώδικας 5.4

```

1 module countersyn4 (clear, clock, load, start_stop, count, data);
2   input [3:0] data;
3   output [3:0] count;
4   input start_stop;
5   input load;
6   input clock;
7   input clear;
8   reg [3:0] count;
9
10  always @ (posedge clock)
11    if (clear) count <= 0;
12    else if (load) count <= data;
13    else if (start_stop) count <= count + 1;
14  endmodule
15
16 module countersyn8 (clear, clock, load, start_stop0, count, data)
17   ↪ ;
18   input [7:0] data;
19   output [7:0] count;
20   input start_stop0;
21   input load;
22   input clock;
23   input clear;
24   wire start_stop1;
```

```

24
25 assign start_stop1 = (count[0] && count[1] && count[2] && count
26   ↪ [3]);
27 //Dhmiourgia instances twn metrhtwn 4-bit me
28 //sygxronh eisodo katharismou
29 countersyn4 c0(clear, clock, load, start_stop0, count[3:0],
30   ↪ data[3:0]);
31 countersyn4 c1(clear, clock, load, start_stop1, count[7:4],
32   ↪ data[7:4]);
33 endmodule

```

Στο countersyn_4.v αυτό που άλλαξε είναι ότι αφαιρέθηκε η θετική ακμή του σήματος clear από τη συνθήκη ενεργοποίησης της συνθήκης always. Έτσι, αλλαγές στην τιμή του σήματος count θα πυροδοτούνται μόνο στη θετική ακμή του ρολογιού. Το σύνολο διανυσμάτων για την εξομοίωση θα είναι στο αρχείο testcountersyn_8.v, το οποίο φαίνεται στον κώδικα 5.5.

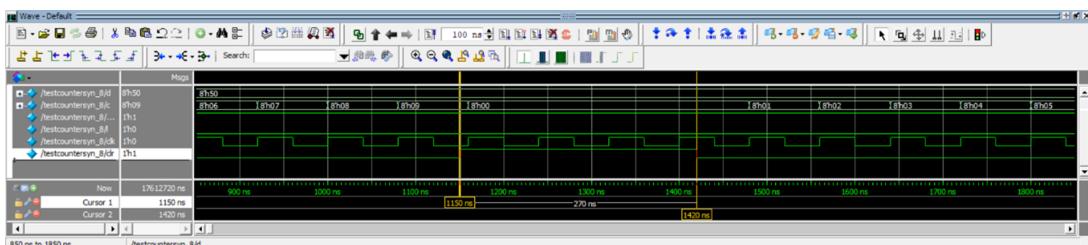
Κώδικας 5.5

```

1 module testcountersyn_8 ();
2 reg [7:0] d;
3 wire [7:0] c;
4 reg s_s, l, clk, clr;
5
6 countersyn8 inst0 (clr, clk, l, s_s, c, d);
7
8 initial
9 begin
10   # 5 clk <= 0; clr <= 0; l <= 0; s_s <= 0; d <= 8'h50;
11 end
12 always # 40 clk <= !clk;
13 initial
14 begin
15   # 100 clr <= 1;
16   # 50 clr <= 0;
17   # 1000 clr <= 1;
18   # 270 clr <= 0;
19 end
20 initial
21 begin
22   # 400 s_s <= 1;
23   # 3000 s_s <= 0;
24 end
25 initial
26 begin
27   # 3500 s_s <= 1;
28   # 3700 l <= 1;
29   # 200 l <= 0;
30 end
31 endmodule

```

Η βασική αλλαγή που έγινε είναι ότι στο initial block που ελέγχει το σήμα clr έχουν προστεθεί οι εντολές `# 1000 clr <= 1; # 270 clr <= 0;` μετά από τις δύο αρχικές που μηδενίζουν τον μετρητή. Έτσι, τη χρονική στιγμή 1150 ns το σήμα clr ενεργοποιείται και μετά από 270 ns (δηλαδή τη στιγμή 1420 ns) απενεργοποιείται. Όμως, παρατηρείται ότι το σήμα εξόδου ανταποκρίνεται στον καθαρισμό τη στιγμή της επόμενης θετικής ακμής του σήματος ρολογιού, οπότε και εξετάζεται το σήμα clr. Αντίστοιχα, ο μετρητής ξεκινά να μετράει πάλι όταν υπάρχει θετική ακμή ρολογιού στην οποία το σήμα clr είναι απενεργοποιημένο. Έτσι, η λειτουργία καθαρισμού του μετρητή γίνεται σύγχρονα με το σήμα clk. Οι παραπάνω παρατηρήσεις φαίνονται στην εικόνα 5.7.



Σχήμα 5.7: Παρατηρήσεις

5.7 Μέγιστη συχνότητα λειτουργίας του μετρητή των 8 bits

Το νέο αρχείο `counterdel_4.v` έχει τον κώδικα 5.6.

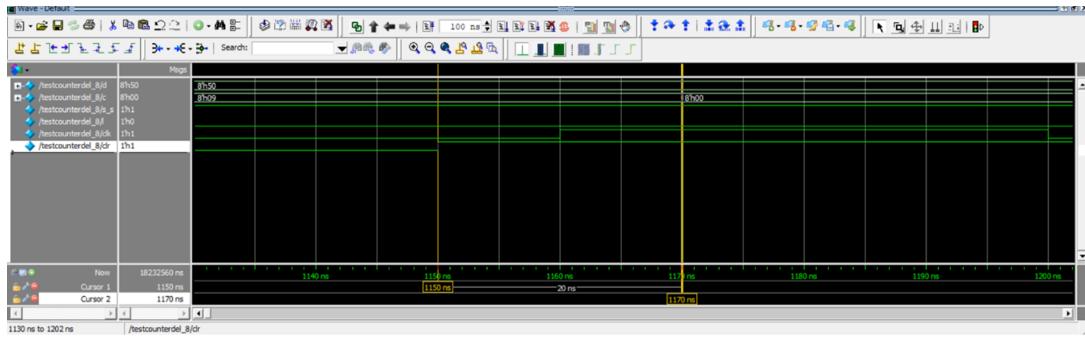
Κώδικας 5.6

```

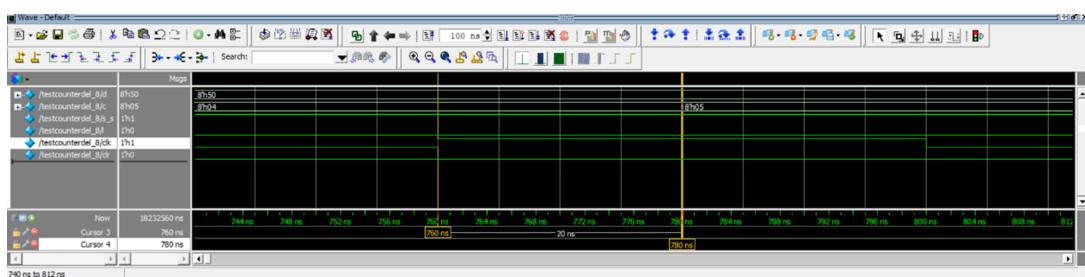
1 module counterdel4 (clear, clock, load, start_stop, count, data);
2   input [3:0] data;
3   output [3:0] count;
4   input start_stop;
5   input load;
6   input clock;
7   input clear;
8   reg [3:0] count;
9
10  always @ (posedge clock or posedge clear)
11    #20
12    if (clear) count <= 0;
13    else if (load) count <= data;
14    else if (start_stop) count <= count + 1;
15  endmodule

```

Η βασική αλλαγή είναι ότι προστέθηκε η εντολή `#20` στην αρχή του `always` block στον μετρητή των 4 bits, ώστε να καθυστερήσουν όλες οι λειτουργίες του κατά 20 χρονικές στιγμές. Κάνοντας εξομοίωση του 8-bit counter, παρατηρείται ότι υπάρχει καθυστέρηση 20 ns σε όλες τις λειτουργίες του μετρητή, όπως στην ανανέωση της τιμής εξόδου του και στον καθαρισμό. Αυτό εντοπίζεται στις εικόνες 5.8 και 5.9.



Σχήμα 5.8: Καθυστέρηση ασύγχρονου clear κατά 20 ns



Σχήμα 5.9: Καθυστέρηση της λειτουργίας μέτρησης κατά 20 ns από την ανοδική ακμή του ρολογιού

Για να βρεθεί η μέγιστη συχνότητα λειτουργίας του 8-bit counter αρκεί να υπολογιστεί η ελάχιστη περίοδος του ρολογιού. Παρατηρείται ότι στα προηγούμενα ερωτήματα, όπου δεν υπήρχε καθυστέρηση εξαιτίας του testbench, το εργαλείο δεν εισήγαγε από μόνο κάποιο delay και έτσι με το που παρατηρούταν θετική ακμή ρολογιού ή ενεργοποίηση του clear (στην ασύγχρονη περίπτωση καθαρισμού) οι σχετικές αλλαγές στις τιμές εξόδου εμφανίζονταν αμέσως. Άρα, στη συγκεκριμένη περίπτωση το ρολόι θα πρέπει να έχει περίοδο τουλάχιστον 20 ns, ώστε να υπάρχει χρόνος για να εκτελεστούν οι εντολές του κυκλώματος πέραν της εξ ορισμού καθυστέρησης. Έτσι, είναι:

$$T \geq 20 \text{ nsec} \Rightarrow \frac{1}{T} \leq \frac{1}{20 \cdot 10^{-9}} \text{ Hz} \Rightarrow f \leq 0.05 \cdot 10^9 \text{ Hz} = 0.05 \text{ GHz}$$

Άρα, η μέγιστη συχνότητα λειτουργίας του μετρητή των 8 bits είναι $f_{max} = 0.05 \text{ GHz}$.

