

© 2019 Αραβαντινός Λεωνίδης Χρήστος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ
ΣΥΣΤΗΜΑΤΩΝ

Διπλωματική Εργασία

**ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΣΕ ΥΛΙΚΟ ΨΗΦΙΑΚΟΥ
ΦΙΛΤΡΟΥ ΑΠΟΘΡΥΒΟΠΟΙΗΣΗΣ ΕΙΚΟΝΩΝ ΜΕ ΧΡΗΣΗ
ΤΕΧΝΙΚΩΝ ΣΥΝΘΕΣΗΣ ΥΨΗΛΟΥ ΕΠΙΠΕΔΟΥ**

ΑΡΑΒΑΝΤΙΝΟΣ ΛΕΩΝΙΔΗΣ ΧΡΗΣΤΟΣ

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Διπλώματος Μηχανικού Πληροφοριακών και Επικοινωνιακών Συστημάτων

2019

Η ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΔΙΔΑΣΚΟΝΤΩΝ ΕΠΙΚΥΡΩΝΕΙ
ΤΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΤΟΥ
ΑΡΑΒΑΝΤΙΝΟΥ ΛΕΩΝΙΔΗ ΧΡΗΣΤΟΥ

Εμμανουήλ Καλλίγερος, Επιβλέπων
Τμήμα Μηχανικών Πληροφοριακών και
Επικοινωνιακών Συστημάτων

Γεώργιος Κορμέντζας, Μέλος
Τμήμα Μηχανικών Πληροφοριακών και
Επικοινωνιακών Συστημάτων

Χρήστος Γκουμόπουλος, Μέλος
Τμήμα Μηχανικών Πληροφοριακών και
Επικοινωνιακών Συστημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΘΕΡΙΝΟ ΕΞΑΜΗΝΟ 2019

Περίληψη

Στις μέρες μας, όλο και περισσότεροι τομείς όπως η ιατρική, η επιτήρηση με οπτικά μέσα και η δορυφορική φωτογράφιση, απαιτούν την εξαγωγή πληροφορίας από εικόνες για την ερμηνεία του περιεχομένου τους. Αυτή η ανάγκη είναι ένας από τους βασικούς μοχλούς εξέλιξης της περιοχής της επεξεργασίας εικόνας. Το μεγάλο πλήθος δεδομένων προς επεξεργασία κάνουν τις υλοποιήσεις αλγορίθμων σε λογισμικό αυτής της περιοχής αρκετά αργές. Αντιθέτως, η υλοποίησή τους στο υλικό, μέθοδος γνωστή ως hardware acceleration, σε συνδυασμό με την χρήση συσκευών FPGA μπορεί να εξαλείψει όλους τους περιορισμούς του λογισμικού. Σε αυτή τη διπλωματική εργασία υλοποιούμε ένα φίλτρο αποθορυβοποίησης εικόνων, γνωστό και ως Διμερές Φίλτρο, αρχικά στο λογισμικό σε περιβάλλον MATLAB, και κατόπιν στο υλικό, καθώς το σχεδιάζουμε στο Simulink με την χρήση της βιβλιοθήκης DSP Builder της εταιρείας Intel (τέως Altera). Για την σύνθεση του, χρησιμοποιούμε το Quartus II. Τα Testbenches μας τα γράψαμε σε γλώσσα περιγραφής υλικού Verilog και τις τελικές εξομοιώσεις του φίλτρου τις πραγματοποιήσαμε στο εργαλείο Modelsim. Η σχεδίαση του φίλτρου έγινε στην έκδοση του DSP Builder που συνεργάζεται με το Quartus 15.0 και η οποία είναι ελεύθερη προς χρήση σε περιβάλλον MATLAB (Simulink). Τέλος, παρουσιάζουμε όλα τα αποτελέσματα και συμπεράσματά μας.



Abstract

In our days, an increasing amount of areas such as medicine, video surveillance and satellite image capturing, require image information extraction to interpret their content. This need is one of the main driving factors of the evolution of the area of image processing. The enormous amount of data that require processing tend to slow up the software implementations of the algorithms in this area. Instead, their implementation in hardware, also known as hardware acceleration, combined with the use of FPGA devices, can eliminate all the software restrictions. In this thesis we implement a filter for Image denoising known as Bilateral Filter, at first in software, using the MATLAB environment and afterwards in hardware, designing it in Simulink using Intel's (former Altera) DSP Builder library. For synthesis, we use Quartus II. Our Testbenches were written in Verilog hardware description language and our filter's final simulations were performed using Modelsim. The filter's design was made possible using the DSP Builder's version that collaborates with Quartus 15.0. Finally, we present all our results and conclusions.





Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου για την υπομονή, υποστήριξη και συμπαράσταση που μου έδειξαν σε όλη την διάρκεια της φοίτησής μου στο Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων. Ιδιαίτερες ευχαριστίες θα ήθελα να δώσω στον φίλο μου, από την πρώτη μέρα της σχολής, Γιάννη Αλεξίου. Αν και ακολουθήσαμε διαφορετικό αντικείμενο εν τέλει, η φιλία και η συνεργασία μας πάντα με βοηθούσε να αντιμετωπίσω την όποια δυσκολία και είχα πάντα ένα δυνατό παράδειγμα για να ανταγωνιστώ. Τέλος, δεν θα μπορούσα να παραλείψω να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Εμμανουήλ Καλλίγερο. Ο χαρακτήρας και η διδασκαλία του με βοήθησαν ώστε να διερευνήσω και τελικά να αγαπήσω το συγκεκριμένο αντικείμενο, ενώ η συμβολή του και το ενδιαφέρον του ήταν καθοριστικές ώστε να ολοκληρώσω την διπλωματική μου εργασία.





Περιεχόμενα

Περιεχόμενα.....	8
Κεφάλαιο 1 Εισαγωγή στα Ψηφιακά Φίλτρα.....	14
1.1 Πεδία Χρήσης Ψηφιακών Φίλτρων.....	15
1.2 Πως λειτουργεί ένα διμερές φίλτρο (Bilateral Filter)	16
1.3 Παράθυρο Φίλτρου (Kernel)	16
1.4 Σταθερές - Παράμετροι.....	16
Κεφάλαιο 2 Αριθμητικές Πράξεις.....	19
2.1 Φωτομετρικό Φιλτράρισμα.....	19
2.2 Γεωμετρικό Φιλτράρισμα.....	19
2.3 Εξομάλυνση.....	21
2.4 Άθροισμα.....	21
2.5 Υπολογισμός του υπό Φιλτράρισμα Pixel.....	22
Κεφάλαιο 3 Εξομοίωση φίλτρου σε περιβάλλον MATLAB	23
3.1 Τρόπος εισαγωγής θορύβου στην εικόνα.....	23
3.2 Μετατροπή εικόνας με θόρυβο σε αρχείο .mif	24
3.3 Χρήση μεθόδου zero-padding.....	24
3.4 Δημιουργία του συνόλου των Kernel.....	25
3.5 Υπολογισμός της Φωτομετρικής και Γεωμετρικής συνιστώσας	25
3.6 Υπολογισμός αθροίσματος, εξομάλυνσης και τελικής τιμής pixel	26
3.7 Αναδιάταξη Φιλτραρισμένης Εικόνας-MATLAB.....	27
Κεφάλαιο 4 Υλοποίηση του φίλτρου στο DSP Builder	28
4.1 Κριτήρια επιλογής μεγέθους αναπαράστασης των pixels στο υλικό	28
4.2 Χρήση Μεταβλητών στο DSP Builder από το MATLAB	29
4.3 Γενική Ανάλυση λειτουργίας του φίλτρου στο υλικό	30
4.4 Σχεδίαση του Register Matrix.....	31
4.5 Σχεδίαση της Φωτομετρικής Συνιστώσας.....	32
4.6 Σχεδίαση της Γεωμετρικής Συνιστώσας	33
4.7 Σχεδίαση του Procedure.....	34
4.8 Σχεδίαση της Διαίρεσης	35
Κεφάλαιο 5 Σύνθεση στο Quartus και Εξομοίωση φίλτρου στο Modelsim	36
5.1 Φίλτρο με ενσωματωμένη μνήμη	36





5.1.1	Testbench Φίλτρου.....	37
5.2	Φίλτρο χωρίς ενσωματωμένη μνήμη.....	38
5.2.1	Επιλογή πίνακα συνιστωσών βάση θορύβου	39
5.2.2	Δημιουργία μνήμης RAM μέσω του Quartus.....	40
5.2.3	Testbench Φίλτρου.....	41
5.3	Εξομοίωση Φίλτρου	42
Κεφάλαιο 6	Διασταύρωση αποτελεσμάτων Modelsim-Simulink	43
6.1	Τρόπος αποθήκευσης αποτελεσμάτων φίλτρου	43
6.2	Ανάγνωση Αποτελεσμάτων στο MATLAB	44
6.3	Δημιουργία της Φιльтраρισμένης Εικόνας-Simulink ή Modelsim	45
6.4	Έλεγχος τελικών αποτελεσμάτων Simulink – Modelsim.....	45
6.5	Υπολογισμός του MSE	46
6.6	Υπολογισμός του PSNR.....	46
6.7	Υπολογισμός του MSSIM.....	47
6.6.1	SSIM	47
6.6.2	Luminance	47
6.6.3	Contrast	48
6.6.4	Structure	48
6.6.5	Mean SSIM.....	49
Κεφάλαιο 7	Σύγκριση αποτελεσμάτων MATLAB-Simulink	50
7.1	Φιльтраρίσμα εικόνας 100x100	50
7.2	Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 100x100	51
7.3	Φιльтраρίσμα εικόνας 150x150	52
7.4	Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 150x150	53
7.5	Φιльтраρίσμα εικόνας 200x200	54
7.6	Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 200x200	55
7.7	Φιльтраρίσμα εικόνας 330x330	56
7.8	Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 330x330	57
7.9	Φιльтраρίσμα εικόνας 512x512	58
7.10	Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 512x512	59
7.11	Χρονικές Εκτιμήσεις	60
Κεφάλαιο 8	Απαιτήσεις σε πόρους.....	62





8.1	Φίλτρο με ενσωματωμένη μνήμη	62
8.2	Φίλτρο χωρίς ενσωματωμένη μνήμη	63
8.3	Διαφορές των δύο Υλοποιήσεων	64
Κεφάλαιο 9	Μελλοντικές Προσθήκες και Αλλαγές	65
9.1	Χρήση Παραθύρου διαφορετικού μεγέθους	65
9.2	Επαναληπτικό φιλτράρισμα με kernel 3x3	66
9.3	Επαναληπτικό φιλτράρισμα με kernel 5x5	67
9.4	Μετρήσεις επαναληπτικού φιλτραρίσματος για θόρυβο σnoise=15	68
9.5	Μετρήσεις επαναληπτικού φιλτραρίσματος για θόρυβο σnoise=35	69
9.6	Φιλτράρισμα έγχρωμων εικόνων RGB	70
	Βιβλιογραφία	72





