**4η Εργαστηριακή Άσκηση**

**VLSI**

Παντελαίος Δημήτριος Α.Μ.: 03118049

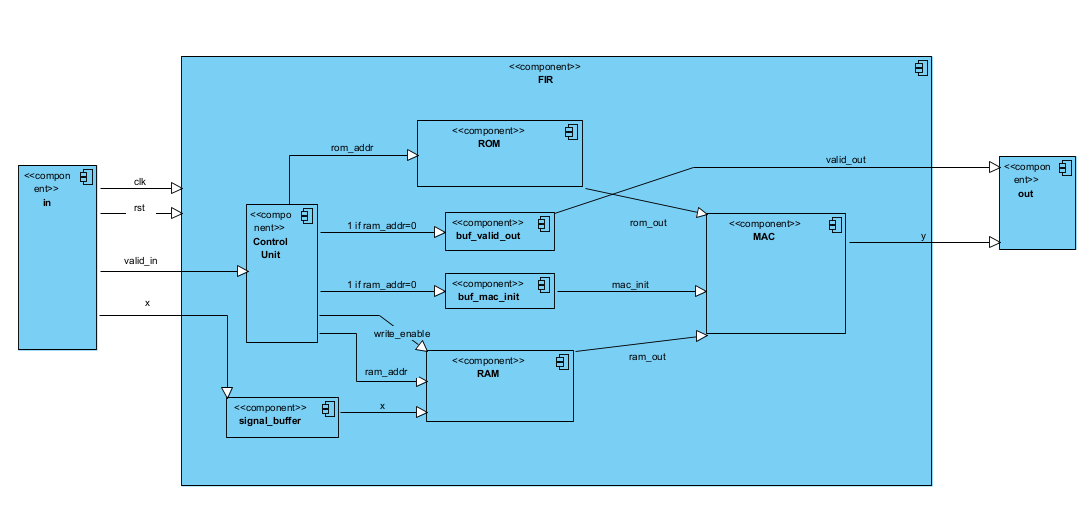
Μοίρας Αλέξανδρος Α.Μ.: 03118081

**Ζητούμενο 1:**

Components:

* Control Unit: Είναι υπεύθυνο για τον συντονισμό και τη λειτουργία όλου του υπόλοιπου φίλτρου. Συγκεκριμένα εκτός από το clock και το reset δέχεται ως είσοδο και το valid\_in σήμα. Παράγει ως εξόδους τις διευθύνσεις των RAM και ROM το write enable της RAM, το σήμα αρχικοποίησης του MAC καθώς και το valid\_out. Αν λάβει reset αρχικοποιεί τις διευθύνσεις των RAM και ROM ώστε να ξεκινήσουν από την πρώτη και την τελευταία θέση αντίστοιχα, στέλνει σήμα αρχικοποίησης στον MAC και θέτει το valid\_out στο 0. Κάθε 8 παλμούς ρολογιού, όπου έχει σαρωθεί όλη η RAM, ελέγχεται αν υπάρχει έγκυρη είσοδος και επίσης τίθεται το valid\_out στην τιμή 1 για τον προηγούμενο υπολογισμό. Αν υπάρχει αυτή εγγράφεται στη θέση 0 της RAM (we=1 και το control unit στέλνει τη διεύθυνση 0) ο MAC αρχικοποιείται μέσω του αντίστοιχου σήματος για να υπολογίσει τη νέα έξοδο του φίλτρου και ύστερα η διεύθυνση της RAM αυξάνεται κατά 1 σε κάθε κύκλο ρολογιού και της ROM μειώνεται κατά 1 ώστε να υπολογίζονται τα κατάλληλα γινόμενα και να προστίθενται στον MAC. Σε περίπτωση που ολοκληρωθεί ένας υπολογισμός και δεν υπάρχει έγκυρη είσοδος το σύστημα μπαίνει σε παύση μην προχωρώντας τις διευθύνσεις και μην παράγοντας valid out έως ότου δοθεί νέα έγκυρη είσοδος και η διαδικασία ξεκινήσει από την αρχή.
* RAM: Αποθηκεύονται οι είσοδοι που διαβάζονται για να χρησιμοποιηθούν στους υπολογισμούς. Έχει 8 θέσεις και κάθε νέα είσοδος αποθηκεύεται στη θέση 0 ενώ όλες οι προηγούμενες κάνουν shift μία θέση πάνω με την παλαιότερη να φεύγει από την RAM.
* ROM: Έχει επίσης 8 θέσεις που περιέχουν τα coefficients. Συγκεκριμένα στη θέση 0 περιέχεται το 8 (h[7]), στην 1 το 7 (h[6]) και ούτω καθεξής.
* MAC: Σε κάθε κύκλο ρολογιού αθροίζει στο αποτέλεσμα που έχει ήδη αποθηκευμένο, το γινόμενο των αριθμών που λαμβάνει από RAM και ROM ώστε να σχηματίσει το τελικό αποτέλεσμα. Αν λάβει σήμα αρχικοποίησης μηδενίζει την τιμή εντός του ώστε να υπολογίσει νέο άθροισμα γινομένων.