Έκτη εργαστηριακή άσκηση

6.1 Εισαγωγή

Τα αρχεία που αναφέρονται στην άσκηση είναι τα εξής:

- dip2leds.v, dip2leds.ucf και dip2leds.bit για το πρώτο ζητούμενο
- unsigned_mult.v, unsigned_mult.ucf και unsigned_mult.bit για το πρώτο σκέλος του δεύτερου ζητούμενου
- signed_mult.v, signed_mult.ucf και signed_mult.bit για το δεύτερο σκέλος του δεύτερου ζητούμενου

6.2 Κύκλωμα απεικόνισης δυαδικού αριθμού από τα dip switches στα led bars

Στο πρώτο ζητούμενο, ο αριθμός θα δίνεται μέσω των DIP Switches σε δυαδική μη προσημασμένη μορφή. Γίνεται η παραδοχή ότι οι τιμές των switches που φτάνουν στο FPGA θα είναι είτε στο λογικό 0 είτε στο 1 και δε θα έχουν τιμή x (άγνωστη) ή z (υψηλής εμπέδησης). Η απεικόνιση θα γίνεται στο 7 - segment LED bar του XSA, όμως η εκφώνηση δε διευκρινίζει το σύστημα αναπαράστασης. Γι' αυτό επιλέχθηκε να είναι σε δεκαεξαδικό σύστημα αρίθμησης, ώστε να μπορούν να αναπαρασταθούν και οι τετραψήφιοι αριθμοί 1010_b έως και 1111_b , οι οποίοι στο δεκαδικό σύστημα θα απαιτούσαν δύο ψηφία αντί για ένα. Η περιγραφή του κυκλώματος για το module dip2leds φαίνεται στον κώδικα 6.1.

Κώδικας 6.1

```
1 module dip2leds(a, b, c, d, s, dp);
2     input a, b, c, d;
3     output reg [6:0] s;
4     output dp;
5     reg [3:0] o;
6
7     always @ (a or b or c or d) begin
8         o = ~{a, b, c, d};
9         case (o)
10             4'b0000: s = 7'b1110111; //0
11             4'b0001: s = 7'b0010010; //1
12             4'b0010: s = 7'b1011101; //2
13             4'b0011: s = 7'b1011011; //3
14             4'b0100: s = 7'b0111010; //4
15             4'b0101: s = 7'b1101011; //5
16             4'b0110: s = 7'b1101111; //6
```

```

17      4'b0111: s = 7'b1110010; //7
18      4'b1000: s = 7'b1111111; //8
19      4'b1001: s = 7'b1111011; //9
20      4'b1010: s = 7'b1111110; //A
21      4'b1011: s = 7'b0101111; //b
22      4'b1100: s = 7'b1100101; //C
23      4'b1101: s = 7'b0011111; //d
24      4'b1110: s = 7'b1101101; //E
25      4'b1111: s = 7'b1101100; //F
26      endcase
27  end
28  assign dp = 0;
29 endmodule

```

Εδώ το κύκλωμα όταν ανιχνεύει αλλαγή στις στάθμες των εισόδων a, b, c ή d θα ελέγχει την τιμή που αυτές σχηματίζουν και θα ανάβει τα αντίστοιχα LEDs του 7 - segment display. Σημαντική παρατήρηση ότι η πολικότητα των τιμών 0 και 1 στα switches είναι ανεστραμμένη απ' ό,τι αναμένεται και γι' αυτό πριν γίνει ο έλεγχος στις περιπτώσεις της δομής case, αντιστρέφονται τα bits εισόδου. Το LED dp θα είναι μόνιμα OFF. Το αρχείο location constraint ονομάζεται dip2leds.ucf και παραμένει ίδιο με το mlx.ucf. Για να βρεθεί η μέγιστη συχνότητα λειτουργίας του κυκλώματος, αρκεί να γίνει Timing Analysis και να εντοπιστεί η μέγιστη καθυστέρηση στο κύκλωμα, pad to pad, οπότε αυτή θα είναι η ελάχιστη περίοδος λειτουργίας του. Χρησιμοποιώντας το εργαλείο Project Navigator και τροφοδοτώντας το με τα αρχεία dip2leds.v και dip2leds.ucf, πραγματοποιήθηκαν οι διαδικασίες Synthesis, Translation, Map και Place & Route μέσω της επιλογής Implement Top Module. Έπειτα, στο εργαλείο Timing Analyzer παρατηρείται ότι η μέγιστη συχνότητα λειτουργίας του κυκλώματος είναι $f_{max} = \frac{1}{13.094 \cdot 10^{-9}} Hz = 0.076 GHz$. Τέλος, το ζητούμενο αρχείο προγραμματισμού ονομάζεται dip2leds.bit και βρίσκεται στο φάκελο με τα υπόλοιπα αρχεία της άσκησης.

6.3 Κύκλωμα Multiplier 2 αριθμών

6.3.1 Οι αριθμοί να είναι μη προσημασμένοι

Θεωρείται ότι οι δύο αριθμοί των 2 bits θα δίνονται στο FPGA μέσω των DIP Switches, ενώ το αποτέλεσμα θα αναπαρίσταται στο 7 - segment LED display. Έστω a1a0 και b1b0 οι δύο αριθμοί με πιο σημαντικά ψηφία τα a1 και b1. Τότε, το a1 θα συνδεθεί στο pin k4, το a0 στο k3, το b1 στο k2 και το b0 στο j4. Ο κώδικας περιγραφής της συγκεκριμένης περίπτωσης περιλαμβάνει το module unsigned_mult, στο οποίο γίνεται μια κλασική πράξη πολλαπλασιασμού. Ανάλογα με το αποτέλεσμα, θα αναπαρίσταται ο αριθμός στα LEDs σε δεκαδικό σύστημα, καθώς ο μεγαλύτερος σε αναπαράσταση αριθμός θα είναι το 9, άρα ένα ψηφίο αρκεί. Επίσης, το LED dp θα είναι μόνιμα σβηστό (OFF). Τέλος, σε περίπτωση που υπάρχει σφάλμα και το γινόμενο prod αποκτήσει τιμή εκτός των επιτρεπτών αριθμών, στα LEDs θα εμφανιστεί το γράμμα E, από τη λέξη Error. Το περιεχόμενο του αρχείου unsigned_mult.v φαίνεται στον κώδικα 6.2.

Κώδικας 6.2

```

1 module unsigned_mult(a, b, s, dp);
2     input [1:0]a;
3     input [1:0]b;
4     output reg [6:0]s;
5     output dp;
6     reg [3:0]prod;
7     reg [1:0]corr_a;
8     reg [1:0]corr_b;
9
10    always @ (a, b) begin
11        corr_a = ~a;
12        corr_b = ~b;
13        prod = corr_a * corr_b;
14        case (prod)
15            4'b0000: s[6:0] = 7'b1110111; //0
16            4'b0001: s[6:0] = 7'b0010010; //1
17            4'b0010: s[6:0] = 7'b1011101; //2
18            4'b0011: s[6:0] = 7'b1011011; //3
19            4'b0100: s[6:0] = 7'b0111010; //4
20            4'b0110: s[6:0] = 7'b1101111; //6
21            4'b1000: s[6:0] = 7'b1111111; //8
22            4'b1001: s[6:0] = 7'b1111011; //9
23        default s[6:0] = 7'b1101101; //Error
24        endcase
25    end
26    assign dp = 0;
27 endmodule

```

Το αρχείο με τους περιορισμούς τύπου location είναι το `unsigned_mult.ucf`, του οποίου τα περιεχόμενα φαίνονται στον κώδικα 6.3.

Κώδικας 6.3

```

1 net a<1> loc=k4;
2 net a<0> loc=k3;
3 net b<1> loc=k2;
4 net b<0> loc=j4;
5 net s<6> loc=r10;
6 net s<5> loc=t7;
7 net s<4> loc=p10;
8 net s<3> loc=r7;
9 net s<2> loc=n6;
10 net s<1> loc=m11;
11 net s<0> loc=m6;
12 net dp loc=n11;

```

Για να βρεθεί η μέγιστη συχνότητα λειτουργίας του κυκλώματος θα ακολουθηθεί ίδια με πρώτο ζητούμενο διαδικασία. Παρατηρείται ότι η μέγιστη καθυστέρηση, pad to pad, είναι 13.175 ns και εντοπίζεται στη διαδρομή από το pad $a < 1 >$ στο $s < 6 >$. Άρα, η μέγιστη συχνότητα θα είναι:

$$f_{max} = \frac{1}{13.175 \cdot 10^{-9}} Hz = 0.076 GHz$$

Τέλος, το ζητούμενο αρχείο προγραμματισμού ονομάζεται `unsigned_mult.bit` και βρίσκεται στο φάκελο με τα υπόλοιπα αρχεία της άσκησης.

6.3.2 Οι αριθμοί να είναι σε παράσταση συμπληρώματος του 2

Εδώ δημιουργείται το module `signed_mult` στο αρχείο `signed_mult.v`. Πλέον, το εύρος αναπαράστασης θα είναι από το -8 έως το +7. Τα μέτρα των αριθμών θα αναπαρίστανται όπως και πριν στα LEDs του 7 - segment display, ενώ το πρόσημο θα δηλώνεται από το LED dp. Συγκεκριμένα, αν ο αριθμός είναι αρνητικός, τότε το dp θα είναι ON, ενώ αν είναι θετικός, τότε το dp θα είναι OFF. Ο κώδικας που αντιστοιχεί στο ζητούμενο design είναι ο κώδικας 6.3.