Κατάλογος Εικόνων

Εικόνα 1:	Στάδια επεξεργασίας σήματος-πληροφορίας σε ένα ψηφιακό φίλτρο	14
Εικόνα 2:	Μοναδικοί Συντελεστές ανάλογα με την παράμετρο σ_{noise}	17
Εικόνα 3:	Κώδικας εισαγωγής θορύβου σε εικόνα.....	23
Εικόνα 4:	Διαφορές μεταξύ χρήσης και μη χρήσης της μεθόδου Zero Padding	24
Εικόνα 5:	Κώδικας δημιουργίας των kernels	25
Εικόνα 6:	Κώδικας υπολογισμού Φωτομετρικών και Γεωμετρικών συνιστωσών.....	26
Εικόνα 7:	Κώδικας υπολογισμού εξομάλυνσης, αθροίσματος και τελικής τιμής pixel.....	26
Εικόνα 8:	Κώδικας MATLAB αναδιάταξης φιλτραρισμένης εικόνας	27
Εικόνα 9:	Σχέδιο Λειτουργίας Διμερούς φίλτρου	30
Εικόνα 10:	Register Matrix.....	31
Εικόνα 11:	Δομή Υποσυστήματος Υπολογισμού Γεωμετρικής Συνιστώσας.....	32
Εικόνα 12:	Πίνακας Look-up για επιλογή τιμής της μεταβλητής C_0 με βάση σ_c	33
Εικόνα 13:	Τα υποσυστήματα C_0 και C_1	33
Εικόνα 14:	Εσωτερικό ενός εκ των έξι Procedures	34
Εικόνα 15:	Υποσύστημα Διαίρεσης.....	35
Εικόνα 16:	Μνήμη RAM και γειτονικά blocks από τη Standard βιβλιοθήκη	36
Εικόνα 17:	Κώδικας Verilog Testbench για Φίλτρο με ενσωματωμένη Μνήμη	37
Εικόνα 18:	Τοποθετώντας καταχωρητές εισόδου και εξόδου στη θέση της μνήμης RAM... ..	38
Εικόνα 19:	Επιλογή Πίνακα Look-up με βάση τον ορισμένο θόρυβο στο Testbench	39
Εικόνα 20:	Δημιουργία μνήμης RAM στο Quartus	40
Εικόνα 21:	Κώδικας Verilog Testbench για φίλτρο χωρίς ενσωματωμένη μνήμη	41
Εικόνα 22:	Εξομίωση στο Modelsim και επιβεβαίωση των αποτελεσμάτων	42
Εικόνα 23:	Ο τρόπος αποθήκευσης των εξαγόμενων δεδομένων από το Simulink	43
Εικόνα 24:	Blocks για έλεγχο τιμής και αποθήκευσης εξόδου	44
Εικόνα 25:	Κώδικας ανάγνωσης κύριας εικόνας και τιμών από αρχείο κειμένου	44
Εικόνα 26:	Κώδικας δημιουργίας τελικής εικόνας	45
Εικόνα 27:	Εντολή ελέγχου της ομοιότητας των εξαγόμενων αποτελεσμάτων.....	45
Εικόνα 28:	Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 100x100	50
Εικόνα 29:	Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 150x150	52
Εικόνα 30:	Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 200x200	54
Εικόνα 31:	Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 330x330	56
Εικόνα 32:	Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 512x512	58
Εικόνα 33:	Αποτελέσματα αποθορυβοποίησης για θόρυβο $\sigma_{noise}=40$	65
Εικόνα 36:	Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=15$ και kernel 3x3	66
Εικόνα 37:	Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=35$ και kernel 3x3	66
Εικόνα 38:	Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=15$ και kernel 5x5	67
Εικόνα 39:	Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=35$ και kernel 5x5	67
Εικόνα 40:	Παράλληλη αρχιτεκτονική φίλτρου για RGB εικόνες	70
Εικόνα 41:	Σειριακή αρχιτεκτονική φίλτρου για RGB εικόνες	71





Κατάλογος Πινάκων

Πίνακας 1:	Μέσο τετραγωνικό σφάλμα ανάλογα με την αναπαράσταση	29
Πίνακας 2:	Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 100x100	52
Πίνακας 3:	Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 150x150	54
Πίνακας 4:	Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 200x200	56
Πίνακας 5:	Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 330x330	58
Πίνακας 6:	Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 512x512	60
Πίνακας 7:	Χρόνοι φιλτραρίσματος ανά εικόνα	61
Πίνακας 8:	Χρήση πόρων για την υλοποίηση του φίλτρου με ενσωματωμένη μνήμη	62
Πίνακας 9:	Χρήση πόρων για την υλοποίηση φίλτρου χωρίς ενσωματωμένη μνήμη	63
Πίνακας 10:	Διαφορές μεταξύ των δύο σχεδιασμών του φίλτρου	64





Κατάλογος Γραφημάτων

Γράφημα 1: PSNR για εικόνα 100x100	51
Γράφημα 2: MSSIM για εικόνα 100x100.....	51
Γράφημα 3: PSNR για εικόνα 150x150	53
Γράφημα 4: MSSIM για εικόνα 150x150.....	53
Γράφημα 5: PSNR για εικόνα 200x200	55
Γράφημα 6: MSSIM για εικόνα 200x200.....	55
Γράφημα 7: PSNR για εικόνα 330x330	57
Γράφημα 8: MSSIM για εικόνα 33x330.....	57
Γράφημα 9: PSNR για εικόνα 512x512	59
Γράφημα 10: MSSIM για εικόνα 512x512	59
Γράφημα 11: PSNR ανά επανάληψη για θόρυβο σnoise=15	68
Γράφημα 12: MSSIM ανά επανάληψη για θόρυβο σnoise=15	68
Γράφημα 13: PSNR ανά επανάληψη για θόρυβο σnoise=35	69
Γράφημα 14: MSSIM ανά επανάληψη για θόρυβο σnoise=35	69





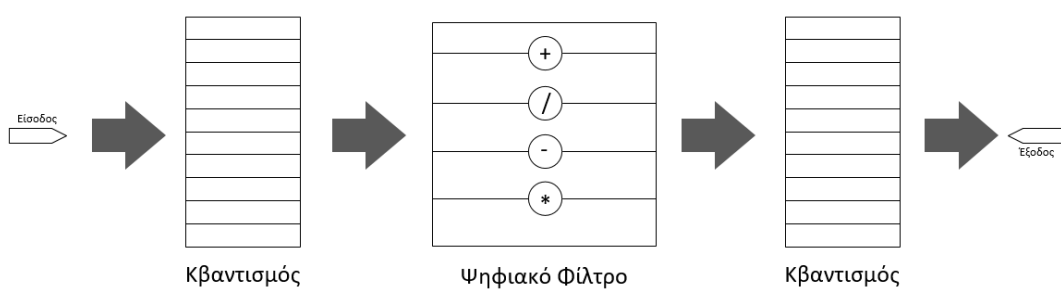
Κεφάλαιο 1

Εισαγωγή στα Ψηφιακά Φίλτρα

Κατά την επεξεργασία σημάτων, ένα φίλτρο είναι ένα σύστημα στο οποίο εκτελούνται μία σειρά από αριθμητικές πράξεις, ανάλογα με τους σκοπούς της επεξεργασίας. Είναι ευρέως γνωστό, ότι υπάρχουν δύο κατηγορίες σημάτων, **συνεχούς χρόνου** και **διακριτού**. Συνεπώς, τα υποσυστήματα που πραγματοποιούν την επεξεργασία χωρίζονται αντίστοιχα σε αυτές τις δύο κατηγορίες.

Δεδομένου ότι τα ψηφιακά συστήματα που σχεδιάζονται στο υλικό βασίζονται σε σύγχρονα ακολουθιακά κυκλώματα, το οποίο σημαίνει ότι η ταχύτητα αλλαγής μεταξύ μιας τωρινής και μιας επόμενης κατάστασης καθορίζεται από το ρολόι (clock), τα σήματά που μπορούμε να επεξεργαστούμε σε τέτοια συστήματα είναι διακριτού χρόνου. Ταυτόχρονα, πολύ σημαντικό ρόλο παίζει η ακρίβεια με την οποία αναπαριστούμε την πληροφορία στην σχεδίαση ενός ψηφιακού φίλτρου καθώς η ακρίβεια είναι ανάλογη του κόστους σε πόρους που θα έχει ο σχεδιασμός.

Επομένως, αντί να έχουμε απεριόριστη ακρίβεια στην αναπαράστασή μας, με χρήση του κβαντισμού, ορίζουμε πόσες διακριτές τιμές θέλουμε να έχει ο σχεδιασμός και οι τιμές εισόδου παίρνουν την τιμή της πλησιέστερης στάθμης κβάντισης. Το ίδιο συμβαίνει και με τις τιμές εξόδου. Συνεπώς, τα σήματά μας είναι διακριτού χρόνου και περιορισμένης ακρίβειας.



Εικόνα 1: Στάδια επεξεργασίας σήματος-πληροφορίας σε ένα ψηφιακό φίλτρο





1.1 Πεδία Χρήσης Ψηφιακών Φίλτρων

Τα ψηφιακά φίλτρα λόγω των αποτελεσμάτων που επιφέρουν έχουν βρει ευρεία χρήση σε πολλούς επιστημονικούς τομείς. Κάποιοι από αυτούς είναι :

Ιατρική

Με χρήση του ψευδοχρωματισμού, περιοχές με συγκεκριμένες ιδιότητες, τις οποίες το ανθρώπινο μάτι δεν μπορεί να διακρίνει, γίνονται πιο έντονες ούτως ώστε να κάνουν την διάγνωση πιο εύκολη. Το διμερές φίλτρο, με το οποίο θα ασχοληθούμε στην διπλωματική αυτή, έχει χρησιμοποιηθεί επιτυχημένα και σε τομογραφίες αφού με τη χρήση του απαιτούνται μικρότερης ισχύος ακτίνες Χ ώστε να επέλθει το επιθυμητό αποτέλεσμα καθώς αποθρομβοποιεί τις τελικές εικόνες.

NDT

Ο πλήρης όρος είναι nondestructive testing και αναφέρεται σε τεχνικές ανάλυσης που χρησιμοποιούνται για να προσδιορίσουν τις ιδιότητες ενός εξαρτήματος, υλικού ή συστήματος χωρίς να επέλθει η καταστροφή του ιδίου. Με την χρήση ενός ψηφιακού φίλτρου όπως το διμερές μπορεί να επιτευχθεί αυτό το αποτέλεσμα πιο γρήγορα κάνοντας την διαδικασία πιο αποδοτική καθώς η ανάλυση του αντικειμένου απαιτεί και μεγάλες ποσότητες ενέργειας αλλά και μεγάλο χρονικό διάστημα.

Συστήματα Video Surveillance

Στην αγορά της παρακολούθησης μέσω βίντεο, το στάνταρ είναι η χρήση ψηφιακών συστημάτων έναντι αναλογικών. Υπάρχει αυξανόμενη απαίτηση για χρήση υψηλότερης ανάλυσης και υψηλότερων frame rates τα οποία απαιτούν και τη χρήση των νέων τεχνικών συμπίεσης (πρότυπα H.264 και H.265), τα οποία απαιτούν μεγαλύτερη επεξεργαστική ισχύ. Το προαναφερθέν φαινόμενο ενισχύεται, από την εξέλιξη στις κάμερες ώστε να υποστηρίζουν WDR (Wide Dynamic Range).

Συνεπώς, κάποιες από τις τεχνικές που χρησιμοποιούνται είναι:

- Αφαίρεση καμένου pixel
- Αποθρομβοποίηση
- Tone-mapping
- Χρωματική Διόρθωση





1.2 Πως λειτουργεί ένα διμερές φίλτρο (Bilateral Filter)

Σε αυτό το κεφάλαιο θα εξηγήσουμε αναλυτικά, σε θεωρητικό επίπεδο και με κάποια διαγράμματα, όλα τα τμήματα που απαρτίζουν το διμερές φίλτρο και θα εξηγήσουμε τη χρησιμότητα του καθενός από αυτά.

1.3 Παράθυρο Φίλτρου (Kernel)

Το παράθυρο του φίλτρου καθορίζει τη γειτονιά την οποία παίρνουμε υπόψιν μας ούτως ώστε να καθορίσουμε την τελική τιμή του μεσαίου pixel μέσα από τις πράξεις που θα εκτελεστούν στο φίλτρο μας. Το μέγεθος του παραθύρου παίζει πολύ σημαντικό ρόλο διότι όσο μεγαλύτερο είναι αυξάνεται το φαινόμενο θολώματος (blurring effect), ενώ όσο πιο μικρό είναι καταφέρνουμε μικρότερη αποθορυβοποίηση. Επιπλέον, ένα μεγαλύτερο παράθυρο οδηγεί σε μεγαλύτερη εικόνα επεξεργασίας λόγω zero-padding, ούτως ώστε να μην έχουμε απώλεια στον τελικό αριθμό των pixels (θα αναφερθούμε σε αυτό σε επόμενο κεφάλαιο).