Layout του συστήματος: Τα components που αναφέρθηκαν παραπάνω διασυνδέονται ως εξής:



Χρησιμοποιείται ένας buffer στην είσοδο, ώστε όταν έρθει νέα είσοδος και πρόκειται να εισέλθει στη RAM, το control unit να έχει προλάβει να δώσει σωστά τη διεύθυνση 0 στη RAM. Επίσης χρησιμοποιείται buffer για το valid out που παράγεται από το Control Unit ώστε να φτάσει στην έξοδο έναν κύκλο αφού παραχθεί (παράγεται όταν η διεύθυνση της RAM είναι 0 δηλαδή εκείνη τη στιγμή στην είσοδο του MAC βρίσκονται οι δύο τελευταίοι αριθμοί που πρέπει να πολλαπλασιαστούν) με σκοπό να έχει ολοκληρώσει ως τότε ο MAC τον υπολογισμό. Εντελώς αντίστοιχα και το mac\_init φτάνει έναν κύκλο αργότερα ώστε να έχει ολοκληρωθεί ο προηγούμενος υπολογισμός.

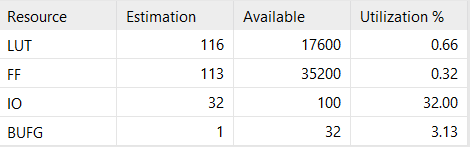
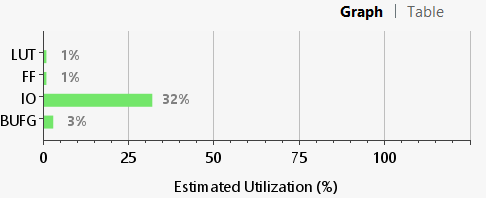
Θα μπορούσαμε να δίνουμε και το valid out σαν είσοδο στον MAC ώστε να δείχνει στην έξοδό του το αποτέλεσμα που έχει αποθηκευμένο εντός μόνο όταν πρόκειται για το τελικό και όχι για κάποιο ενδιάμεσο ώστε να αποφύγουμε την παραγωγή παρασιτικών παλμών που θα αυξήσει την κατανάλωση ενέργειας σε επόμενες βαθμίδες καθώς και στην ίδια.

Testbench:

Παρακάτω παραθέτουμε ένα τμήμα του testbench που επιβεβαιώνει την ορθή λειτουργία του κυκλώματος:

Κατανάλωση πόρων:

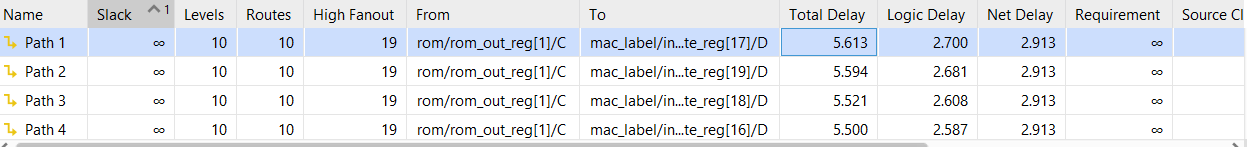
Οι πόροι που καταναλώθηκαν για το συγκεκριμένο φίλτρο είναι οι εξής:



Παρατηρούμε κατανάλωση LUTs για τους υπολογισμούς, Flip Flops για τους registers, ΙΟs καθώς και buffers για το ρολόι.