Κώδικας 6.4

```

1  module signed_mult(a, b, s, dp);
2      input [1:0]a;
3      input [1:0]b;
4      output reg signed [6:0]s;
5      output reg dp;
6      reg signed [3:0]prod;
7      reg signed [1:0]corr_a;
8      reg signed [1:0]corr_b;
9
10     always @ (a, b) begin
11         corr_a = ~a;
12         corr_b = ~b;
13         prod = corr_a * corr_b;
14         case (prod)
15             4'b0000: {s, dp} = 8'b11101110; //0
16             4'b0001: {s, dp} = 8'b00100100; //1
17             4'b0010: {s, dp} = 8'b10111010; //2
18             4'b0100: {s, dp} = 8'b01110100; //4
19             4'b1110: {s, dp} = 8'b10111011; //-2
20             4'b1111: {s, dp} = 8'b00100101; //-1
21         default s[6:0] = 7'b1101101; //Error
22         endcase
23     end
24 endmodule

```

Το αρχείο με τα location constraints είναι το `signed_mult.ucf` και είναι το ίδιο με αυτό του πρώτου σκέλους του δεύτερου ζητήματος. Πραγματοποιώντας *Implement Top Module*

και κάνοντας Timing Analysis, παρατηρείται ότι η μέγιστη καθυστέρηση είναι από το pad $b < 0 >$ στο $s < 6 >$ και ισούται με 12.553 ns. Άρα, η μέγιστη συχνότητα θα είναι:

$$f_{max} = \frac{1}{12.553 \cdot 10^{-9}} \text{ Hz} = 0.08 \text{ GHz}$$

Τέλος, το ζητούμενο αρχείο προγραμματισμού ονομάζεται `signed_mult.bit` και βρίσκεται στο φάκελο με τα υπόλοιπα αρχεία της άσκησης.

Έβδομη εργαστηριακή άσκηση

7.1 Εισαγωγή

Τα αρχεία που αναφέρονται στην άσκηση είναι τα εξής:

- bin_2_leds.v με τα μοδυλες bin_2_7 και bin_2_bar
- cnt.v με τα modules cnt25, cnt20, cnt8b, cnt5b, TenHertz και OneHertz
- secondcounter.v με τα modules singleseconds, tenths of seconds και secondcounter
- tops.v με το main module tops
- Το αρχείο με τα location constraints tops.ucf
- tops.bit που περιέχει το bitstream για τον προγραμματισμό του FPGA

7.2 Λειτουργία stop-watch

Για τη λειτουργία stop - watch αρκεί να προσθέσουμε μια επιπλέον είσοδο στο κύκλωμα, η οποία όταν ενεργοποιείται θα πρέπει να «παγώνει» το 7-segment LED αλλά η μέτρηση να συνεχίζεται κανονικά. Επιλέχθηκε ο διακόπτης αυτός να είναι το dip switch1-2 από την πλακέτα XSA, δηλαδή δίπλα στο reset, που έχει το pin k3. Ο διακόπτης stop - watch θα είναι ενεργοποιημένος όταν είναι με φορά προς τα πάνω (λογικό 0) και αντίστοιχα απενεργοποιημένος όταν είναι με φορά προς τα κάτω (λογικό 1). Επειδή η μέτρηση θα συνεχίζεται κανονικά, η είσοδος του switch1-2 θα επηρεάζει μόνο το module απεικόνισης των αριθμών στα 7-segment displays του XST και όχι τα modules που έχουν αναλάβει τη μέτρηση. Έτσι, το module bin_2_7 μετά την προσθήκη της λειτουργίας stop - watch περιγράφεται από τον κώδικα 7.1.

Κώδικας 7.1

```
1 module bin_2_7 (x, pause, s);
2   input           pause;
3   input [3:0] x;
4   output [6:0] s;
5
6   assign s =           (pause == 0 ) ? s : // 
7     ↪ pause
8     (x == 4'd0 ) ? 7'b1111110 : //0
9     (x == 4'd1 ) ? 7'b0110000 : //1
10    (x == 4'd2 ) ? 7'b1101101 : //2
11    (x == 4'd3 ) ? 7'b1111001 : //3
12    (x == 4'd4 ) ? 7'b0110011 : //4
13    (x == 4'd5 ) ? 7'b1011011 : //5
```

```

13      (x == 4'd6) ? 7'b1011111 : //6
14      (x == 4'd7) ? 7'b1110010 : //7
15      (x == 4'd8) ? 7'b1111111 : //8
16      (x == 4'd9) ? 7'b1111011 : //9
17      (x == 4'd10) ? 7'b1110111 : //A
18      (x == 4'd11) ? 7'b0011111 : //b
19      (x == 4'd12) ? 7'b1001110 : //C
20      (x == 4'd13) ? 7'b0111101 : //d
21      (x == 4'd14) ? 7'b1000111 : 7'b1000111; //E, F
22 endmodule

```

7.3 Μέτρηση δεκάτων

Για τη μέτρηση δεκάτων του δευτερολέπτου όμως χρησιμοποιηθεί το led bar του XST το οποίο όμως μετρά από το 0 έως το 9 σε στυλ θερμομέτρου με φορά από κάτω προς τα πάνω. Για τη χρήση του led bar όμως χρησιμοποιηθούν τα pins εξόδου L5, N2, M3, N1, T13, L15, J13, H15, J16 και J14. Παρ' ότι αρκεί να χρησιμοποιηθούν τα 9 από τα 10 bars, στην άσκηση χρησιμοποιούνται όλα ώστε να έχουν σαφή τιμή και να μην εκπέμπουν καθόλου φως όταν δε ζητείται. Αρχικά πρέπει να σχεδιαστεί κύκλωμα μέτρησης από το 0 έως το 9 που όμως λειτουργεί με βάση το ρολόι του συστήματος, αλλά όμως δέχεται είσοδο επίτρεψης μέτρησης σύμφωνα με τη συχνότητα των 10 Hz. Για να παραχθούν τα 10 Hz από τα 100 MHz, όμως πρέπει να διαιρέσουμε τη συχνότητα του ρολογιού του συστήματος με 10^7 , μιας και $\frac{10^8}{10^7} Hz = 10 Hz$. Η κατασκευή ενός μετρητή με $\lceil \log_2 10^7 \rceil = 24$ δυαδικά ψηφία είναι αρκετά κοστοβόρα και γι' αυτό όμως κατασκευαστούν μικρότεροι μετρητές συνδεδεμένοι σε σειρά όπως στο πρώτο ζητούμενο. Είναι $10^7 = 25^3 \cdot 2^5 \cdot 20$, άρα αρκεί να χρησιμοποιηθούν 3 mod-5 μετρητές, ένας των 5 bits και τέλος ένας mod-20. Ο mod-25 μετρητής είναι ήδη έτοιμος, οπότε πρέπει να κατασκευαστούν σε κώδικα τα modules *cnt5b* και *cnt20* καθώς και το module *TenHertz* που όμως συνθέτει τον παλμό των 10 Hz. Ο σχετικός κώδικας περιέχεται στο αρχείο *cnt.v* και είναι ο κώδικας 7.2.

Κώδικας 7.2

```

1 module cnt5b (reset, clk, enable, clkdiv32);
2   input      reset, clk, enable;
3   output     clkdiv32;
4   reg [4:0]  cnt;
5
6   assign clkdiv32 = (cnt==8'd31);
7   always @ (posedge reset or posedge clk)
8     if (reset) cnt <= 0;
9     else if (enable) cnt <= cnt + 1;
10  endmodule
11 //
12   //-----//
13
12 module cnt20 (reset, clk, enable, clkdiv20);
13   input      reset, clk, enable;
14   output     clkdiv20;

```

```

15 reg [4:0] cnt;
16
17 assign clkdiv20 = (cnt==5'd19);
18 always @(posedge reset or posedge clk)
19   if (reset) cnt <= 0;
20   else if (enable)
21     if (clkdiv20) cnt <= 0;
22     else cnt <= cnt + 1;
23 endmodule
24 //
25
26
27 module TenHertz(reset, clk, en_nxt);
28   input      clk, reset;
29   output     en_nxt;
30   wire       clk10Hz;
31
32   cnt25 i0 (reset, clk, 1'b1, first);
33   cnt25 i1 (reset, clk, first, second);
34   cnt25 i2 (reset, clk, first & second, third);
35   cnt5b i3 (reset, clk, first & second & third, fourth);
36   cnt20 i4 (reset, clk, first & second & third & fourth, clk10Hz);
37   assign en_nxt = first & second & third & fourth & clk10Hz;
38 endmodule

```

To module απεικόνισης των αριθμών 0 έως 9 στο led bar της πλακέτας XST είναι το bin_2_bar, το οποίο υποστηρίζει τη δυνατότητα stop - watch μέσω του σήματος εισόδου pause. Το λογισμικό σε Verilog είναι ο κώδικας 7.3.