Όσον αφορά λοιπόν στο μέγεθος του παραθύρου, ορίζουμε τις διαστάσεις του ίσες με 5x5 διότι αυτές μας έδωσαν μία καλή ισορροπία μεταξύ θολώματος και αποθορυβοποίησης. Όλος ο σχεδιασμός, ο μέγιστος αριθμός πόρων που χρησιμοποιήθηκαν αλλά και οι κύκλοι ρολογιού που απαιτούνται για το τελικό αποτέλεσμα επηρεάζονται άμεσα από αυτή την επιλογή.

1.4 Σταθερές - Παράμετροι

Όσον αφορά το κομμάτι των μαθηματικών πράξεων, οι πράξεις μας περιλαμβάνουν κάποιες σταθερές οι οποίες είτε άμεσα, μέσω άμεσης εισόδου, είτε έμμεσα προσδιορίζονται.

σ_{noise}

Η μεταβλητή αυτή καθορίζει την ποσότητα του θορύβου που θα έχει η εικόνα για την οποία θα γίνεται το φιλτράρισμα. Ο θόρυβος που εισάγουμε είναι γκαουσιανός και περισσότερα αναφέρουμε στη διαδικασία εισαγωγής θορύβου στην εικόνα. Καθώς αυτό το σενάριο είναι το ιδανικό (γνωρίζουμε δηλαδή την ακριβή ποσότητα του θορύβου), στις πράξεις μας συμπεριλαμβάνουμε τον θόρυβο τον οποίο αρχικά έχουμε εισάγει.





Σε ένα πιο πραγματικό σενάριο θα μπορούσαμε να μην γνωρίζουμε την ένταση του θορύβου και για αυτόν τον λόγο προτείνουμε στο τέλος, σχετικά με πιθανή μελλοντική ενασχόληση για βελτίωση του σχεδιασμού, μία μέθοδο ώστε να αντιμετωπιστεί αυτό το πρόβλημα.

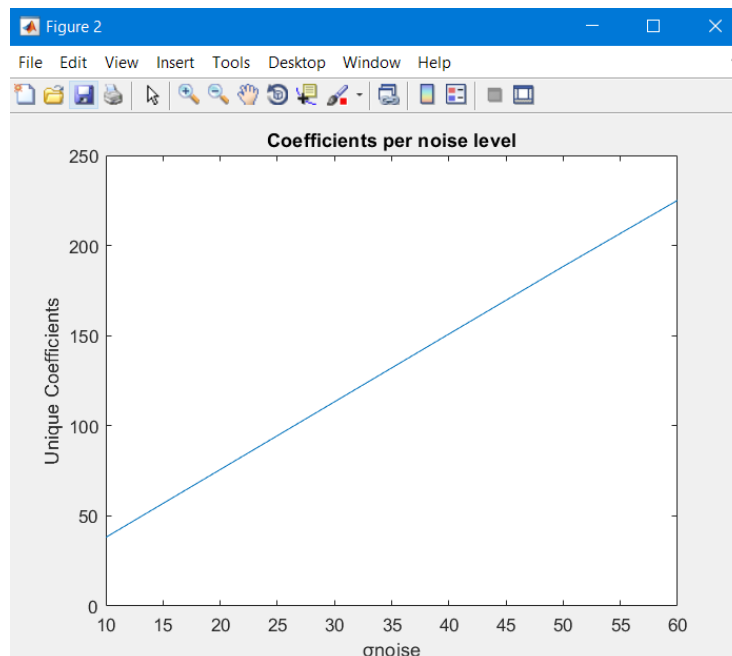
σ_{ph}

Η τιμή της παραμέτρου προκύπτει από την παράμετρο σ_{noise} και ο τύπος εύρεσης της τιμής της είναι:

$$\sigma_{ph} = 3 * \sigma_{noise}$$

Η συγκεκριμένη μεταβλητή επηρεάζει, ανάλογα με την τιμή της, τον μέγιστο αριθμό μοναδικών συντελεστών που θα περιέχονται στα look-up tables μας κατά το φωτομετρικό φιλτράρισμα.

Εικόνα 2: Μοναδικοί Συντελεστές ανάλογα με την παράμετρο σ_{noise}





σ_c

Πρόκειται για παράμετρο, η οποία μπορεί να καθοριστεί από είσοδο χρήστη και επηρεάζει την τελική τιμή της γεωμετρικής συνιστώσας. Με βάση την ακρίβεια που χρησιμοποιούμε για να αναπαραστήσουμε τις διακριτές τιμές στο υλικό, παρατηρήσαμε πως το εύρος τιμών της παραμέτρου είναι:

$$1 \leq \sigma_c \leq 32$$

Η παράμετρος, για μεγαλύτερες τιμές από το 32 προσφέρει παρόμοια αποτελέσματα άρα δεν υπάρχει λόγος για μεγαλύτερο εύρος τιμών. Παρ' όλα αυτά, σπανίως χρησιμοποιήθηκε με τιμή διαφορετική από $\sigma_c = 1$ καθώς η αύξησή της προκαλεί μεγαλύτερο θόλωμα. Δεδομένου ότι η κεντρική έγνοιά μας ήταν η σωστή λειτουργία του φίλτρου για την επίτευξη αποθορυβοποίησης χωρίς την απώλεια των άκρων της εικόνας (edge preserving), δεν δόθηκε μεγαλύτερη σημασία ούτε παρατηρήθηκε κάποιο σενάριο, στο οποίο η αλλαγή της τιμής της μεταβλητής σ_c να προσφέρει κάποια αξιοσημείωτη διαφορά.





Κεφάλαιο 2

Αριθμητικές Πράξεις

Παρακάτω θα παρουσιάσουμε, με μία θεωρητική μαθηματική ματιά, όλες τις μεταβλητές και τους τύπους του φίλτρου, καθώς και τη σχέση που έχουν μεταξύ τους. Σε πολλούς από τους τύπους περιλαμβάνονται και οι σταθερές που έχουμε αναφέρει παραπάνω.

2.1 Φωτομετρικό Φιλτράρισμα

Η φωτομετρική συνιστώσα συμβολίζεται ως $S(\varphi(m_0), \varphi(m))$ και είναι η πρώτη από τις δύο κύριες συνιστώσες που πρέπει να υπολογιστούν, καθώς τα αποτελέσματά τους μας χρειάζονται για τις υπόλοιπες πράξεις, με τελικό σκοπό τον υπολογισμό της νέας τιμής του υπό φιλτράρισμα κεντρικού pixel του παραθύρου (kernel). Η φωτομετρική συνιστώσα συγκρίνει την τιμή του κεντρικού pixel με τις τιμές των pixel της γειτονιάς του και υπολογίζει τον συντελεστή με βάση την παράμετρο σ_{ph} . Συνεπώς:

$$S(\varphi(m_0), \varphi(m)) = \exp\left(-\frac{1}{2}\left(\frac{\|\varphi(m_0) - \varphi(m)\|}{\sigma_{ph}}\right)^2\right)$$

$\varphi(m_0) \rightarrow$ η τιμή του μεσαίου pixel του παραθύρου

$\varphi(m) \rightarrow$ η τιμή ενός pixel της γειτονιάς του παραθύρου

2.2 Γεωμετρικό Φιλτράρισμα

Η γεωμετρική συνιστώσα συμβολίζεται ως $C(m_0, m)$ και είναι η δεύτερη συνιστώσα που πρέπει να υπολογιστεί ώστε να χρησιμοποιηθούν τα αποτελέσματά της, μαζί με τα αποτελέσματα της φωτομετρικής συνιστώσας, για την εκτέλεση των υπόλοιπων πράξεων που είναι απαραίτητες για τον υπολογισμό της νέας τιμής του μεσαίου, υπό φιλτράρισμα pixel, του παραθύρου (kernel). Η γεωμετρική συνιστώσα συγκρίνει την θέση ανάμεσα στο μεσαίο pixel και τα pixel της γειτονιάς του και υπολογίζει τον συντελεστή με βάση την παράμετρο σ_c . Επομένως:





$$C(m_0, m) = \exp\left(-\frac{1}{2}\left(\frac{\|m_0 - m\|}{\sigma_c}\right)^2\right)$$

$m_0 \rightarrow$ θέση του μεσαίου pixel με συντεταγμένες (m_0, n_0)

$m \rightarrow$ θέση ενός pixel της γειτονιάς με συντεταγμένες (m, n)

Καθώς ο αριθμός των θέσεων, είτε ανά γραμμή είτε ανά στήλη, στο kernel μπορεί να είναι από 1 έως 5 και η τιμή της θέσης του μεσαίου pixel είναι η τιμή 3 θα κάνουμε μία περαιτέρω ανάλυση ώστε να δούμε πόσους συντελεστές εν τέλει θα πρέπει να αποθηκεύσουμε στον πίνακα look-up της γεωμετρικής συνιστώσας του φίλτρου. Πιο συγκεκριμένα:

$$C(1, 3) = \exp\left(-\frac{1}{2}\left(\frac{\|1 - 3\|}{\sigma_c}\right)^2\right)$$

$$C(2, 3) = \exp\left(-\frac{1}{2}\left(\frac{\|2 - 3\|}{\sigma_c}\right)^2\right)$$

$$C(3, 3) = 1, \text{ καθώς προκύπτει } e^0 \text{ και άσχετα από } \sigma_c$$

Λόγω του ότι οι συγκεκριμένες τιμές πολλαπλασιάζονται με την φωτομετρική συνιστώσα και εφόσον ο πολλαπλασιασμός με το 1 είναι περιττός αγνοούμε την συνιστώσα $C(3, 3)$.

$$C(4, 3) = \exp\left(-\frac{1}{2}\left(\frac{\|4 - 3\|}{\sigma_c}\right)^2\right) = C(2, 3)$$

$$C(5, 3) = \exp\left(-\frac{1}{2}\left(\frac{\|1 - 3\|}{\sigma_c}\right)^2\right) = C(1, 3)$$

$$\text{Άρα } C(1, 3) = C(5, 3) = C_0 \text{ και } C(2, 3) = C(4, 3) = C_1$$

Συνεπώς, ανάλογα την θέση της κάθε τιμής στο παράθυρο, αυτή θα πολλαπλασιαστεί είτε με τη μεταβλητή C_0 είτε με τη C_1 .