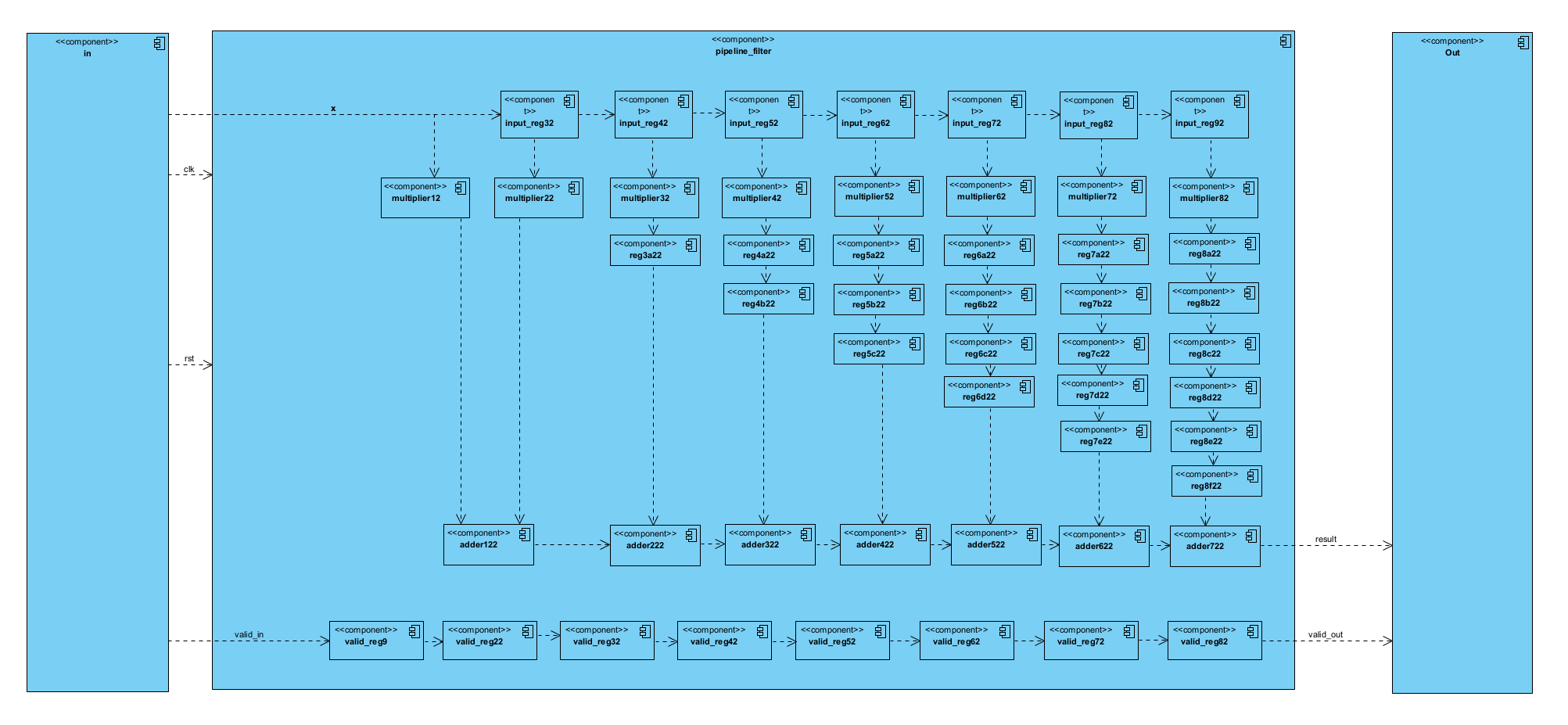
Critical Path:

Το κρίσιμο μονοπάτι είναι από την έξοδο της ROM έως τον MAC με καθυστέρηση 5.613ns.