Κώδικας 7.3

```

1 module cnt5b (reset, clk, enable, clkdiv32);
2   input      reset, clk, enable;
3   output     clkdiv32;
4   reg [4:0] cnt;
5
6   assign clkdiv32 = (cnt==8'd31);
7   always @(posedge reset or posedge clk)
8     if (reset) cnt <= 0;
9     else if (enable) cnt <= cnt + 1;
10  endmodule
11 //
12
13 module cnt20 (reset, clk, enable, clkdiv20);
14   input      reset, clk, enable;
15   output     clkdiv20;
16   reg [4:0] cnt;
17   assign clkdiv20 = (cnt==5'd19);

```

```

18  always @ (posedge reset or posedge clk)
19      if (reset) cnt <= 0;
20      else if (enable)
21          if (clkdiv20) cnt <= 0;
22          else cnt <= cnt + 1;
23  endmodule
24 //
25  module TenHertz (reset, clk, en_nxt);
26  input     clk, reset;
27  output    en_nxt;
28  wire      clk10Hz;
29
30  cnt25 i0 (reset, clk, 1'b1, first);
31  cnt25 i1 (reset, clk, first, second);
32  cnt25 i2 (reset, clk, first & second, third);
33  cnt5b i3 (reset, clk, first & second & third, fourth);
34  cnt20 i4 (reset, clk, first & second & third & fourth, clk10Hz);
35  assign en_nxt = first & second & third & fourth & clk10Hz;
36  endmodule

```

Στην αρχή θα μετράνε σε σειρά οι τρεις μετρητές mod-25, μετά αυτός των 5 bits και τέλος ο mod-20. Στο τέλος, το σήμα εξόδου από το module *TenHertz* θα συνδέεται με το σήμα επίτρεψης μέτρησης ενός στιγμιότυπου του module *singleseconds*. Αυτό με τη σειρά του θα τροφοδοτεί με αριθμούς από το 0 έως το 9 το module *bin_2_bar* οπότε έτσι θα γίνεται η απεικόνιση των δεκάτων του δευτερολέπτου. Η τελική μορφή του module *tops* φαίνεται στον κώδικα 7.4.

Κώδικας 7.4

```

1  module tops (reset, clk, pause, left, right, bar);
2      input             reset, clk, pause;
3      output [6:0] left, right;
4      output [9:0] bar;
5      wire      [2:0] ts;
6      wire      [3:0] ss, bs;
7
8      TenHertz           i0 (reset, clk, en_bar);
9      singleseconds      i1 (reset, clk, en_bar, bs, nxt);
10     bin_2_bar          bt (bs, pause, bar);
11     OneHertz            i2 (reset, clk, en_nxt);
12     secondcounter i3 (reset, clk, en_nxt, ts, ss);
13     bin_2_7             lt ({1'b0,ts}, pause, left);
14     bin_2_7             rt (ss, pause, right);
15  endmodule

```

Τέλος, στο αρχείο *tops.ucf* με τα location constraints προστίθεται οι περιορισμοί net pause loc=k3; και NET "pause" CLOCK_DEDICATED_ROUTE = FALSE;.Η δεύτερη γραμμή προστέθηκε στο UCF αρχείο μετά από υπόδειξη του εργαλείου WebPack ISE

ώστε να αρθεί η βελτιστοποίηση που γίνεται στο routing του clock και έτσι να μπορεί να δρομολογηθεί το σήμα pause στο pin k3. Παρόλο που έγιναν αρκετές δοκιμές τόσο στα switches και buttons του XSA όσο και σε αυτά του XST, δε βρέθηκε κάποιο pin που να μην αποτύγχανε στο βήμα place & route λόγω των conflicts που δημιουργούνται. Έτσι, συνειδητά επιλέχθηκε το pin k3 με τη σκέψη ότι η άρση βελτιστοποίησης στο routing του clock δε θα δημιουργήσει πρόβλημα στο (μικρής έκτασης) κύκλωμα της άσκησης.

'Ογδοη εργαστηριακή άσκηση

8.1 Εισαγωγή

Τα αρχεία που αναφέρονται στην άσκηση είναι τα εξής:

- `kbd_protocol.v`, που περιέχει τα modules `kbd_protocol` και `check_type`
- `scan_2_7.v`, που περιέχει τα modules `scan_2_7seg`, `res_2_7segXSA` και `display_previous`
- `calculator.v`, που περιέχει τα modules `code_2_num`, `calculator` και `divider`
- `eight_top.v`, που περιέχει το top module `eight`
- `eight.ucf`, που περιέχει τα location constraints του σχεδιασμού
- `eight_top.bit`, που περιέχει το bitstream για τον προγραμματισμό του FPGA

8.2 Εφαρμογή ροής σύνθεσης και υλοποίησης του κυκλώματος

Κρατώντας συνεχώς ένα αριθμητικό πλήκτρο πατημένο, παρατηρείται ότι δεν απεικονίζεται στο 7-segment LED του XST. Αυτό συμβαίνει καθώς μέσω του κώδικα υλοποίησης του keyboard protocol, έχει γίνει η παραδοχή ότι ένα scancode ανιχνεύεται μόνο αμέσως μετά το `< F0 >`, δηλαδή όταν το πλήκτρο αφήνεται (on release). Έτσι, όσο διάστημα και να έχουμε πατημένο το αριθμητικό πλήκτρο, η ανίχνευση του αριθμού θα γίνει όταν το πλήκτρο αφεθεί και θα θεωρηθεί ότι πατήθηκε για μία μόνο στιγμή.

8.3 Λειτουργία προηγούμενου πλήκτρου

Η κεντρική ιδέα φαίνεται στο module `display_previous`. Εφόσον έχει πατηθεί ένα πλήκτρο, ελέγχεται ο τύπος του, δηλαδή αν είναι αριθμητικό πλήκτρο, έγκυρος τελεστής (αφορά και το δεύτερο σκέλος του ζητούμενου) ή κάτι άλλο οπότε και θα αγνοείται. Εφόσον ο τύπος του πλήκτρου είναι αριθμός, ελέγχεται αν είναι η πρώτη φορά που πατιέται αριθμητικό πλήκτρο με τη βοήθεια της μεταβλητής `first_number`. Αν είναι η πρώτη φορά, τότε εμφανίζεται τιμή μόνο αριστερό 7-segment LED αλλιώς στο αριστερό θα εμφανίζεται ο ποιο πρόσφατος αριθμός και στο δεξί ο προηγούμενος. Σε αυτή τη λογική βοήθησε η χρήση blocking assignments, ώστε πρώτα να γίνει η ανάθεση του αριστερού τελεστή στον δεξί και μετά να ανατεθεί νέα τιμή στον αριστερό. Ο έλεγχος του τύπου του scancode γίνεται στο module `check_type`. Αν το πλήκτρο είναι αριθμητικό, τότε θα έχει `type = 0`, αν είναι έγκυρος τελεστής θα έχει `type = 1` και αν είναι οπιδήποτε άλλο θα έχει `type = 2`. Τα πλήκτρα με `type = 2` θα αγνοούνται σε κάθε περίπτωση. Επίσης, το flag που σηματοδοτεί το πάτημα ενός πλήκτρου θα είναι η

μεταβλητή `button_flag` η οποία ελέγχεται από το module `kbd_protocol` και ενεργοποιείται μόνο όταν έχει πατηθεί και αφεθεί ένα οποιοδήποτε πλήκτρο. Ο κώδικας 8.1 περιγράφει τα modules `kbd_protocol`, `display_previous` και `check_type`. Οι προσθήκες στο module `kbd_protocol` είναι οι γραμμές 4,5, 29, 36 και 49.

Κώδικας 8.1

```

1 //-----
2 module kbd_protocol (reset, clk, ps2clk, ps2data, button_flag,
3   ↪ scancode);
4   input           reset, clk, ps2clk, ps2data;
5   //Wire button_flag is 1 when a key has been pressed and
6   ↪ released
7   output reg      button_flag;
8   output reg [7:0] scancode;
9
10 // Synchronize ps2clk to local clock and check for falling edge;
11 reg      [7:0] ps2clksamples; // Stores last 8 ps2clk
12   ↪ samples
13 always @(posedge clk or posedge reset)
14   if (reset) ps2clksamples <= 8'd0;
15   else ps2clksamples <= {ps2clksamples[7:0], ps2clk};
16
17 reg      [9:0] shift;    // Stores a serial package, excluding the
18   ↪ stop bit;
19 reg      [3:0] cnt;     // Used to count the ps2data samples
20   ↪ stored so far
21 reg      f0;          // Used to indicate that f0 was
22   ↪ encountered earlier
23
24 // A simple FSM is implemented here. Grab a whole package,
25 // check its parity validity and output it in the scancode
26 // only if the previous read value of the package was F0
27 // that is, we only trace when a button is released, NOT when
28   ↪ it is
29 // pressed.
30
31 always @(posedge clk or posedge reset)
32 begin
33   button_flag <= 0;
34   if (reset)
35     begin
36       cnt      <= 4'd0;
37       scancode <= 8'd0;
38       shift    <= 10'd0;
39       f0       <= 1'b0;
40     end
41   end
42 endmodule