2.3 Εξομάλυνση

Η εξομάλυνση (normalization) συμβολίζεται ως $K(m_0)$ και είναι η μία από τις δύο τιμές που χρειαζόμαστε για να υπολογίσουμε την τιμή του μεσαίου υπό φιλτράρισμα pixel του παραθύρου. Μπορούμε να την υπολογίσουμε αφού βρούμε το σύνολο των γεωμετρικών και φωτομετρικών συνιστωσών. Διαιρώντας με την εξομάλυνση εφαρμόζουμε ένα χαμηλοπερατό φίλτρο αφού μειώνουμε το μήκος του ακέραιου μέρους της εξόδου κατά 3 bit. Για να υπολογίσουμε την εξομάλυνση πρέπει να αθροίσουμε το γινόμενο των επιμέρους φωτομετρικών και γεωμετρικών συνιστωσών. Πιο συγκεκριμένα:

$$K(m_0) = \sum_{m \in F} S(\varphi(m_0), \varphi(m)) \cdot C(m_0, m)$$

$F \rightarrow$ η γειτονιά του μεσαίου pixel

$\varphi(m_0) \rightarrow$ η τιμή του μεσαίου pixel του παραθύρου

$\varphi(m) \rightarrow$ η τιμή ενός pixel της γειτονιάς του παραθύρου

$m_0 \rightarrow$ θέση του μεσαίου pixel με συντεταγμένες (m_0, n_0)

$m \rightarrow$ θέση ενός pixel της γειτονιάς με συντεταγμένες (m, n)

2.4 Άθροισμα

Το άθροισμα δεν περιλαμβάνεται στο paper που μελετήσαμε σαν έννοια, παρόλα αυτά εμείς το ορίζουμε ως μία μεταβλητή, ώστε να μπορούμε εύκολα και άμεσα να αναφερόμαστε στο σύνολο των υπόλοιπων πράξεων που δεν έχουν κάποιον συμβολισμό. Θα συμβολίζουμε αυτή τη μεταβλητή ως $Symsum(m_0)$ ώστε να ταυτίζεται με το όνομα της αντίστοιχης μεταβλητής που χρησιμοποιήσαμε στο MATLAB. Το άθροισμα υπολογίζεται αφού έχουμε υπολογίσει τις φωτομετρικές και γεωμετρικές συνιστώσες και είναι η δεύτερη απαραίτητη τιμή για τον υπολογισμό του μεσαίου υπό φιλτράρισμα pixel του παραθύρου. Άρα:

$$Symsum(m_0) = \sum_{m \in F} \varphi(m) \cdot S(\varphi(m_0), \varphi(m)) \cdot C(m_0, m)$$

$F \rightarrow$ η γειτονιά του μεσαίου pixel





$\varphi(m_0) \rightarrow$ η τιμή του μεσαίου *pixel* του παραθύρου

$\varphi(m) \rightarrow$ η τιμή ενός *pixel* της γειτονιάς του παραθύρου

$m_0 \rightarrow$ θέση του μεσαίου *pixel* με συντεταγμένες (m_0, n_0)

$m \rightarrow$ θέση ενός *pixel* της γειτονιάς με συντεταγμένες (m, n)

2.5 Υπολογισμός του υπό Φιλτράρισμα Pixel

Η τιμή του *pixel* μετά το φιλτράρισμα συμβολίζεται ως $\bar{\varphi}(\overline{m_0})$ και είναι η έξοδος του φίλτρου για ένα συγκεκριμένο μέγεθος παραθύρου που έχουμε επιλέξει, από το σύνολο των *pixels* της εκάστοτε εικόνας προς φιλτράρισμα. Για να υπολογίσουμε την τιμή του πρέπει να διαιρέσουμε το άθροισμα με την εξομάλυνση. Η ίδια διαδικασία γίνεται για κάθε γειτονιά μέχρι να υπολογιστεί η τιμή όλων των μεσαίων *pixels* κάθε παραθύρου. Πιο συγκεκριμένα:

$$\bar{\varphi}(\overline{m_0}) = \frac{1}{K(m_0)} \cdot \text{Symsum}(m_0)$$

$K(m_0) \rightarrow$ εξομάλυνση

$\text{Symsum}(m_0) \rightarrow$ άθροισμα

$m_0 \rightarrow$ θέση του μεσαίου *pixel* με συντεταγμένες (m_0, n_0)





Κεφάλαιο 3

Εξομοίωση φίλτρου σε περιβάλλον MATLAB

Σε αυτό το σημείο θα δείξουμε αναλυτικά το σενάριο που υλοποιήσαμε σε περιβάλλον MATLAB το οποίο εξομοιώνει την λειτουργία του Διμερούς φίλτρου. Θα προσπαθήσουμε να εξηγήσουμε όσο πιο αναλυτικά γίνεται τη λειτουργία του σεναρίου καθώς οι περισσότερες εντολές που χρησιμοποιούνται για τις μαθηματικές πράξεις απαιτούν γνώση του MATLAB και της μεθόδου Vectorization, η οποία είναι πολύ αποδοτική για πράξεις μεταξύ πινάκων.

3.1 Τρόπος εισαγωγής θορύβου στην εικόνα

Για να εξετάσουμε την σωστή λειτουργία του φίλτρου πρέπει αρχικά να προσθέσουμε θόρυβο στην αρχική εικόνα που επεξεργαζόμαστε. Ο θόρυβος που προσθέτουμε στις εικόνες μας είναι λευκός γκαουσιανός που αποτελεί το πιο ευρέως χρησιμοποιούμενο είδος θορύβου που εισάγεται σε εικόνες για τον έλεγχο της απόδοσης των φίλτρων αποθορυβοποίησης. Για να προσθέσουμε τον θόρυβο στην εικόνα αρχικά βρίσκουμε την τυπική απόκλιση της εικόνας και από αυτήν τον λόγο του σήματος προς τον θόρυβο (SNR). Στη συνέχεια, υπολογίζουμε τον λόγο σήματος προς θόρυβο με βάση την ένταση του θορύβου που θέλουμε να εισάγουμε στην εικόνα, ο οποίος καθορίζεται από την μεταβλητή σ_{noise} . Τέλος, για τη δημιουργία της εικόνας με τον θόρυβο χρησιμοποιούμε την συνάρτηση **awgn** η οποία λαμβάνει υπόψιν τον τρέχοντα σηματοθορυβικό λόγο και αυτόν που θέλουμε να έχει και κάνει την ανάλογη μετατροπή. Δεν υπάρχει περίπτωση να προκύψουν από 2 σενάρια ίδιες τελικές θορυβώδεις εικόνες.

```
function [noisy_im] = insert_noise(input_image,s_noise)

%Τυπική απόκλιση εικόνας, θορύβου
s_im = std2(input_image);

%Εισαγωγή θορύβου στην εικόνα
SNR_im=to_db(s_im);
SNR=to_db( s_im / s_noise );

%Τελική με θόρυβο
noisy_im=awgn(input_image, SNR, SNR_im);

end
```

Εικόνα 3: Κώδικας εισαγωγής θορύβου σε εικόνα



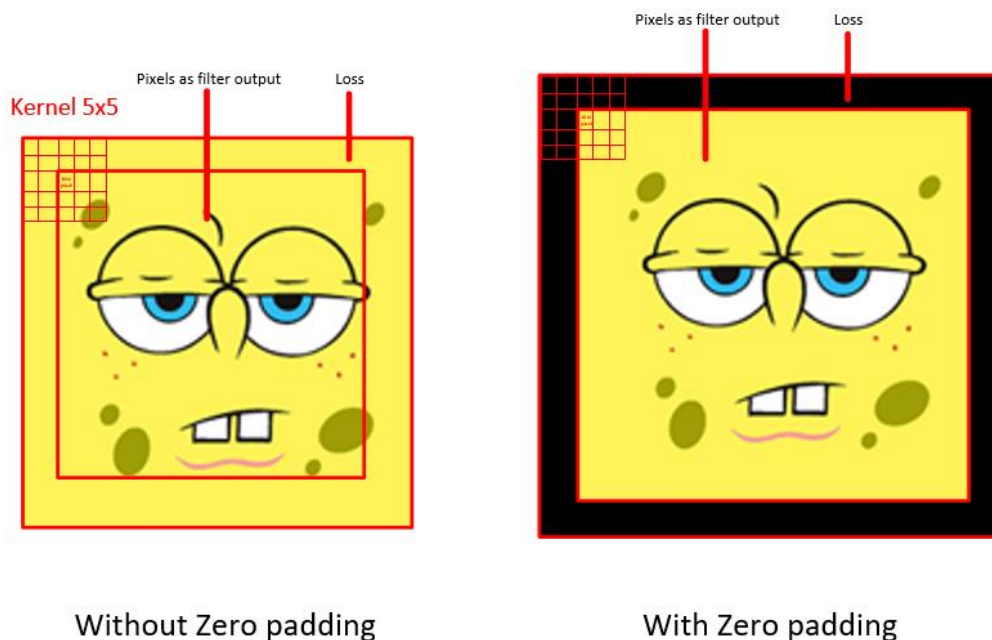


3.2 Μετατροπή εικόνας με θόρυβο σε αρχείο .mif

Τη διαδικασία αυτή, τη δημιουργήσαμε διότι το Modelsim μπορεί να αναγνωρίσει αρχεία για να εξομοιώσει RAM μόνο αν αυτά είναι σε μορφή .hex . Η δημιουργία ενός script ώστε να μετατρέπει μία εικόνα σε .hex ήταν αρκετά πολύπλοκη και χρονοβόρα και για αυτόν τον λόγο αποφασίσαμε να δημιουργήσουμε ένα script που θα μετατρέπει την εικόνα σε μορφή .mif, κάτι που ήταν πολύ πιο εύκολο και σχετικά γρήγορο, και στη συνέχεια, χρησιμοποιώντας το Quartus, μετατρέπουμε την εικόνα από μορφή .mif σε μορφή .hex από τις διαθέσιμες επιλογές του εργαλείου.

3.3 Χρήση μεθόδου zero-padding

Η μέθοδος zero-padding εφαρμόζεται πριν τη χρήση του Διμερούς Φίλτρου και ουσιαστικά δημιουργεί μία περιοχή με μαύρα pixels, δηλαδή pixels όπου η τιμή τους είναι 0, γύρω από την εικόνα μας. Το μέγεθος του zero padding είναι ανάλογο του μεγέθους του παραθύρου του φίλτρου. Ο λόγος που χρειαζόμαστε την μέθοδο αυτή είναι επειδή το φίλτρο μας χρησιμοποιεί μία γειτονιά, της οποίας το μέγεθος καθορίζεται από το kernel, για να υπολογίσει την τιμή του νέου φιλτραρισμένου pixel. Το pixel αυτό είναι το μεσαίο του παραθύρου. Δεδομένου του προαναφερθέντος γεγονότος, αν εφαρμόζαμε φιλτράρισμα, χωρίς zero-padding, θα χάναμε πληροφορία διότι δεν θα είχαμε φιλτράρει τα pixels στην περιφέρεια της εικόνας.



Εικόνα 4: Διαφορές μεταξύ χρήσης και μη χρήσης της μεθόδου Zero Padding





3.4 Δημιουργία του συνόλου των Kernel

Για να γίνει αποδοτικά η εξομοίωση του φίλτρου, αντί να δημιουργήσουμε μία διαδικασία η οποία για όλες τις γραμμές και στήλες εκτελεί τις πράξεις σε εμφωλευμένες επαναλήψεις, προτιμήσαμε να κρατάμε κάθε kernel από τα οποία, μέσω του φίλτρου, θα προκύπτει μία φιλτραρισμένη τιμή για το μεσσαίο pixel.

```
my_kernel=zeros(5,5,kern_nums);

w=1;
for i=3:1:(x-2)
    for j=3:1:(y-2)

        my_kernel(:, :,w) =[noisy_im(i-2,j-2) noisy_im(i-2,j-1) noisy_im(i-2,j) noisy_im(i-2,j+1) noisy_im(i-2,j+2);
                             noisy_im(i-1,j-2) noisy_im(i-1,j-1) noisy_im(i-1,j) noisy_im(i-1,j+1) noisy_im(i-1,j+2);
                             noisy_im(i,j-2)   noisy_im(i,j-1)   noisy_im(i,j)   noisy_im(i,j+1)   noisy_im(i,j+2);
                             noisy_im(i+1,j-2) noisy_im(i+1,j-1) noisy_im(i+1,j) noisy_im(i+1,j+1) noisy_im(i+1,j+2);
                             noisy_im(i+2,j-2) noisy_im(i+2,j-1) noisy_im(i+2,j) noisy_im(i+2,j+1) noisy_im(i+2,j+2)];

        w=w+1;
    end
end
```

Εικόνα 5: Κώδικας δημιουργίας των kernels

3.5 Υπολογισμός της Φωτομετρικής και Γεωμετρικής συνιστώσας

Έχοντας λοιπόν το σύνολο των kernels σαν πληροφορία αλλά και έναν πίνακα που αντιπροσωπεύει την θέση του κάθε pixel στο kernel, εκτελώντας πράξεις μεταξύ πινάκων μπορούμε πολύ αποδοτικά και γρήγορα να υπολογίσουμε τα αποτελέσματα των μαθηματικών πράξεων του `parer` και να υπολογίσουμε την φωτομετρική και γεωμετρική συνιστώσα, την εξομάλυνση, το άθροισμα και εν τέλει την τιμή του κάθε, υπό φιλτράρισμα, pixel.