**Ζητούμενο 2:**

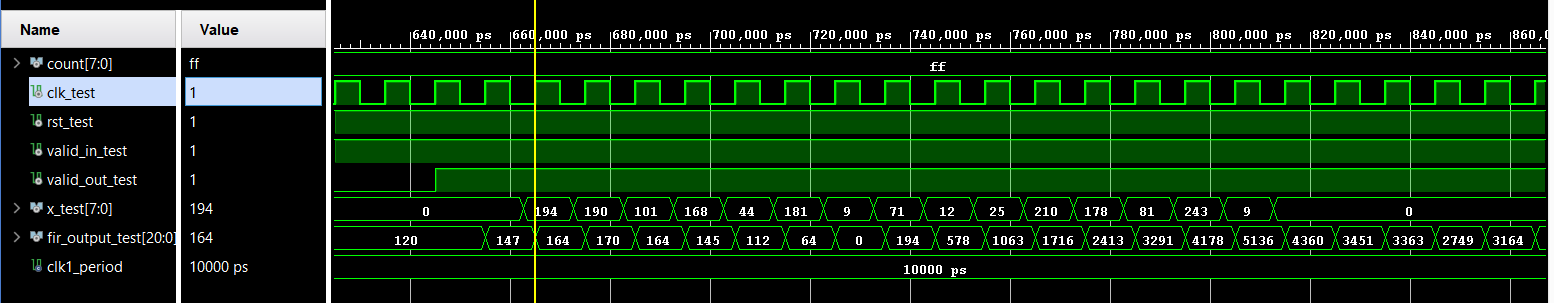
Για την κατασκευή του pipelined FIR φίλτρου των 8 bits και 8 συντελεστών θα χρησιμοποιηθούν σύγχρονοι αθροιστές και πολλαπλασιαστές καθώς επίσης και registers για τον συγχρονισμό των δεδομένων στις εισόδους των components. Το σχηματικό του διαγράμματος παρουσιάζεται παρακάτω:



Κρατάμε σε buffers τις προηγούμενες εισόδους για να τις χρησιμοποιήσουμε για τους υπολογισμούς. Κάθε φορά που έρχεται νέα είσοδος μπαίνει άμεσα στον πρώτο πολλαπλασιαστή και στην είσοδο του πρώτου register, ώστε να υπολογιστεί το πρώτο γινόμενο του αθροίσματος. Τα γινόμενα υπολογίζονται όλα ταυτόχρονα ωστόσο εισέρχονται στους αθροιστές με καθυστέρηση προκειμένου να έχει ολοκληρωθεί η προηγούμενη άθροιση. Μετά από ένα latency 8 κύκλων ξεκινούν να παράγονται αποτελέσματα και παράγεται νέο αποτέλεσμα σε κάθε κύκλο. Οπότε η έξοδος θα είναι έγκυρη 8 κύκλους αφού δοθεί έγκυρη είσοδος.

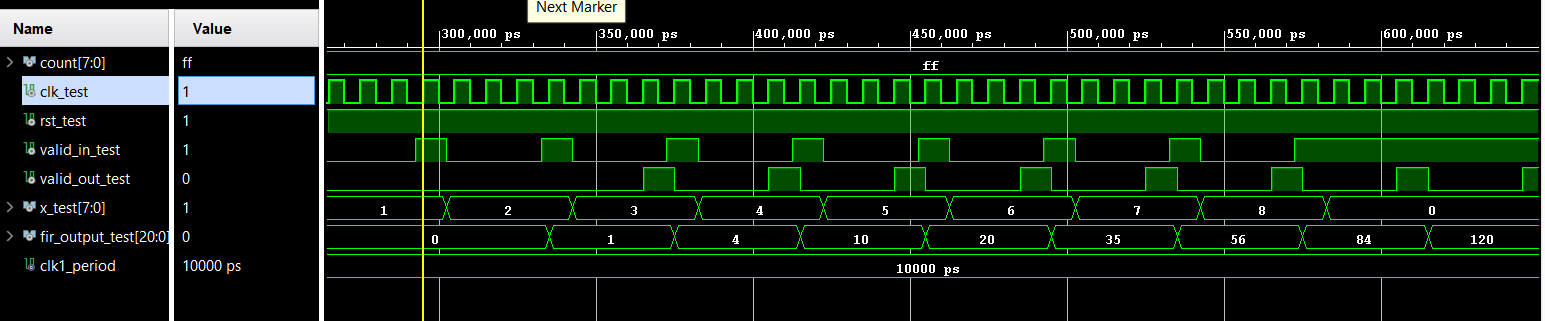
Οι buffers εισόδου ελέγχουν στη θετική ακμή του ρολογιού και την εγκυρότητα της εισόδου μέσω του σήματος valid\_in ώστε να μην αποθηκεύσουν κάποια μη έγκυρη είσοδο που θα αλλοίωνε τα μετέπειτα αποτελέσματα:

Τμήμα του testbench που επιβεβαιώνει την ορθή λειτουργία του κυκλώματος:

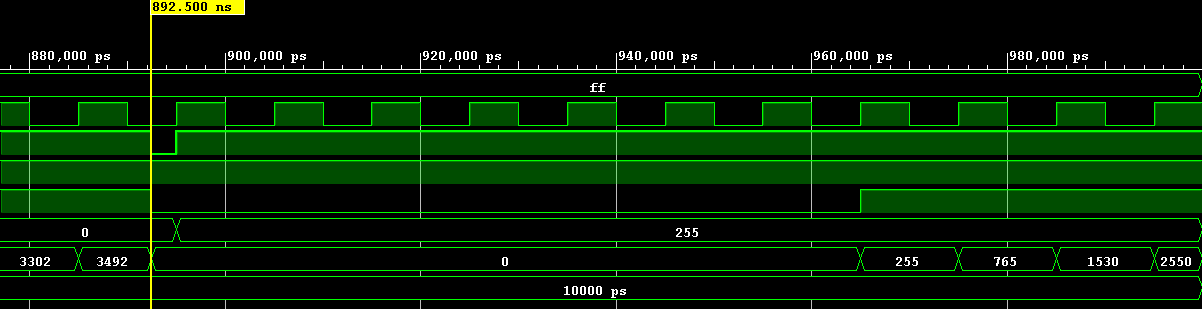


Όπου παρατηρούμε ότι 8 κύκλους ρολογιού αφού δοθεί το πρώτο σήμα (194) αρχίζουν να παράγονται τα σωστά αποτελέσματα από το φίλτρο σε κάθε κύκλο ρολογιού.

Για την κατάσταση όπου δε δίνεται συνεχώς valid\_in:



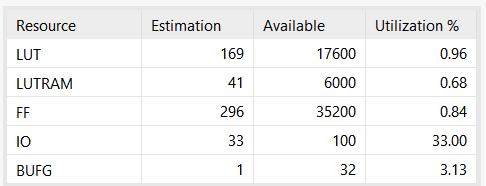
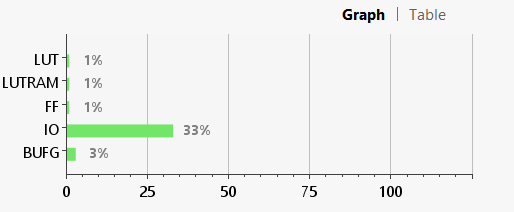
Και για το reset:



όπου βλέπουμε ότι αμέσως μηδενίζεται η έξοδος και το valid\_out ενώ 8 κύκλους αφού το σήμα reset επανέλθει στο 1 έχουμε πάλι valid έξοδο στο φίλτρο.

Κατανάλωση πόρων:

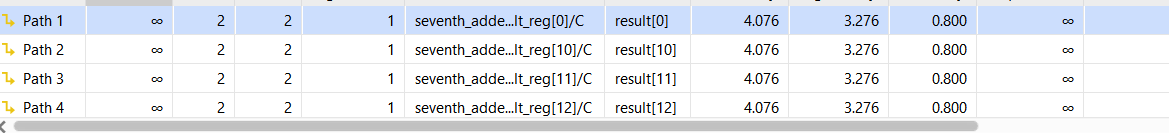
Οι πόροι που καταναλώθηκαν για το συγκεκριμένο φίλτρο είναι οι εξής:

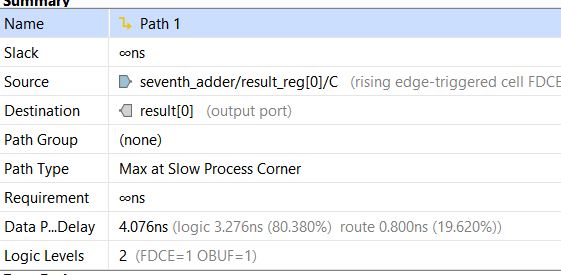


Αυτό το φίλτρο σε αντίθεση με το προηγούμενο χρησιμοποιεί και RAM γεγονός οξύμωρο καθώς τώρα δεν έχουμε κατασκευάσει κάποιο RAM Module, όπως είχαμε κάνει πριν. Ωστόσο στο προηγούμενο ζητούμενο το RAM Module που κατασκευάσαμε είχε την ιδιότητα να κάνει shift τις τιμές στις θέσεις μνήμης του κάτι που δεν μπορεί να γίνει πάνω στη RAM του FPGA οπότε πιθανότατα υλοποιήθηκε με LUT.

Όσον αφορά τους υπόλοιπους πόρους καταναλώνει περισσότερους κάτι που είναι λογικό αφού έχει πολύ περισσότερους buffers (άρα και flip flops) και πολύ περισσότερες υπολογιστικές μονάδες (πολλαπλασιαστές, αθροιστές σε αντίθεση με έναν MAC πριν άρα περισσότερα LUTs) ώστε να υποστηρίξει το pipeline.

Critical Path:

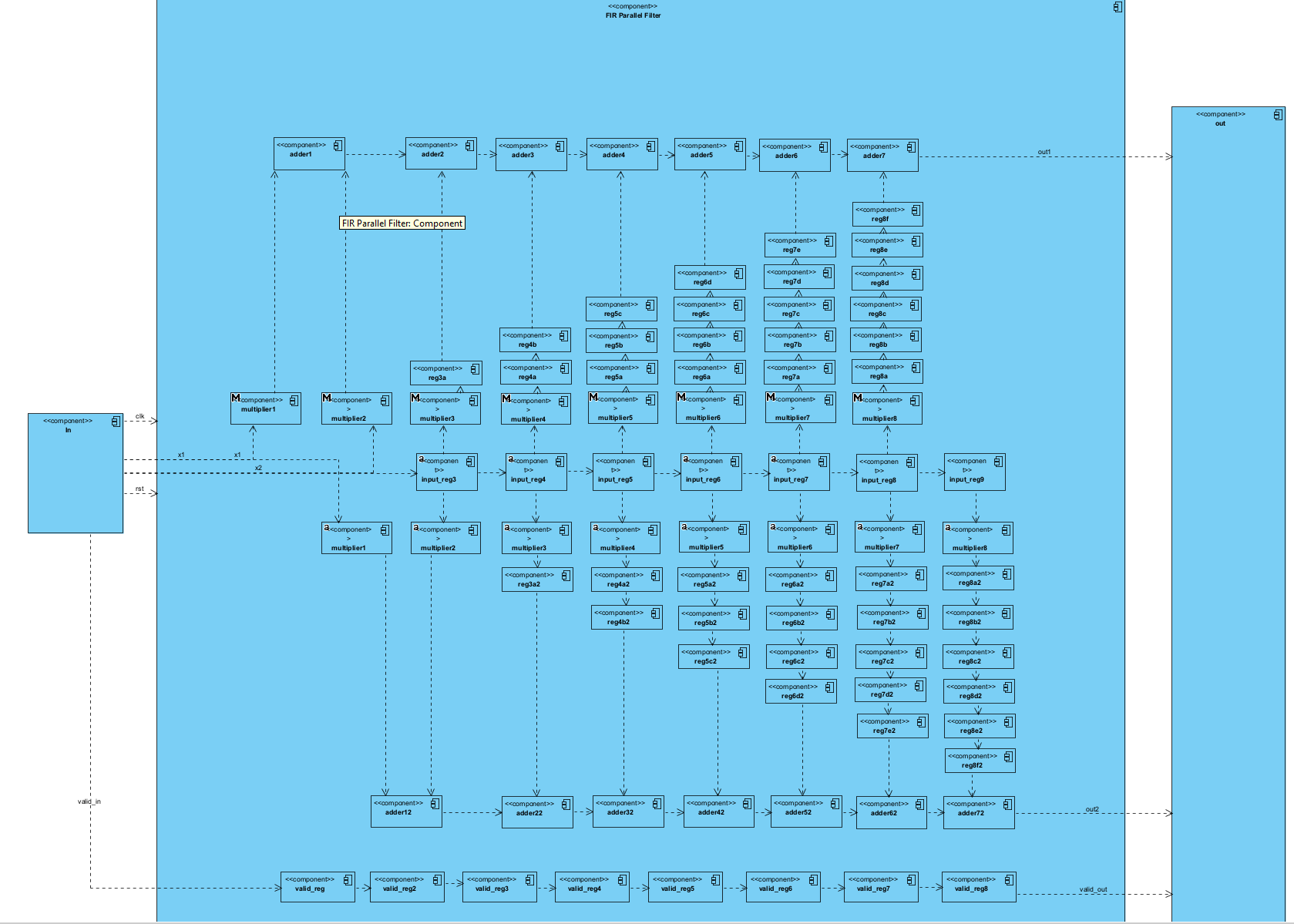




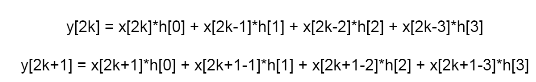
Το κρίσιμο μονοπάτι είναι από τον τελευταίο αθροιστή μέχρι το πρώτο bit του αποτελέσματος με καθυστέρηση 4.076ns. Επομένως η μέγιστη συχνότητα λειτουργίας που μπορεί να υποστηριχτεί είναι 245.33MHz.

**Ζητούμενο 3:**

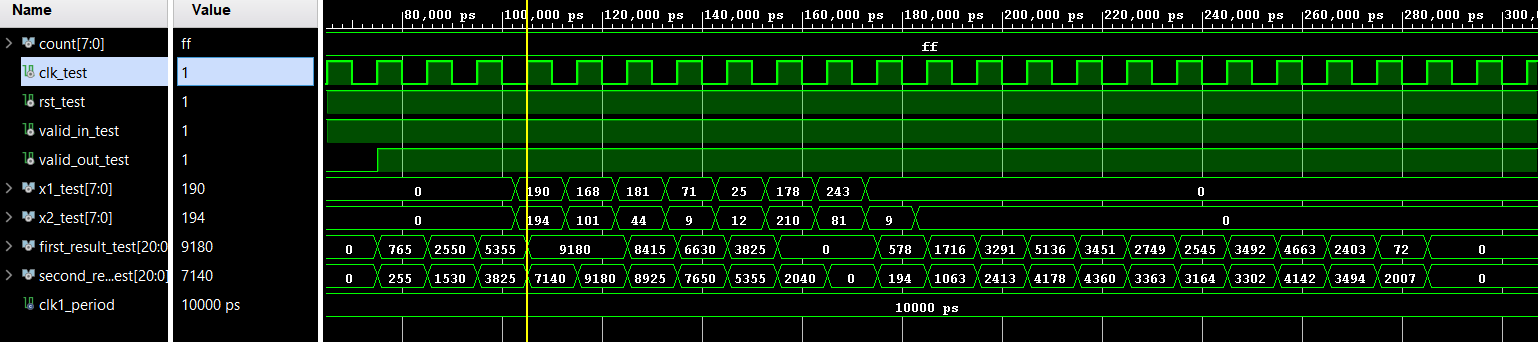
Για να μετατρέψουμε το προηγούμενο φίλτρο σε παράλληλο πρέπει πρακτικά να διπλασιάσουμε το hardware που εκτελεί τους υπολογισμούς. Συγκεκριμένα σε κάθε κύκλο ρολογιού θα διαβάζεται τόσο το x[n] όσο και το x[n-1] και θα δίνονται στους registers. Το ένα υποφίλτρο θα χρησιμοποιεί τους πρώτους 8 registers εισόδου ενώ το άλλο τους τελευταίους 8. Η λειτουργία του παράλληλου φίλτρου αποτυπώνεται καλύτερα στο παρακάτω διάγραμμα:



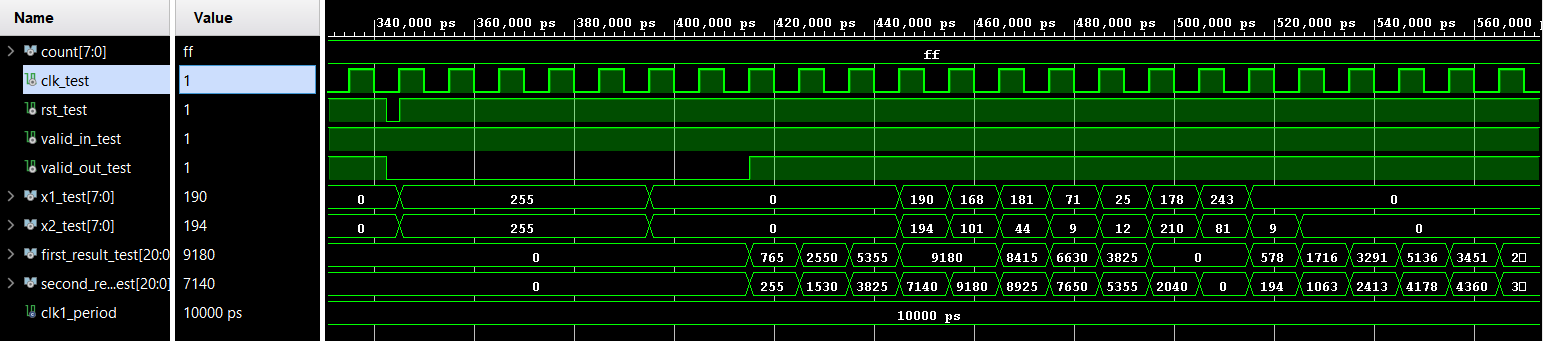
Συγκεκριμένα το x1 αντιστοιχεί στο x[2k+1] και το x2 στο x[2k] δηλαδή το σήμα που είχε έρθει την προηγούμενη χρονική στιγμή. Η έξοδος out2 αντιστοιχεί στο y[2k] και η out1 στο y[2k+1] σύμφωνα με την παρακάτω εξίσωση:



Testbench: Μέρος του testbench που επιβεβαιώνει την ορθή λειτουργία του κυκλώματος:

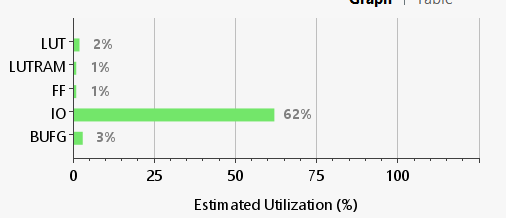
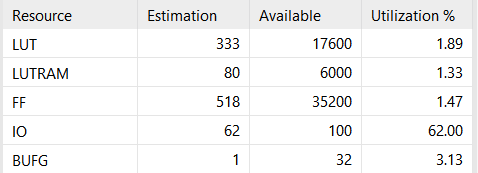


Παρατηρούμε ότι 8 κύκλους μετά την είσοδο ζευγαριού έγκυρων εισόδων παράγεται η σωστή έγκυρη έξοδος. Επίσης για τον έλεγχο του reset:



Κατανάλωση πόρων:

Οι πόροι που καταναλώθηκαν για το συγκεκριμένο φίλτρο είναι οι εξής:

Παρατηρούμε πρακτικά διπλάσια κατανάλωση των ίδιων πόρων με το Pipelined μη παράλληλο FIR φίλτρο του προηγούμενου ζητουμένου κάτι που είναι αναμενόμενο καθώς όπως φαίνεται και στο διάγραμμα για να κατασκευάσουμε το παράλληλο φίλτρο πρακτικά διπλασιάσαμε σχεδόν όλες τις δομικές μονάδες κατασκευάζοντας δύο Pipelined φίλτρα μέσα σε ένα.

Critical Path:

Το κρίσιμο μονοπάτι είναι από τον τελευταίο αθροιστή (διάφορα bits του) έως το αντίστοιχο bit της εξόδου με καθυστέρηση 4.076ns. Επομένως η μέγιστη συχνότητα λειτουργίας που μπορεί να υποστηριχτεί είναι 245.33MHz ακριβώς όπως στο απλό Pipelined φίλτρο. Άρα ουσιαστικά με αυτό το φίλτρο μπορούμε να δεχόμαστε δεδομένα με διπλάσιο ρυθμό σε σχέση με το απλό pipelined πληρώνοντας αυτή τη διπλάσια ταχύτητα με διπλάσιο hardware πάνω στο FPGA.