```

```

36     button_flag <= 1'b0;
37     end
38   else if (fall_edge)
39   begin
40     if (cnt == 4'd10) // we just received what should be
41       ↪ the stop bit
42     begin
43       cnt <= 0;
44       if ((shift[0] == 0) && (ps2data == 1) && (^shift[9:1]
45         ↪ ==1)) // A well received serial packet
46       begin
47         if (f0) // following a scancode of f0. So a
48           ↪ key is released !
49       begin
50         scancode    <= shift[8:1];
51         f0          <= 0;
52       button_flag <= 1;
53     end
54   else if (shift[8:1] == 8'hF0) f0 <= 1'b1;
55   end // All other packets have to do with key
56   ↪ presses and are ignored
57   end
58   else
59   begin
60     shift <= {ps2data, shift[9:1]}; // Shift right
61       ↪ since LSB first is transmitted
62     cnt    <= cnt+1;
63   end
64 end
65 endmodule
66
67 // -----
68
69 module display_previous (reset, clk, button_flag, type, scancode,
70   ↪ two_ops, left, right, operator);
71 input           reset, clk, button_flag;
72 input [1:0]      type;
73 input [7:0]      scancode;
74 output reg      two_ops;
75 output reg [7:0] left, right, operator;
76 reg [7:0]        previous;
77 reg              first_time;
78
79 //H metavlhth first_time einai energopoimenh mono sthn arxh pou
80   ↪ den exoun plhktrologithei akomh dyo arithmoi
81 initial
82 begin
83   first_time = 1'b1;
84   two_ops    = 1'b0;

```

```

79   end
80
81   always @( posedge clk or posedge reset )
82   begin
83     if ( reset )
84       begin
85       left      = 8'd0;
86       right     = 8'd0;
87       previous  = 8'd0;
88       first_time = 1'b1;
89       two_ops    = 1'b0;
90     end
91   else if ( button_flag ) //exei patithei ena pliktro
92     begin
93       //Elegxei to type wste na apeikonisei mono arithmous kai
94       //→ egkyrous telestes,
95       //enw ola ta alla plhktra tha agnoountai
96       if ( type == 2'd1 )
97         operator = scancode;
98       else if ( type == 2'd0 )
99         if ( first_time ) //prwto arithmitiko plhktr
100        begin
101          left      = scancode;
102          previous  = scancode;
103          first_time = 1'b0;
104        end
105      else
106        begin
107          left      = scancode;
108          right     = previous;
109          previous  = scancode;
110          two_ops    = 1'b1;
111        end
112     end
113   endmodule
114
115 //-----
116
117 module check_type(button_flag, scan, type);
118   input           button_flag;
119   input [7:0] scan;
120   output reg [1:0] type;
121
122   always @( scan )
123   begin
124     if ( button_flag )
125       if ( ( scan == 8'h45 ) || ( scan == 8'h16 ) || ( scan == 8'h1E
126       → ) || ( scan == 8'h26 ) || ( scan == 8'h25 ) ||

```

```

126      ( scan == 8'h2E ) || ( scan == 8'h36 ) || ( scan == 8'h3D )
127          ↪ || ( scan == 8'h3E ) || ( scan == 8'h46 ) )
128      type <= 2'd0;
129  else if ( ( scan == 8'h79 ) || ( scan == 8'h7B ) || ( scan ==
130          ↪ 8'h7C ) || ( scan == 8'h4A ) )
131      type <= 2'd1;
132  else type <= 2'd2;
133 end
134 endmodule
135 // -----
136

```

8.4 Λειτουργία calculator

Η κεντρική σκέψη για το σχεδιασμό της αριθμομηχανής μη προσημασμένης αριθμητικής φαίνεται στο module *calculator*, το οποίο υλοποιεί έναν σύγχρονο *calculator* με βάση το *clk* και το *reset*. Το αποτέλεσμα θα υπολογίζεται εφόσον ο τύπους του τρέχοντος *scancode* είναι $1'd1$, δηλαδή operator, και εφόσον έχουν ήδη δοθεί δύο αριθμοί (μεταβλητή *two_ops*). Η μεταβλητή *two_ops* διασφαλίζει ότι δε θα γίνει υπολογισμός στην περίπτωση που έχει δοθεί μόνο τελεστής και κανένας αριθμός, ή έχει δοθεί τελεστής και ένας μόνο αριθμός. Ο έλεγχός της γίνεται στο module *display_previous*. Το αποτέλεσμα μια πράξης (μεταβλητή *tmp*) μπορεί να φτάσει μέχρι και 7 δυαδικά ψηφία, στην περίπτωση της πράξης $9 \cdot 9 = 81_d = 1010001_b$. Στη μεταβλητή *εξόδου apotelesma* θα ανατεθούν τα 4 λιγότερο σημαντικά bits του *tmp*. Προκειμένου να εμφανιστεί το least significant digit στον πολλαπλασιασμό, όταν ανιχνεύεται ο τελεστής \div θα εκτελείται ένα for loop 8 επαναλήψεων όπου σε κάθε επανάληψη αν το γινόμενο είναι μεγαλύτερο του 10, θα αφαιρείται μία δεκάδα. Έτσι, στη χειρότερη περίπτωση που το γινόμενο είναι 81 θα αφαιρεθούν 8 δεκάδες και στο τέλος θα υπάρχει μόνο το ψηφίο των μονάδων. Ο συγκεκριμένος σχεδιασμός του πολλαπλασιασμού αρκεί για το μικρό σε έκταση κύκλωμα της άσκησης. Όμως, δεν είναι βέλτιστος και επεκτάσιμος για μεγαλύτερα κυκλώματα. Σε τέτοια περίπτωση το κύκλωμα που υπολογίζει το λιγότερο σημαντικό ψηφίο του γινομένου μπορεί να σχεδιαστεί σαν FSM ώστε να είναι synthesizable για το FPGA. Σχετικά με την πράξη \div της διαίρεσης, ο τελεστής \div δεν είναι synthesizable οπότε θα πρέπει να υλοποιηθεί χειροκίνητα με Verilog. Η διαίρεση γίνεται στο module *divider* σε συλ look-up table για να μειωθούν στο ελάχιστο οι κύκλοι ρολογιού για τον υπολογισμό του πηλίκου. Η υλοποίηση γίνεται σύγχρονα ως προς το *clk* και το *reset*, ενώ το πηλίκο υπολογίζεται εφόσον το πιο πρόσφατο *scancode* είναι ο τελεστής \div και έχουν διαβαστεί δύο αριθμοί. Πρώτα ελέγχεται αν ο διαιρέτης είναι 0, οπότε και θα εμφανιστεί error (E). Επειτα, αν ο διαιρετέος είναι μικρότερος του διαιρέτη, οπότε το πηλίκο θα είναι 0. Σε περίπτωση που διαιρετέος και διαιρέτης έχουν ίδια τιμή το αποτέλεσμα θα είναι 1. Σε κάθε άλλη περίπτωση το πηλίκο είναι η τιμή που υποδεικνύεται από το structure case. Ο κώδικας 8.2 περιγράφουν τα modules *calculator* και *divider*.

Κώδικας 8.2

```

1 module calculator( reset , clk , type , two_ops , operator , left_op ,
    ↪ right_op , phliko , apotelesma );
2     input          reset , clk , type , two_ops ;
3     input [7:0] operator;
4     input [3:0] left_op , right_op , phliko;
5     output [3:0] apotelesma;
6     reg      [6:0] tmp;
7     integer i;
8
9     initial
10    begin
11        tmp = 7'd0;
12    end
13
14    always @ (posedge clk or posedge reset)
15    begin
16        if ( reset )
17            begin
18                tmp = 7'd0;
19            end
20
21        else if ( type && two_ops )
22            begin
23                case ( operator )
24                    8'h79: tmp = left_op + right_op;
25                    8'h7B: tmp = left_op - right_op;
26                    8'h7C: tmp = left_op * right_op;
27                    8'h4A: tmp = phliko;
28                endcase
29                if ( operator == 8'h7C )
30                    for (i = 0; i < 8 ; i = i + 1)
31                        if (tmp >= 7'd10) tmp = tmp - 7'd10;
32            end
33        end
34        assign apotelesma = tmp[3:0];
35    endmodule
36
37 // -----
38
39 module divider( reset , clk , type , two_ops , operator , left_op ,
    ↪ right_op , result);
40     input          reset , clk , two_ops ;
41     input [1:0] type;
42     input [7:0] operator;
43     input [3:0] left_op , right_op;
44     output reg [3:0] result;
45
46 initial

```

```

47 begin
48   result  <= 4'd0;
49 end
50
51 always @(
52   posedge clk or posedge reset )
53 begin
54   if ( reset )
55     begin
56       result <= 4'd0;
57     end
58   else if ( type == 2'd1 && two_ops && operator == 8'h4A )
59     if      ( right_op == 4'd0 )      result <= 4'b1111;
60   else if ( left_op < right_op )    result <= 4'd0;
61   else if ( left_op == right_op )   result <= 4'd1;
62   else if ( right_op == 4'd1 )      result <= left_op;
63   else if( left_op > right_op )
64     case ( {left_op, right_op} )
65       8'b00110010: result <= 4'd1; // 3/2
66       8'b01000011: result <= 4'd1; // 4/3
67       8'b01000010: result <= 4'd2; // 4/2
68       8'b01010100: result <= 4'd1; // 5/4
69       8'b01010011: result <= 4'd1; // 5/3
70       8'b01010010: result <= 4'd2; // 5/2
71       8'b01100101: result <= 4'd1; // 6/5
72       8'b01100100: result <= 4'd1; // 6/4
73       8'b01100011: result <= 4'd2; // 6/3
74       8'b01100010: result <= 4'd3; // 6/2
75       8'b01110110: result <= 4'd1; // 7/6
76       8'b01110101: result <= 4'd1; // 7/5
77       8'b01110100: result <= 4'd1; // 7/4
78       8'b01110011: result <= 4'd2; // 7/3
79       8'b10000111: result <= 4'd1; // 8/7
80       8'b10000110: result <= 4'd1; // 8/6
81       8'b10000101: result <= 4'd1; // 8/5
82       8'b10000100: result <= 4'd2; // 8/4
83       8'b10000011: result <= 4'd2; // 8/3
84       8'b10000010: result <= 4'd4; // 8/2
85       8'b10011000: result <= 4'd1; // 9/8
86       8'b10010111: result <= 4'd1; // 9/7
87       8'b10010110: result <= 4'd1; // 9/6
88       8'b10010101: result <= 4'd1; // 9/5
89       8'b10010100: result <= 4'd2; // 9/4
90       8'b10010011: result <= 4'd3; // 9/3
91       8'b10010010: result <= 4'd4; // 9/2
92     endcase
93   end
94 endmodule

```

Κατά τη λειτουργία του χυκλώματος, πρώτα γίνεται μετατροπή των αριστερά και δεξιά

τελεστέων από scancode σε αριθμούς μέσω του module *code_2_num* του οποίου το λογισμικό περιγράφεται στον κώδικα 8.3.