Ο μηδενισμός της κεντρικής τιμής συμβαίνει διότι υπολογίζουμε μία παραπάνω τιμή για την γεωμετρική και φωτομετρική συνιστώσα, η οποία αφορά το ίδιο το μεσαίο pixel. Επειδή η μέθοδος είναι πολύ αποδοτική χρονικά, δεν μας απασχολεί το γεγονός ότι μετά πρέπει να μηδενίσουμε αυτή την κεντρική τιμή για κάθε συνιστώσα, ώστε εν τέλει να έχουμε μόνο τους 24 επιθυμητούς συντελεστές.





```
v=[1 1 1 1 1;
    2 2 2 2 2;
    3 3 3 3 3;
    4 4 4 4 4;
    5 5 5 5 5];

my_kernel_fixed= Fi_this1(my_kernel, 1, 20, 8);

%%FLOAT%% και %%FIXED%% Υπολογισμός του S και αφαίρεση του 25ου coefficient
for i=1:kern_nums

    S_float(:,i)=(exp( ( -(1/2) ) * ((abs(minus(my_kernel(3,3,i) , my_kernel(:,i)))/s_ph).^2) ));
    S_float(3,3,i)=0;

    C_float(:,i)=(exp( ( -(1/2) ) .* ((abs(minus(3,v)))/s_c).^2) ));
    C_float(3,3,i)=0;

end
```

Εικόνα 6: Κώδικας υπολογισμού Φωτομετρικών και Γεωμετρικών συνιστωσών

3.6 Υπολογισμός αθροίσματος, εξομάλυνσης και τελικής τιμής pixel

Αφού έχουμε υπολογίσει τη γεωμετρική και φωτομετρική συνιστώσα, με πράξεις μεταξύ πινάκων ίδιου μεγέθους, υπολογίζουμε τους πίνακες του αθροίσματος και της εξομάλυνσης και αμέσως μετά, με διαίρεση μεταξύ των δύο πινάκων, παίρνουμε την τελική τιμή εξόδου για κάθε φιλτραρισμένο pixel.

```
%Υπολογισμός για float της Εξομάλυνσης, του αθροίσματος και της
%τελικής τιμής του προς φιλτράρισμα pixel
```

```
K_float = sum(sum(S_float.*C_float));

Symsum_float = sum(sum(S_float.*C_float.*my_kernel));

result_float = Symsum_float./K_float;
```

Εικόνα 7: Κώδικας υπολογισμού εξομάλυνσης, αθροίσματος και τελικής τιμής pixel





3.7 Αναδιάταξη Φιλτραρισμένης Εικόνας-MATLAB

Αφού έχουμε στο σύνολό τους όλες τις τιμές εξόδου του φίλτρου, πρέπει να αναδιατάξουμε την εικόνα ως ένα δισδιάστατο πίνακα ώστε να μπορούμε να την εκτιμήσουμε σε σχέση με την αρχική αλλά και με την εικόνα με θόρυβο. Παρακάτω δείχνουμε τον κώδικα για την συγκεκριμένη ενέργεια. Οι μεταβλητές x και y διαχειρίζονται με τρόπο ώστε η τελική εικόνα να έχει το αρχικό της μέγεθος και όχι αυτό με το zero-padding.

```
%Κατασκευή εικόνας από την έξοδο του φίλτρου float και fixed

w=1;
for i=3:(x-2)
    for j=3:(y-2)

        new_image_float(i-2,j-2)=result_float(1,1,w);
        new_image_fixed(i-2,j-2)=result_fixed(1,1,w);
        w=w+1;

    end
end
```

Εικόνα 8: Κώδικας MATLAB αναδιάταξης φιλτραρισμένης εικόνας





Κεφάλαιο 4

Υλοποίηση του φίλτρου στο DSP Builder

Στο συγκεκριμένο κεφάλαιο, θα παρουσιάσουμε όλα τα βήματα τα οποία ακολουθήσαμε για να σχεδιάσουμε το φίλτρο στο υλικό. Η σχεδίαση του φίλτρου χρειάστηκε τόσο κάποιες αποφάσεις πριν την σχεδίαση όσο και κατά την διάρκεια της σχεδίασης των επιμέρους υποσυστημάτων που το απαρτίζουν. Για την σχεδίαση και επαλήθευση της σωστής λειτουργίας του φίλτρου χρησιμοποιήσαμε το περιβάλλον Simulink του εργαλείου MATLAB, σε συνδυασμό με την έκδοση 15.0 της βιβλιοθήκης DSP Builder της Altera (νυν Intel), η οποία είναι ελεύθερη για χρήση και εξομοιώσεις στο περιβάλλον Simulink του MATLAB.

4.1 Κριτήρια επιλογής μεγέθους αναπαράστασης των pixels στο υλικό

Η επιλογή του κατάλληλου μεγέθους αναπαράστασης είναι ένα πολύ σημαντικό κομμάτι της συνολικής υλοποίησης. Η χρήση μικρότερου μεγέθους αναπαράστασης μπορεί να μην οδηγήσει στα αποτελέσματα που επιθυμούμε ή έχουμε προβλέψει στα σενάρια μας στο MATLAB. Από την άλλη η χρήση μεγαλύτερου μεγέθους, χρησιμοποιεί περισσότερους πόρους από όσους πραγματικά χρειαζόμαστε. Για να έχουμε μία άποψη συγκρίναμε το μέσο σφάλμα ανάμεσα στη φιλτραρισμένη εικόνα, με χρήση 5 διαφορετικών αναπαραστάσεων περιορισμένης ακρίβειας σταθερής υποδιαστολής (fixed-point), και στην εκδοχή της ίδιας φιλτραρισμένης εικόνας με χρήση ακρίβειας κινητής υποδιαστολής (float). Η σύγκριση επαναλήφθηκε για 6 διαφορετικές ποσότητες θορύβου. Η αναπαράσταση fixed-point θεωρεί υποδιαστολή από κάποιο bit και μετά.

Την ακρίβεια fixed-point, με βάση τις συναρτήσεις που χρησιμοποιήσαμε στο MATLAB, την συμβολίζουμε ως εξής:

$$fi(a, b, c)$$

$a \rightarrow$ ένδειξη προσημασμένης αναπαράστασης με εύρος τιμών [0,1]

$b \rightarrow$ ένδειξη ως προς τον συνολικό αριθμό των bit της αναπαράστασης

$c \rightarrow$ πόσα bit της αναπαράστασης b συνολικά είναι μετά την υποδιαστολή





Πίνακας 1: Μέσο τετραγωνικό σφάλμα ανάλογα με την αναπαράσταση

Μέσο MSE	fi (1,27,13)	fi (1,25,11)	fi (1,22,8)	fi (1,20,6)	fi (1,18,4)
$\sigma_{noise} = 10$	0,0836	0,2193	0,7774	1,8706	5,48
$\sigma_{noise} = 20$	5,3e-06	8,3e-05	0,0045	0,0632	1,008
$\sigma_{noise} = 30$	3,3e-06	5,08e-05	0,0033	0,0479	0,8416
$\sigma_{noise} = 40$	3,38e-06	4,94e-05	0,0033	0,0478	0,8181
$\sigma_{noise} = 50$	3,59e-06	5,06e-05	0,0035	0,0506	0,8368
$\sigma_{noise} = 60$	3,88e-06	5,28e-05	0,0039	0,0550	0,8722

Όσο αυξάνεται η μεταβλητή σ_{noise} τόσο αυξάνονται οι αρνητικοί αριθμοί καθώς και οι τιμές τους απομακρύνονται περισσότερο από το 0. Παρομοίως, προκύπτουν θετικοί αριθμοί που η τιμή τους είναι πολύ μεγαλύτερη από την μέγιστη ενός pixel, δηλαδή από την τιμή 255. Αυτή η λεπτομέρεια επηρεάζει αρκετά τη διαδικασία εύρεσης του μέσου τετραγωνικού σφάλματος MSE και για αυτό τον λόγο βλέπουμε τις μεγάλες διαφορές ανάμεσα στα μέσα σφάλματα για τις πιο μικρές τιμές του σ_{noise} σε σχέση με τις μεγαλύτερες τιμές του. Εν τέλει, η πιο ισορροπημένη αναπαράσταση είναι η μεσαία, δηλαδή η **fi(1,22,8)**. Κατά τη διάρκεια του σχεδιασμού μας παρατηρήσαμε πως το ακέραιο μέρος για την αναπαράσταση των pixels της φωτογραφίας με θόρυβο αλλά και η εξομάλυνση μπορούν να αναπαρασταθούν με ακρίβεια **fi(1,20,8)**. Η φωτομετρική συνιστώσα και το άθροισμα **Symsum** από την άλλη απαιτούν την αναπαράσταση **fi(1,22,8)** για όλο το εύρος των διαφορετικών σεναρίων θορύβου.

4.2 Χρήση Μεταβλητών στο DSP Builder από το MATLAB

Για να λειτουργήσει σωστά η εξομοίωση του φίλτρου πρέπει, πριν τρέξουμε την εξομοίωση στο Simulink, να τρέξουμε το **look_up.m** αρχείο. Έτσι, αποθηκεύουμε την εικόνα που προσθέτουμε θόρυβο στο αρχείο **image_with_noise.mat**, αν αυτό δεν υπάρχει, και μπορούμε να επαναλάβουμε την εξομοίωση για την ίδια ενθόρυβη εικόνα. Επιπλέον, με το τρέξιμο του αρχείου **look-up** αρχικοποιούνται:

- Οι διαστάσεις και η ροή της εικόνας με θόρυβο
- Οι πίνακες **look-up**
- Η σταθερά σ_c
- Οι φωτομετρικές και γεωμετρικές συνιστώσες





4.3 Γενική Ανάλυση λειτουργίας του φίλτρου στο υλικό

Παρακάτω παρουσιάζουμε τα επιμέρους υποσυστήματα του φίλτρου μας δίνοντας μία γενική περιγραφή της λειτουργίας τους:

Register Matrix

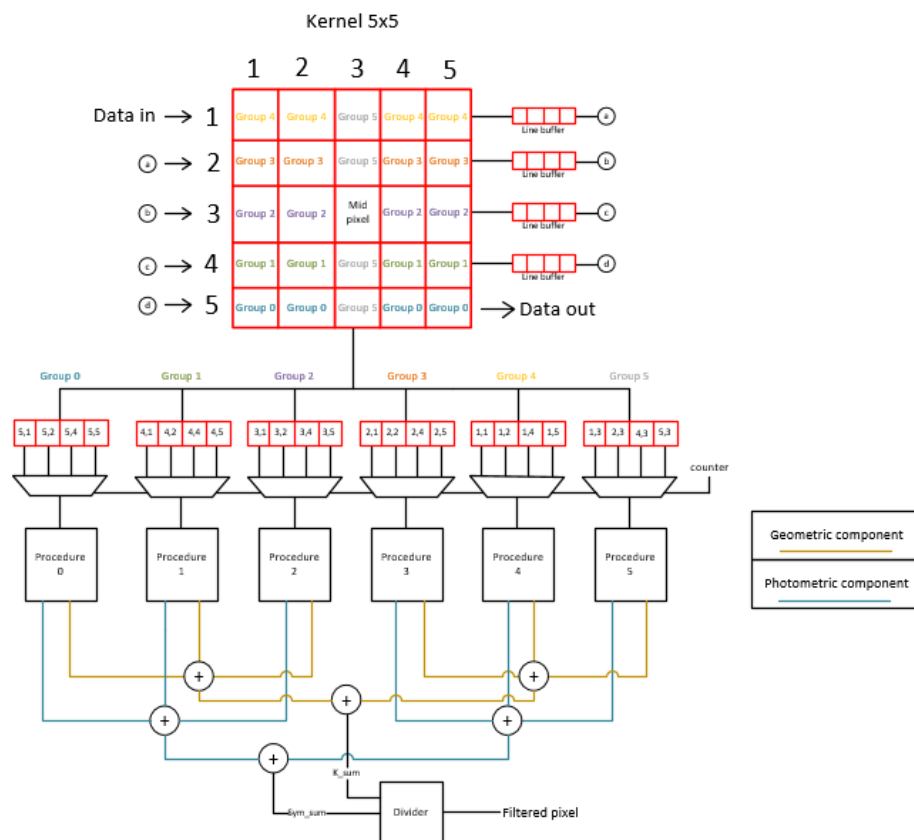
Εντός του παραθύρου του φίλτρου μπορούμε να παρατηρήσουμε τις ομάδες (groups_#) στις οποίες χωρίζονται οι τιμές που θα περάσουν στα επιμέρους procedures.