Κώδικας 8.3

```

1 module code_2_num ( scancode , number );
2   input  [7:0] scancode;
3   output [3:0] number;
4
5   assign number =
6     ( scancode == 8'h45 ) ? 4'd0:
7     ( scancode == 8'h16 ) ? 4'd1:
8     ( scancode == 8'h1E ) ? 4'd2:
9     ( scancode == 8'h26 ) ? 4'd3:
10    ( scancode == 8'h25 ) ? 4'd4:
11    ( scancode == 8'h2E ) ? 4'd5:
12    ( scancode == 8'h36 ) ? 4'd6:
13    ( scancode == 8'h3D ) ? 4'd7:
14    ( scancode == 8'h3E ) ? 4'd8:
15    ( scancode == 8'h46 ) ? 4'd9: 4'hE;
16
17 endmodule

```

Έπειτα καλείται ένα στιγμιότυπο του module *divider* ώστε να υπολογιστεί το πηλίκο μόνο αν χρειαστεί. Το πηλίκο είναι τώρα είσοδος στο module *calculator* το οποίο με τη σειρά του έχει σαν έξοδο το αποτέλεσμα της πράξης. Το αποτέλεσμα αυτό αναπαρίσταται στο 7-segment LED της πλακέτας XSA μέσω του module *res_2_7segXSA*, του οποίου το λογισμικό περιγράφεται στον κώδικα 8.4.

Κώδικας 8.4

```

1 module res_2_7segXSA ( apotelesma , res );
2   input  [7:0] apotelesma;
3   output [7:0] res;
4
5   assign res = ( apotelesma == 7'd0 ) ? 8'b11101110: //0
6     ( apotelesma == 7'd1 ) ? 8'b00100100: //1
7     ( apotelesma == 7'd2 ) ? 8'b10111010: //2
8     ( apotelesma == 7'd3 ) ? 8'b10110110: //3
9     ( apotelesma == 7'd4 ) ? 8'b01110100: //4
10    ( apotelesma == 7'd5 ) ? 8'b11010110: //5
11    ( apotelesma == 7'd6 ) ? 8'b11011110: //6
12    ( apotelesma == 7'd7 ) ? 8'b11100100: //7
13    ( apotelesma == 7'd8 ) ? 8'b11111110: //8
14    ( apotelesma == 7'd9 ) ? 8'b11110110: 8'b11011010; //9, E
15 endmodule

```

Εδώ σε περίπτωση που ο προς αναπαράσταση αριθμός είναι μεγαλύτερος του 9, θα εμφανιστεί το γράμμα E, δηλαδή Error. Η τελική μορφή του top module *eight* περιγράφεται στον κώδικα 8.5.

Κώδικας 8.5

```

1  module eight (reset, clk, ps2clk, ps2data, left, right, res);
2      input          reset, clk;
3      input          ps2clk, ps2data;
4      output [7:0]   left, right, res;
5      wire   [7:0]   scan, current, previous, operator;
6      wire   [3:0]   left_op, right_op, phliko, apotelesma;
7      wire   [1:0]   type;
8      wire          button_flag, two_ops;
9
10     kbd_protocol   kbd (reset, clk, ps2clk, ps2data, button_flag,
11                         ↪ scan);
12     check_type      cht (button_flag, scan, type);
13     display_previous dp  (reset, clk, button_flag, type, scan,
14                         ↪ two_ops, current, previous, operator);
15     code_2_num       c2nL(current, left_op);
16     code_2_num       c2nR(previous, right_op);
17     divider         phl (reset, clk, type, two_ops, operator, left_op,
18                         ↪ right_op, phliko);
19     calculator       calc(reset, clk, type, two_ops, operator,
20                         ↪ left_op, right_op, phliko, apotelesma);
21     scan_2_7seg     lft (current, left);
22     scan_2_7seg     rght(previous, right);
23     res_2_7segXSA   rs  (apotelesma, res);
24
25 endmodule

```


Project

9.1 Εισαγωγή

Τα αρχεία Verilog που γράφηκαν για την υλοποίηση της εργασίας είναι:

- `clock.v`, που περιέχει τους απαραίτητους clock dividers για την άσκηση, δηλαδή ένα module που παράγει τον παλμό των $25MHz$ για το VGA και ένα module με συχνότητα $.5Hz$ που χρησιμοποιείται για τη ζητούμενη λειτουργία flashing
- `kbd_controller.v`, που αποτελεί το driver του πληκτρολογίου και βασίζεται στο `kbd_protocol.v` της άσκησης 8
- `horizontal_counter.v` και `vertical_counter.v`, που περιέχουν τα modules με τους δείκτες - μετρητές των pixels και των rows της οθόνης
- `vga_controller.v`, που αποτελεί το driver της οθόνης με σύνδεση τύπου VGA
- `top.v`, που αποτελεί τη `main` της εργασίας
- `mapper.ucf`, που περιέχει τα location constraints του σχεδιασμού

9.2 Το πληκτρολόγιο

Ο χρήστης ζητείται από την εκφώνηση να μπορεί να δίνει είσοδο μέσω του πληκτρολογίου, έτσι ώστε να πραγματοποιούνται διάφορες αλλαγές στο παραλληλόγραμμο που περιγράφεται στην εκφώνηση. Το module `kbd_controller` περιέχει το λογισμικό για αυτόν το σκοπό. Ως είσοδο, το `kbd_controller` δέχεται τα σήματα `reset`, `clk`, `ps2clk` και `ps2data` τα οποία είναι γνωστά από την άσκηση 8. Ωστόσο, η έξοδος του `kbd_controller` είναι διαφορετική από πριν. Το module έχει προγραμματιστεί έτσι ώστε να κάνει επεξεργασία του πλήκτρου που πατήθηκε on-the-fly. Ουσιαστικά, αντί να δίνουμε ως έξοδο πλέον το `scancode` του πλήκτρου που πατήθηκε, επιστρέφουμε κατευθείαν στο κατάλληλο module τις αλλαγμένες μεταβλητές με βάση αυτό το πλήκτρο. Οι μεταβλητές που επηρεάζονται ανάλογα με την είσοδο που δίνει ο χρήστης είναι το μήκος (`width`) και το ύψος (`height`) του παραλληλογράμμου του οποίου η περιφέρεια πρέπει να εμφανίζεται με άσπρο χρώμα. Επίσης, ο χρήστης έχει τη δυνατότητα να αντιστρέψει την πολικότητα της εικόνας, το οποίο υλοποιείται με μια μεταβλητή `rev`. Τέλος, ο χρήστης μπορεί να ενεργοποιεί τη λειτουργία flashing της οθόνης, το οποίο υλοποιείται με μια μεταβλητή `f1`. Αυτές είναι οι μεταβλητές που επιστρέφει ο driver του πληκτρολογίου στο driver της οθόνης, έτσι ώστε να υπολογιστεί η αντίστοιχη έξοδος. Το λογισμικό για το πληκτρολόγιο είναι στον κώδικα 9.1.

Κώδικας 9.1

```
1 module kbd_controller (reset, clk, ps2clk, ps2data, height, width
  ↵ , rev, f1);
```

```

2
3      input      reset, clk, ps2clk, ps2data;
4      output     height;
5      output     width;
6      output     rev;
7      output     fl;
8      reg [7:0]  scancode;
9      reg        fl, rev;
10     reg [9:0]  width;
11     reg [8:0]  height;
12     reg [9:0]  swap;
13     reg        explode;
14     reg [31:0] divider;
15
16 //                                     ← Synchronize
17 //                                     ← ps2clk
18 //                                     ← to local
19 //                                     ← clock
20 //                                     ← and
21 //                                     ← check
22 //                                     ← for
23 //                                     ← falling
24 //                                     ← edge
25 // Stores
26 // indicates
27 // Stores a
28 // Used to
29 // Used to
30
31 reg [7:0] ps2clksamples;           // last 8 ps2clk samples
32 wire      fall_edge;              // indicates a falling_edge at ps2clk
33 reg [9:0] shift;                 // serial package, excluding the stop bit
34 reg [3:0] cnt;                   // count the ps2data samples stored so far
35 reg        f0;                   // indicate that f0 was encountered earlier
36
37 parameter V_ROWS    = 400;
38 parameter H_PIXELS = 640;
39 parameter CYCLES   = 50000000;
40
41 initial begin
42     width  <= 10'd20;
43     height <= 9'd10;
44     divider<= 32'b0;
45     fl     <= 1'b0;               // flash mode
46     ↪ OFF
47     rev    <= 1'b0;               // reverse
48     ↪ image OFF
49     explode<= 1'b0;              // explode
50     ↪ mode OFF
51 end