Procedures

Στα 6 αυτά υποσυστήματα υπολογίζονται 4 φωτομετρικές και γεωμετρικές συνιστώσες ανά υποσύστημα και τα αθροίσματα οδηγούνται στο υποσύστημα της διαίρεσης.

Divider

Το άθροισμα Symsum και η εξομάλυνση K οδηγούνται σε αυτό το υποσύστημα και το αποτέλεσμα της διαίρεσής τους είναι η έξοδος του φίλτρου.



Εικόνα 9: Σχέδιο Λειτουργίας Διμερούς φίλτρου

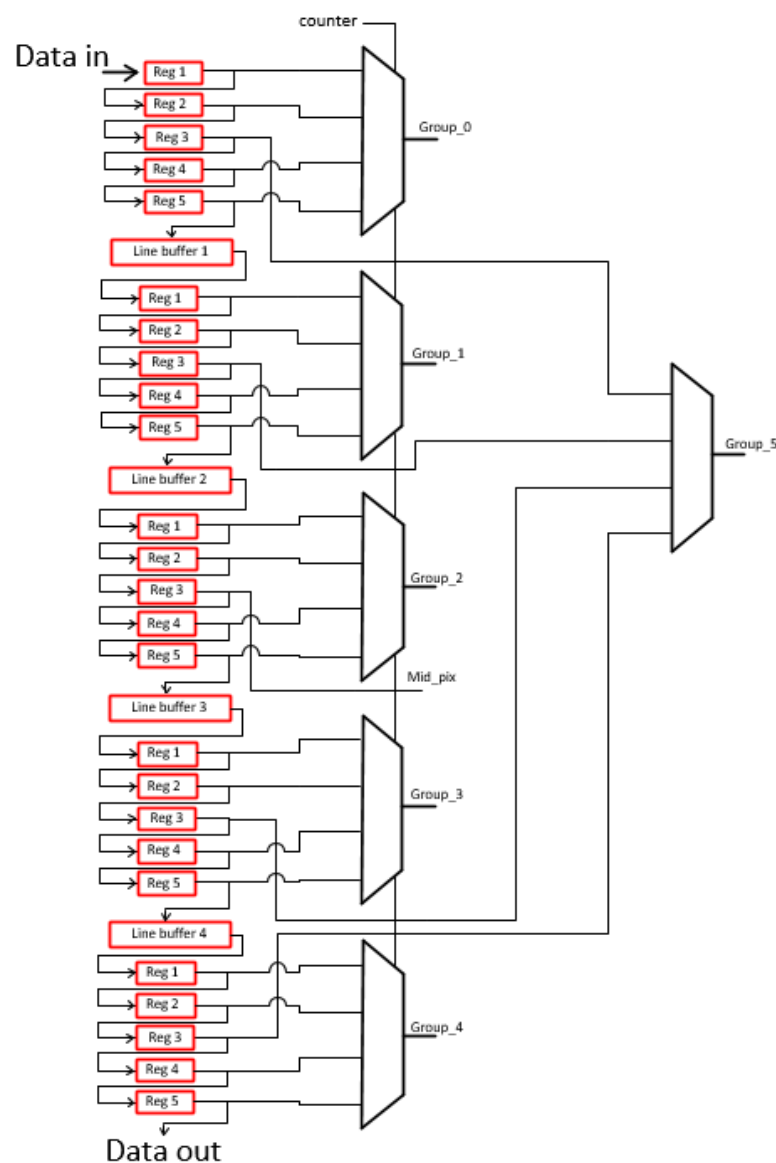




4.4 Σχεδίαση του Register Matrix

Το υποσύστημα Register Matrix περιλαμβάνει μικρές ομάδες από καταχωρητές (Registers), ανάλογα τις διαστάσεις του παραθύρου του φίλτρου, οι οποίοι έχουν ενδιάμεσά τους μεγαλύτερες ομάδες από καταχωρητές (Line Buffers), των οποίων το μέγεθος εξαρτάται από τον αριθμό των pixels ανά γραμμή της εκάστοτε εικόνας.

Οι Line Buffers φροντίζουν να καθυστερούν τις τιμές αρκετά ώστε οι ομάδες καταχωρητών να περιλαμβάνουν κάθε χρονική στιγμή τις τιμές των pixel μίας γειτονιάς μεγέθους 5x5 της εικόνας και, κατά συνέπεια, να προκύπτουν στα Procedures οι σωστές τιμές φιλτραρίσματος.



Εικόνα 10: Register Matrix



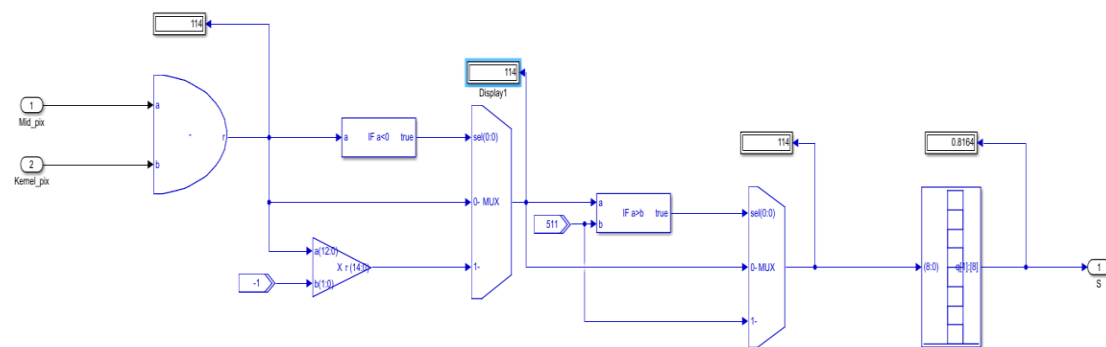


4.5 Σχεδίαση της Φωτομετρικής Συνιστώσας

Το υποσύστημα της φωτομετρικής συνιστώσας περιλαμβάνεται σε κάθε procedure που απαρτίζει το φίλτρο μας.

Για τον υπολογισμό της εκάστοτε φωτομετρικής συνιστώσας στο υλικό, αρχικά υπολογίζουμε την διαφορά της τιμής του μεσαίου pixel από την τιμή της γειτονιάς. Καθώς ο μαθηματικός τύπος χρησιμοποιεί την απόλυτη τιμή του αποτελέσματος, με τη χρήση ενός μπλοκ If, στην περίπτωση που το αποτέλεσμα είναι αρνητικό θα πολλαπλασιαστεί με το -1 ώστε να το μετατρέψουμε σε θετικό.

Έπειτα, ελέγχουμε αν το αποτέλεσμα της αφαίρεσης είναι μεγαλύτερο του 511. Αυτό γίνεται διότι έχουμε αρχικοποιήσει ένα πίνακα look-up ο οποίος έχει 512 προϋπολογισμένες φωτομετρικές συνιστώσες για αποτέλεσμα αφαίρεσης από 0 έως 511 και για συγκεκριμένο σ_{ph} άρα και σ_{noise} . Στην περίπτωση λοιπόν που το αποτέλεσμα είναι μεγαλύτερο από το προαναφερόμενο νούμερο θα του δώσουμε την τιμή της συνιστώσας της θέσης 512 του πίνακα look-up. Σε περίπτωση αριθμού με δεκαδικά ψηφία, θα επιλεγεί στον look-up πίνακα η θέση που αντιστοιχεί στο ακέραιο μέρος του αποτελέσματος.



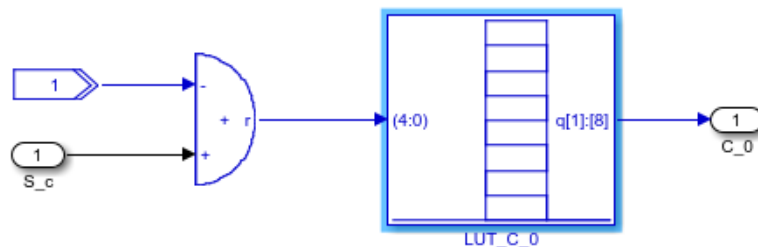
Εικόνα 11: Δομή Υποσυστήματος Υπολογισμού Γεωμετρικής Συνιστώσας





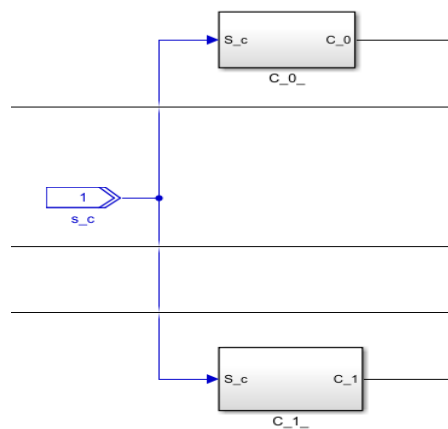
4.6 Σχεδίαση της Γεωμετρικής Συνιστώσας

Ο σχεδιασμός της γεωμετρικής συνιστώσας είναι συγκριτικά ο πιο απλός από όλα τα υποσυστήματα που απαρτίζουν το φίλτρο. Ο πίνακας look-up είναι γεμάτος με τις προϋπολογισμένες τιμές, τοποθετημένες στις θέσεις 0-31, ενώ ανάλογα με την επιλογή της συνιστώσας σ_c επιλέγεται η τιμή στη θέση $\sigma_c - 1$, η οποία ουσιαστικά είναι το αποτέλεσμα της γεωμετρικής συνιστώσας για την ανάλογη τιμή της σ_c .



Εικόνα 12: Πίνακας Look-up για επιλογή τιμής της μεταβλητής C_0 με βάση σ_c

Όπως αναφέραμε και στην ανάλυση των μαθηματικών τύπων της γεωμετρικής συνιστώσας, οι τιμές που πρέπει να υπολογίσουμε και μας χρειάζονται για τις μετέπειτα πράξεις μας είναι δύο, η C_0 και η C_1 . Ο τρόπος εύρεσης τους είναι ίδιος.



Εικόνα 13: Τα υποσυστήματα C_0 και C_1



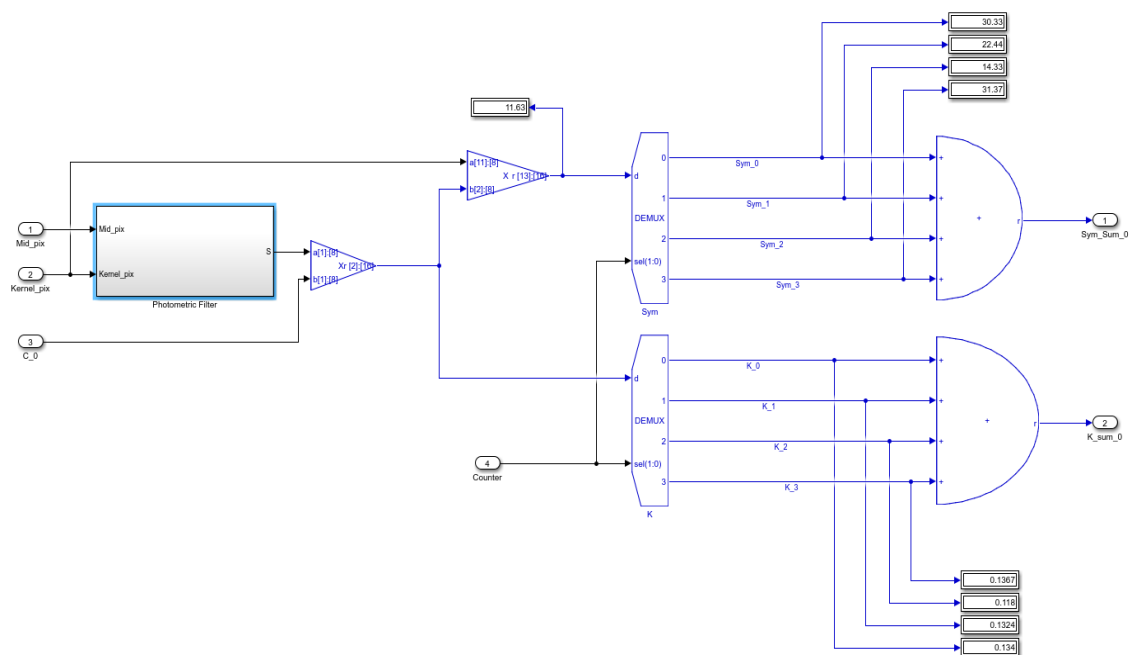


4.7 Σχεδίαση του Procedure

Το κάθε procedure δέχεται σαν είσοδο το μεσσαίο pixel του παραθύρου, μία από τις τιμές της ομάδας, ανάλογα την ομάδα που είναι χωρισμένα τα υπόλοιπα pixel του παραθύρου, την τιμή του counter, καθώς και την κατάλληλη γεωμετρική συνιστώσα. Το μεσσαίο pixel και η τιμή της ομάδας θα είναι οι είσοδοι για το υποσύστημα εύρεσης της φωτομετρικής συνιστώσας, το αποτέλεσμα του οποίου θα πολλαπλασιαστεί με την γεωμετρική συνιστώσα.

Στη συνέχεια, το αποτέλεσμα του πρώτου πολλαπλασιασμού, το οποίο είναι η εξομάλυνση **K** για το συγκεκριμένο pixel της γειτονιάς, θα οδηγηθεί σε έναν δεύτερο πολλαπλασιασμό, αυτή τη φορά με την τιμή του pixel της γειτονιάς και θα μας δώσει τη συνιστώσα του αθροίσματος **Symsum**.

Τόσο η συνιστώσα της εξομάλυνσης όσο και του αθροίσματος θα οδηγηθούν σε ξεχωριστούς αποπλέκτες, οι οποίοι θα τις συνδέσουν σε διαφορετικό wire που οδηγείται στην ανάλογη είσοδο του αθροιστή τους. Οι έξοδοι των δύο αθροιστών θα είναι και οι έξοδοι του κάθε procedure. Τέλος, τα επιμέρους αποτελέσματα των procedures θα αθροιστούν και θα περάσουν στο block της διαίρεσης.



Εικόνα 14: Εσωτερικό ενός εκ των έξι Procedures

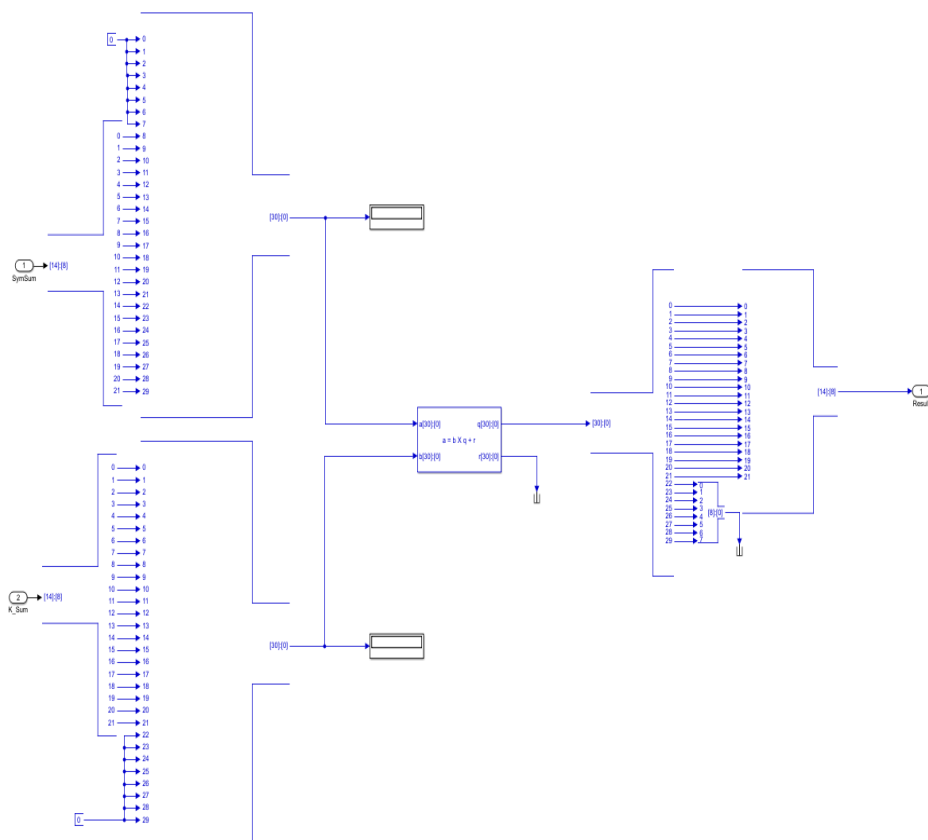




4.8 Σχεδίαση της Διαίρεσης

Εφόσον έχουμε υπολογίσει την εξομάλυνση K και το άθροισμα **Symsum**, για να βρούμε την τιμή του μεσαίου ρίxel θα πρέπει να διαιρέσουμε το άθροισμα με την εξομάλυνση. Για να γίνει σωστά η διαίρεση, πρέπει να τοποθετηθούν στον αριθμητή μηδενικά από το λιγότερο σημαντικό bit και προς τα δεξιά ενώ στον παρονομαστή από το πιο σημαντικό bit και προς τα αριστερά. Ο αριθμός των bit που πρέπει να προστεθούν ισούται με το πλήθος των ψηφίων μετά την υποδιαστολή που θεωρούμε.

Τέλος, μετά τον υπολογισμό της διαίρεσης κρατάμε τα λιγότερο σημαντικά ψηφία του αποτελέσματος (δεξιότερα) και απορρίπτουμε τα περισσότερα σημαντικά ψηφία τα οποία πάλι πρέπει να ισούται σε πλήθος με τα ψηφία μετά την υποδιαστολή. Εδώ αυτό σημαίνει ότι απορρίπτουμε 8 bit.



Εικόνα 15: Υποσύστημα Διαίρεσης





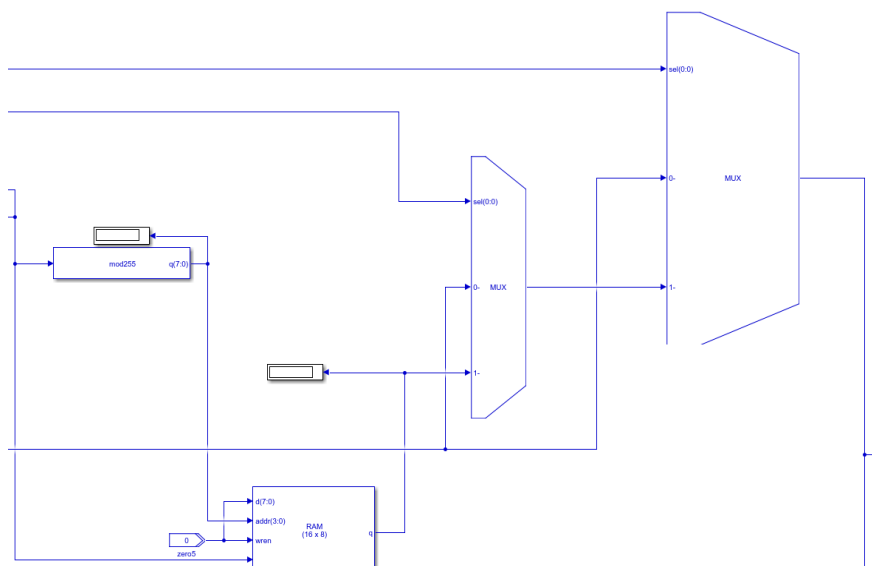
Κεφάλαιο 5

Σύνθεση στο Quartus και Εξομοίωση φίλτρου στο Modelsim

Μετά την σχεδίαση του φίλτρου μας στο Simulink και αφού βεβαιωθήκαμε ότι δουλεύει σωστά μέσα από τις εξομοιώσεις και την τοποθέτηση των φιλτραρισμένων pixels στις κατάλληλες θέσεις της τελικής εικόνας, το επόμενο βήμα ήταν να περάσουμε το φίλτρο μας από σύνθεση στο Quartus και κατόπιν, αφού γράψουμε το Modelsim Testbench σε Verilog, να επαναλάβουμε τη προαναφερόμενη διαδικασία επιβεβαίωσης της ορθής λειτουργίας του. Σημειώνουμε ότι για την μεταφορά του φίλτρου από το Simulink στο Quartus χρειάστηκε να λάβουμε μία προσωρινή άδεια (trial) του DSP Builder.

5.1 Φίλτρο με ενσωματωμένη μνήμη

Έχοντας αποθηκεύσει την εικόνα προς φιλτράρισμα ως μονοδιάστατο πίνακα στο workspace του MATLAB, μπορούμε να την περάσουμε σε RAM block του DSP Builder. Η συγκεκριμένη μνήμη χρησιμοποιείται είτε για διάβασμα είτε για γράψιμο. Όταν θέλουμε να διαβάσουμε, μας χρειάζονται ως είσοδοι η διεύθυνση μνήμης και η μεταβλητή **Write Enable** = 0. Εάν θέλουμε να γράψουμε σε κάποια διεύθυνση μας ενδιαφέρει να είναι η μεταβλητή **Write Enable** = 1 και επιπλέον πρέπει να δώσουμε μία τιμή που θα αντικαταστήσει την παλιά στην διεύθυνση που μας ενδιαφέρει.



Εικόνα 16: Μνήμη RAM και γειτονικά blocks από τη Standard βιβλιοθήκη





5.1.1 Testbench Φίλτρου

Η δημιουργία του Testbench για το σενάριο με την ενσωματωμένη μνήμη είναι απλό. Αρχικά, μας ενδιαφέρει να βρούμε τον συνολικό αριθμό από εισόδους και εξόδους του top-level αρχείου μας, οι οποίες δεν είναι άλλες από τις Clock, Reset και την έξοδο, καθώς στο φίλτρο, κατά το compilation, ενσωματώθηκε σε μία RAM η εικόνα που του δώσαμε στην εξομοίωση, ενώ όλες οι παράμετροι του φίλτρου δίνονται ως σταθερές.