```

```

35
36     always @(posedge clk or posedge reset) begin
37         if (reset) begin
38             ps2clksamples <= 8'd0;
39         end
40         else begin
41             ps2clksamples <= {ps2clksamples[7:0], ps2clk};
42         end
43     end
44
45     assign fall_edge = (ps2clksamples[7:4] == 4'hF) & (
46         ↪ ps2clksamples[3:0] == 4'h0);
47
48     /*
49      A simple FSM is implemented here. Grab a whole package,
50      check its parity validity and output it in the scancode
51      only if the previous read value of the package was F0
52      that is, we only trace when a button is released, NOT when it
53      ↪ is
54      pressed.
55 */
56
57     always @ (posedge clk or posedge reset) begin
58         if (explode == 1) begin
59             if (divider == CYCLES - 1) begin
60                 divider = 32'b0;
61                 if ((width + 10'd20 >= H_PIXELS) || (height + 9'
62                     ↪ d20 >= V_ROWS)) begin
63                     width = 10'd20;
64                     height = 9'd10;
65                 end
66                 else begin
67                     width = width + 10'd10;
68                     height = height + 9'd10;
69                 end
70             end
71             else begin
72                 divider = divider + 32'b1;
73             end
74         end
75         if (reset) begin
76             divider = 32'b0;
77             explode = 1'b0;
78             cnt <= 4'd0;
79             scancode = 8'd0;
80             shift <= 10'd0;
81             f0 = 1'b0;
82             height = 9'd10;
83             width = 10'd20;
84         end
85     end
86 
```

```

81          fl      = 1'd0;                                // flash mode
82          ↪ OFF
83          rev     = 1'd0;                                // reverse
84          ↪ image OFF
85      end
86      else if (fall_edge) begin
87          if (cnt == 4'd10) begin                         // we just
88              ↪ received what should be the stop bit
89              cnt <= 0;
90              if ((shift[0] == 0) && (ps2data == 1) && (^shift[
91                  ↪ 9:1]==1)) begin
92                  ↪ A well
93                  ↪ received
94                  ↪ serial
95                  ↪ packet
96                  if (f0) begin                           // following
97                      ↪ a scancode of f0. So a key is released !
98                      scancode = shift[8:1];
99                      f0 = 0;
100                     case(scancode)
101                         8'h75: height = ( height <= V_ROWS -
102                             ↪ 9'd20 ) ? height + 9'd10 :
103                             ↪ height;
104                         8'h72: height = ( height >= 9'd20 )
105                             ↪ ? height - 9'd10 :
106                             ↪ height;
107                         8'h6B: width  = ( width <= H_PIXELS -
108                             ↪ 10'd20 ) ? width + 10'd10 :
109                             ↪ width;
110                         8'h74: width  = ( width >= 10'd30 )
111                             ↪ ? width - 10'd10 :
112                             ↪ width;
113                         8'h24: explode= ~explode;
114                         8'h4D: begin
115                             swap = width;
116                             if (swap < V_ROWS) begin
117                                 width  = height;
118                                 height = swap;
119                             end
120                         end
121                         8'h2B: fl  = ~fl;
122                         8'h2D: rev = ~rev;
123                     endcase
124                 end
125             else if (shift[8:1] == 8'hF0) begin
126                 f0 = 1'b1;
127             end
128         end
129     end                                              // All other
130     ↪ packets have to do with key presses and are
131     ↪ ignored

```

```

113      end
114  else begin
115      shift <= {ps2data, shift[9:1]};      // Shift
116      ↪ right since LSB first is transmitted
117      cnt   <= cnt+1;
118  end
119 end
120 endmodule

```

9.3 Η οθόνη

Η λειτουργία της οθόνης είναι συνδυασμός τριών modules:

- Του **horizontal_counter**. Το module αυτό μετράει τα pixels μιας οριζόντιας γραμμής από το 0 έως το 799, δηλαδή συνολικά 800 pixels. Η τιμή 800 προκύπτει από την πρόσθεση των παραμέτρων H_FRONT_PORCH + HSYNC_PULSE + H_BACK_PORCH + VISIBLE_PIXELS = 16 + 96 + 48 + 640 = 800. Όταν ολοκληρωθούν τα pixels μια οριζόντιας γραμμής και μόνο τότε ενεργοποιείται το σήμα επίτρεψης μέτρησης των γραμμών ενός frame από το module vertical_counter.
- Του **vertical_counter**. Το module αυτό μετράει τις γραμμές ενός frame από το 0 έως το 448, δηλαδή 449 γραμμές συνολικά. Η τιμή 449 προκύπτει από την πρόσθεση των παραμέτρων V_FRONT_PORCH + VSYNC_PULSE + V_BACK_PORCH + VISIBLE_ROWS = 12 + 2 + 35 + 400 = 449. Αλλαγή γραμμής υπάρχει μόλις ολοκληρωθεί ένας κύκλος μέτρησης στο module horizontal_counter.
- Του **vga_controller**. Το module αυτό είναι υπεύθυνο για το συγχρονισμό του VGA display καθώς και για τα δεδομένα απεικόνισης. Τα σήματα χρονισμού h_sync και v_sync παράγονται ως έξοδοι. Μια οριζόντια γραμμή έχει πρώτα το H_FRONT_PORCH, μετά το HSYNC_PULSE, μετά το H_BACK_PORCH και τέλος τα VISIBLE_PIXELS. Έτσι, θα πρέπει:

```

h_count >= H_FRONT_PORCH           και
h_count <  H_FRONT_PORCH + HSYNC_PULSE, για να ενεργοποιηθεί το h_sync,
δηλαδή να τεθεί στο 0

```

Αντίστοιχα, μια κάθετη γραμμή έχει πρώτα το V_FRONT_PORCH, μετά το VSYNC_PULSE, μετά το V_BACK_PORCH και τέλος τα VISIBLE_ROWS. Έτσι, θα πρέπει:

```

v_count >= V_FRONT_PORCH           και
v_count <  V_FRONT_PORCH + VSYNC_PULSE, για να ενεργοποιηθεί το v_sync,
δηλαδή να τεθεί στο 1

```

Ο χρωματισμός της οθόνης θα γίνει με βάση την περιοχή *visible_area*, η οποία αρχικά θα είναι το παραλληλόγραμμο της εκφώνησης. Ότι ανήκει στην ορατή περιοχή θα χρωματίζεται με λευκό χρώμα, ενώ όλα τα υπόλοιπα pixels θα είναι μαύρα. Ανάλογες θα είναι οι χρωματικές αλλαγές σε περίπτωση που είμαστε σε mode *reverse* ή *flashing*.

- Του **clock**. Το clock περιέχει τα modules που υλοποιούν τη διαίρεση συχνότητας του ρολογιού του FPGA. Για την άσκηση, το Pixel Clock έχει συχνότητα ρολογιού στα $25MHz$, και το flashing στα $0.5Hz$.

Το λογισμικό σε Verilog για την περιγραφή των παραπάνω είναι οι κώδικες 9.2 έως και 9.5.

Κώδικας 9.2

```

1 module cnt4 ( reset , clk , enable , clkdiv4 );
2     input wire reset;
3     input wire clk;
4     input wire enable;
5     output wire clkdiv4;
6
7     reg [1:0] cnt;
8
9     assign clkdiv4 = (cnt == 2'b11);
10    always @ (posedge reset or posedge clk) begin
11        if (reset) begin
12            cnt <= 0;
13        end
14        else if (enable) begin
15            if (clkdiv4) begin
16                cnt <= 0;
17            end
18            else begin
19                cnt <= cnt + 1;
20            end
21        end
22    end
23 endmodule
24
25 module TwentyFiveMHz ( reset , clk , pulse );
26     input wire reset;
27     input wire clk;
28     output wire pulse;
29
30     wire clk25MHz;
31
32     cnt4 vga_freq ( reset , clk , 1'b1 , clk25MHz );
33
34     assign pulse = clk25MHz;
35 endmodule
36
37 /*
38
39 In this example, we are going to use this clock divider to
40 → implement a signal of exactly .5 Hz frequency.
41 First, we will need to calculate the constant.
42 As an example, the input clock frequency of the Spartan3 is 100
43 → MHz. We want our `flash` to be .5 Hz.

```

```

42 So it should take 200000000 clock cycles before `flash` goes to
43 → '1' and returns to '0'.
43 In another words, it takes 100000000 clock cycles for `flash` to
44 → flip its value.
44 So the constant we need to choose here is 100000000.
45
46 */
47
48 module HalfHertz( reset, clk, flash );
49
50   input wire reset;
51   input wire clk;
52   output reg flash;
53
54   parameter CYCLES = 100000000;
55
56   reg [31:0] count = 32'b0;
57
58   always @ (posedge reset or posedge clk) begin
59     if (reset == 1'b1) begin
60       count <= 32'b0;
61       flash <= 1'b0;
62     end
63     else if (count == CYCLES - 1) begin
64       count <= 32'b0;
65       flash <= ~flash;
66     end
67     else begin
68       count <= count + 1;
69       flash <= flash;
70     end
71   end
72
73 endmodule

```

Κώδικας 9.3

```

1 module horizontal_counter( reset, clk, enable_v_counter, h_count
2   );
3   input           reset, clk;                                // pixel
4   input           clock: 25MHz;
5   output reg      enable_v_counter;
6   output reg [9:0] h_count;                                // default
7   output reg [15:0] value [15:0];
8
9   initial begin
10     enable_v_counter <= 1'b0;
11     h_count          <= 10'd0;
12   end
13
14 endmodule

```

```

11      always @ (posedge clk or posedge reset) begin
12          if( reset ) begin
13              h_count <= 10'd0;
14              enable_v_counter <= 1'b0;
15          end
16          else if (h_count < 10'd799) begin
17              h_count           <= h_count + 1'b1;
18              enable_v_counter <= 1'b0;                                // disable
19                                         ↪ vertical counter
20          end
21          else begin
22              h_count           <= 1'b0;                                // reset
23                                         ↪ horizontal counter
24              enable_v_counter <= 1'b1;                                // trigger
25                                         ↪ vertical counter
26          end
27      end
28  endmodule

```

Κώδικας 9.4

```

1 module vertical_counter( reset, clk, enable_v_counter, v_count );
2     input             reset, clk;                                // pixel
3     input             enable_v_counter;
4     output reg [8:0] v_count;                                // default
5                                         ↪ value [15:0]
6
7     initial begin
8         v_count = 9'd0;
9     end
10
11    always @ (posedge clk or posedge reset) begin
12        if (reset) begin
13            v_count <= 9'd0;
14        end
15        else if (enable_v_counter == 1'b1) begin      // keep
16            ↪ counting until 449
17            if (v_count < 9'd448) begin
18                v_count <= v_count + 1'b1;
19            end
20            else begin
21                v_count <= 1'b0;                                // reset
22                                         ↪ vertical counter
23            end
24        end
25    end
26 endmodule

```

Κώδικας 9.5

```

1  module vga_controller( pulse25M, pulse05H, reset, height, width,
2      ↪ rev, fl, h_sync, v_sync, red, green, blue );
3  input      pulse25M, pulse05H, reset;
4  input [9:0] width;
5  input [8:0] height;
6  input      rev, fl;
7  output     h_sync, v_sync;
8  output [2:0] red;
9  output [2:0] green;
10 output [2:0] blue;

11 wire      enable_v_counter;
12 wire [9:0] h_count;
13 wire [8:0] v_count;
14 wire [2:0] flash;
15 wire [2:0] reverse;

16
17 wire [9:0] left, right;
18 wire [8:0] upper, down;

19
20 //horizontal parameters in pixels
21 parameter H_FRONT_PORCH = 16;
22 parameter H_SYNC          = 96;
23 parameter H_BACK_PORCH   = 48;
24 parameter H_PIXELS       = 640;
25 parameter H_NON_VISIBLE  = H_FRONT_PORCH + H_SYNC + H_BACK_PORCH
    ↪ ;
26
27 //vertical parameters in rows
28 parameter V_FRONT_PORCH = 12;
29 parameter V_SYNC          = 2;
30 parameter V_BACK_PORCH   = 35;
31 parameter V_ROWS          = 400;
32 parameter V_NON_VISIBLE  = V_FRONT_PORCH + V_SYNC + V_BACK_PORCH
    ↪ ;
33
34 //Pixel and line counters
35 horizontal_counter vga_h ( reset, pulse25M, enable_v_counter,
    ↪ h_count );
36 vertical_counter    vga_v ( reset, pulse25M, enable_v_counter,
    ↪ v_count );

37
38 //h_sync and v_sync timing
39 assign h_sync = ((h_count > H_FRONT_PORCH - 1'b1) && (h_count <
    ↪ H_FRONT_PORCH + H_SYNC )) ? 1'b0 : 1'b1;
40 assign v_sync = ((v_count > V_FRONT_PORCH - 1'b1) && (v_count <
    ↪ V_FRONT_PORCH + V_SYNC )) ? 1'b1 : 1'b0;

```

```

42 assign flash    = {3{f1 & pulse05H}};
43 assign reverse = {3{rev}};
44
45 assign left   = H_NON_VISIBLE - 1 + H_PIXELS/2 - width/2;
46 assign right  = H_NON_VISIBLE + H_PIXELS/2      + width/2;
47 assign upper  = V_NON_VISIBLE - 1 + V_ROWS/2     - height/2;
48 assign down   = V_NON_VISIBLE + V_ROWS/2       + height/2;
49
50 assign red    = (
51   ((h_count > left - 5) && (h_count < right + 5) && (((v_count >
52     ↪   upper - 5) && v_count <= upper) || ((v_count < down + 5)
53     ↪   && v_count >= down))) ||
54   ((v_count > upper - 5) && (v_count < down + 5) && (((h_count >
55     ↪   left - 5) && h_count <= left) || ((h_count < right + 5)
56     ↪   && h_count >= right))) ? (3'd7) ^ reverse ^ flash : (
57     ↪   h_count >= H_NON_VISIBLE && v_count >= V_NON_VISIBLE) ?
58     ↪   3'd0 ^ reverse : 3'd0;
59
60 assign blue   = (
61   ((h_count > left - 5) && (h_count < right + 5) && (((v_count >
62     ↪   upper - 5) && v_count <= upper) || ((v_count < down + 5)
63     ↪   && v_count >= down))) ||
64   ((v_count > upper - 5) && (v_count < down + 5) && (((h_count >
65     ↪   left - 5) && h_count <= left) || ((h_count < right + 5)
66     ↪   && h_count >= right))) ? (3'd7) ^ reverse ^ flash : (
67     ↪   h_count >= H_NON_VISIBLE && v_count >= V_NON_VISIBLE) ?
68     ↪   3'd0 ^ reverse : 3'd0;
69
70 assign green  = (
71   ((h_count > left - 5) && (h_count < right + 5) && (((v_count >
72     ↪   upper - 5) && v_count <= upper) || ((v_count < down + 5)
73     ↪   && v_count >= down))) ||
74   ((v_count > upper - 5) && (v_count < down + 5) && (((h_count >
75     ↪   left - 5) && h_count <= left) || ((h_count < right + 5)
76     ↪   && h_count >= right))) ? (3'd7) ^ reverse ^ flash : (
77     ↪   h_count >= H_NON_VISIBLE && v_count >= V_NON_VISIBLE) ?
78     ↪   3'd0 ^ reverse : 3'd0;
79
80 endmodule

```

To main - top module που τρέχει για την υλοποίηση των ζητουμένων της εργασίας είναι το top.v, του οποίου το λογισμικό υπάρχει στον κώδικα 9.6.

Κώδικας 9.6

```

1 module top( reset, clk, ps2clk, ps2data, h_sync, v_sync, red,
2   ↪   green, blue );
3   input      reset, clk, ps2clk, ps2data;
4   output     h_sync, v_sync;
5   output [2:0] red;

```

```

5   output [2:0] green;
6   output [2:0] blue;
7   wire      pulse25M, pulse05H;
8   wire      [9:0] width;
9   wire      [8:0] height;
10  wire      fl, rev, f0;
11
12 TwentyFiveMHz vga_clk( reset, clk, pulse25M );
13 HalfHz        flsh_clk( reset, clk, pulse05H );
14 kbd_controller kdb_ctrl( reset, clk, ps2clk, ps2data, height,
15                         ↪ width, rev, fl );
15 vga_controller vga_ctrl( pulse25M, pulse05H, reset, height,
16                         ↪ width, rev, fl, h_sync, v_sync, red, green, blue );
16 endmodule

```

9.4 Η λειτουργία Περιστροφή

Στα πλαίσια της εργασίας, υλοποιήσαμε μια επιπλέον λειτουργία, τη λειτουργία της περιστροφής. Σε περίπτωση που πατηθεί το πλήκτρο *P* - (Peristrophi), τότε τα μεγέθη *width* και *height* ανταλλάζουν τιμές και το σχήμα της ουδόνης αναστρέφεται. Η περιστροφή εκτελείται μόνο όταν το *width* δε ξεπερνά το όριο του *height*, δηλαδή την τιμή 400.

9.5 Το φαινόμενο *Explode*

Στα πλαίσια της εργασίας, υλοποιήσαμε το προαιρετικό φαινόμενο *explode*, το οποίο περιγράφεται στην εκφώνηση. Η συχνότητα της αλλαγής είναι 1 Hz. Το παραλληλόγραμμο επαναφέρεται αυτόματα στο αρχικό του μέγεθος σε περίπτωση που οποιαδήποτε αλλαγή ισοδυναμεί με οριακή τιμή, είτε στον άξονα *x* (*width*), είτε στον άξονα *y* (*height*). Το φαινόμενο εκτελείται παράλληλα με τις λειτουργίες *flashing*, *reverse* και *περιστροφή*. Η υλοποίηση έχει γίνει στο module *kbd_controller*.

9.6 Παράρτημα

Για τον ορισμό των location constraints, γράφηκε το αρχείο *mapper.ucf*, του οποίου τα περιεχόμενα φαίνονται στον κώδικα 9.7.

Κώδικας 9.7

```

1 net ps2clk loc=b16;
2 net ps2data loc=e13;
3 net clk loc=t9;
4 net reset loc=k4;
5 net h_sync loc=b7;
6 net v_sync loc=d8;
7 net red<2> loc=b1;
8 net red<1> loc=d6;
9 net red<0> loc=c8;

```

```
10 net green<2> loc=c3;  
11 net green<1> loc=a5;  
12 net green<0> loc=a8;  
13 net blue<2> loc=d5;  
14 net blue<1> loc=e7;  
15 net blue<0> loc=c9;
```