Η περίοδος του ρολογιού μας είναι 20ns ή αλλιώς η συχνότητα είναι 50MHz. Άρα, για να κάνουμε σωστούς υπολογισμούς, πρέπει να ορίσουμε την καθυστέρηση αλλαγής του ρολογιού στα 10ns. Εφόσον έχουμε ορίσει σε κάθε θετική ακμή του Clock να αποθηκεύεται η έξοδος σε ένα αρχείο, αρχικοποιούμε τις εισόδους μας και ορίζουμε ως στιγμή λήξης της εξομοίωσης, τη χρονική στιγμή που αντιστοιχεί στο πλήθος των βημάτων που εκτελέσαμε στο Simulink επί τα 20ns.

```
1  `timescale 1 ns / 1 ns
2
3  module Filter_Testbench;
4
5
6  integer file;
7
8  reg    Clock, Reset;
9  wire  signed [21:0] Result;
10
11  Bilateral_zero_padded T1(Clock,Result, Reset);
12
13
14  always #10 Clock <= Clock + 1;
15
16  always @(posedge Clock) begin
17
18      $fdisplay(file,"%d", Result);
19
20  end
21
22  initial begin
23
24      Clock    <= 0;
25      Reset    <= 0;
26
27      file = $fopen("C:/Users/CHRIS/Desktop/thesis/system_tasks/Filter_Results_Mdlsim.txt"
28
29  #1240000 begin
30
31      $fclose(file);
32      $finish;
33
34  end
35  end
36  endmodule
```

Εικόνα 17: Κώδικας Verilog Testbench για Φίλτρο με ενσωματωμένη Μνήμη

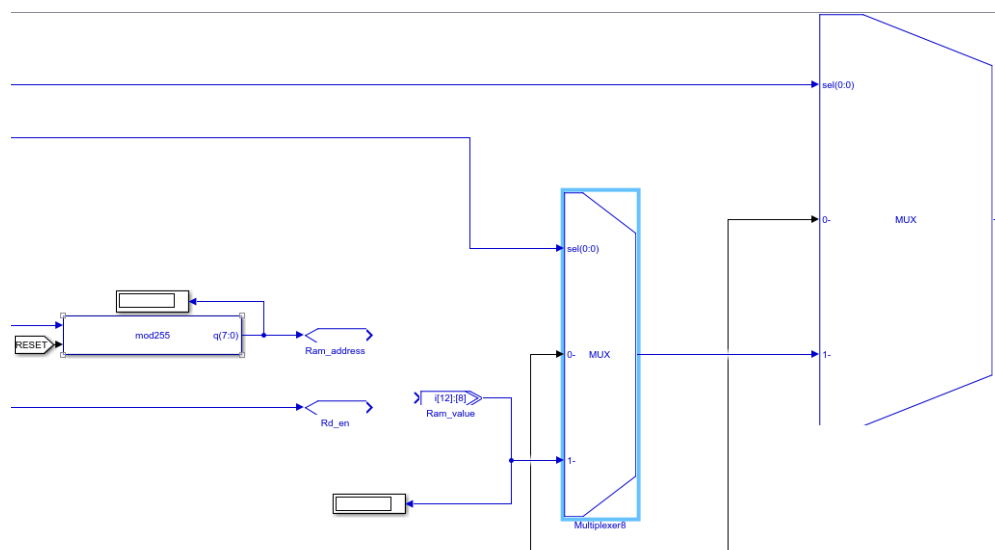




5.2 Φίλτρο χωρίς ενσωματωμένη μνήμη

Από την άλλη, έχουμε το σενάριο στο οποίο δεν χρησιμοποιούμε κάποια μνήμη, γεγονός που μειώνει αρκετά τους πόρους που θα πρέπει να αξιοποιήσει το φίλτρο από το FPGA. Τοποθετούμε συγκεκριμένα input/output blocks, ώστε να φανούν στη σύνθεση ως επιπρόσθετες εισόδους/ εξόδοι και στο Testbench να μπορούμε να συνδέσουμε τη μνήμη που θα έχουμε κάνει σύνθεση.

Δίνουμε μεγάλη προσοχή στην ακρίβεια που θα ορίσουμε αυτές τις εισόδους και εξόδους διότι οι εξόδοι του φίλτρου αλλά και εισόδοι στη RAM, ορίζουν το μέγιστο μέγεθος ανάλυσης εικόνας που μπορούμε να φιλτράρουμε, ενώ η είσοδος του φίλτρου που τροφοδοτείται από την έξοδο της RAM, ορίζει την ακρίβεια της πληροφορίας που θα περάσει στο παράθυρο του φίλτρου.



Εικόνα 18: Τοποθετώντας καταχωρητές εισόδου και εξόδου στη θέση της μνήμης RAM

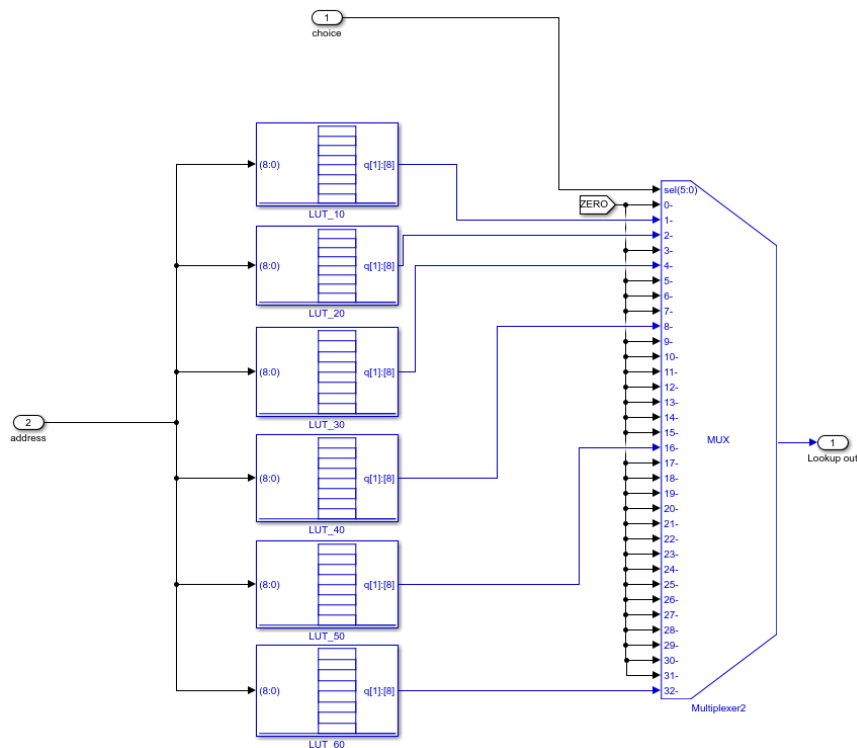




5.2.1 Επιλογή πίνακα συνιστωσών βάση θορύβου

Στην συγκεκριμένη εκδοχή του φίλτρου η σ_{ph} οδηγείται από είσοδο και δεν είναι προκαθορισμένη πριν την σύνθεση. Κατά συνέπεια, ήταν αναγκαία η προσθήκη πινάκων look-up ώστε ανάλογα με την τιμή της σταθεράς, να αναζητείται στον ανάλογο πίνακα η αντίστοιχη φωτομετρική συνιστώσα.

Η συγκεκριμένη ενέργεια προσφέρει περισσότερη ευελιξία καθώς μπορούμε να φιλτράρουμε εικόνες με διαφορετικό σ_{noise} αλλά με ίδια ανάλυση χωρίς να χρειάζεται να αλλάξουμε τον σχεδιασμό μας και να τον ξαναπεράσουμε από σύνθεση. Όμως, κάτι τέτοιο αυξάνει τις απαιτήσεις σε πόρους της υλοποίησης αφού προσθέτουμε look-up πίνακες εσωτερικά όλων των procedures, για κάθε διαφορετική εκδοχή προσθήκης θορύβου στην εικόνα.



Εικόνα 19: Επιλογή Πίνακα Look-up με βάση τον ορισμένο θόρυβο στο Testbench



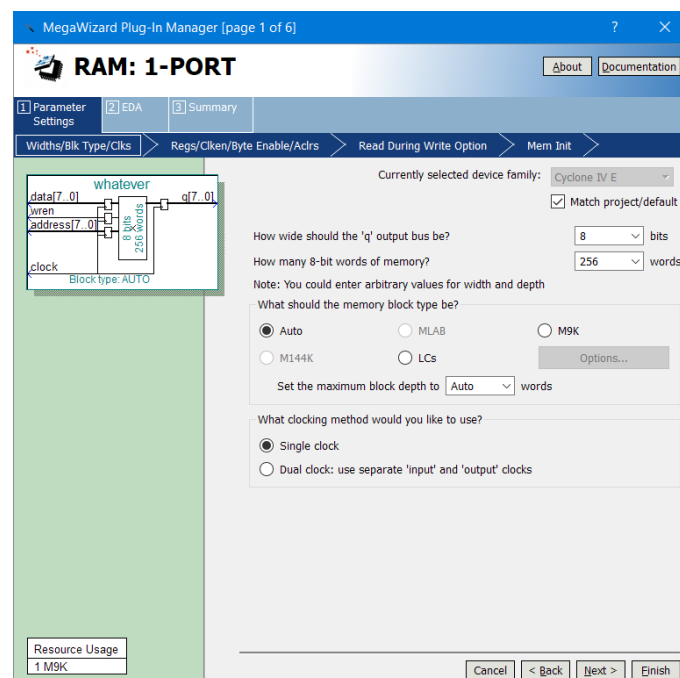


5.2.2 Δημιουργία μνήμης RAM μέσω του Quartus

Η δημιουργία της μνήμης RAM μπορεί εύκολα να γίνει ακολουθώντας τα σχετικά βήματα του εργαλείου Quartus. Πιο συγκεκριμένα, αν κάποιος κοιτάξει στον IP Catalog -> Library -> Basic Functions -> On Chip Memory -> RAM: 1-Port θα μπορέσει να ορίσει μία μνήμη όπως το block RAM που χρησιμοποιήσαμε στο σενάριο με την ενσωματωμένη μνήμη.

Εφόσον έχουμε ανοίξει τον MegaWizard, μπορούμε να ορίσουμε το μέγεθος της μνήμης και τον αριθμό των λέξεων που θα έχει. Αυτό το κομμάτι είναι το πιο σημαντικό καθώς θα πρέπει να ξέρουμε, όταν θα τη δημιουργούμε, τι μεγέθους εικόνα θέλουμε να χρησιμοποιήσουμε. Παράλληλα, μπορούμε να ορίσουμε τους πόρους του FPGA ολοκληρωμένου που θα χρησιμοποιήσει, ένα ή περισσότερα ρολόγια καθώς και κάποιες επιπρόσθετες εισόδους.

Για να φορτώσουμε μία εικόνα στη RAM θα πρέπει να την έχουμε μετατρέψει σε μορφή αρχείου .mif ή .hex ώστε το Quartus να μπορεί να τη διαβάσει. Κατά την ολοκλήρωση της όλης διαδικασίας, κάνουμε Compile το project ως σύνολο και τρέχουμε την προσομοίωση (RTL Simulation).



Εικόνα 20: Δημιουργία μνήμης RAM στο Quartus





5.2.3 Testbench Φίλτρου

Το Testbench για το σενάριο χωρίς την ενσωματωμένη μνήμη είναι ελαφρώς διαφορετικό από το προηγούμενο καθώς, εδώ η RAM δεν είναι μέρος του φίλτρου επομένως, πρέπει να συνδέσουμε μνήμη σε αυτό.

Ανάλογα με τις προδιαγραφές του φίλτρου, το διαφορετικό στα περιεχόμενα κάθε RAM που συνδέσαμε είναι το επίπεδο του θορύβου, διότι το μέγεθος αλλάζει μόνο στη σχεδίαση. Εφόσον έχουμε μία αρχικοποιημένη RAM, μπορούμε να την κάνουμε instantiate στο Testbench προσέχοντας ώστε οι είσοδοι και οι έξοδοι του instantiation, τόσο του φίλτρου όσο και της μνήμης, να είναι συνδεδεμένες σωστά.

Για να είναι σωστά γραμμένο το παρακάτω Testbench, απαιτείται να ορίσουμε κατάλληλα και τις τιμές των S_c και S_noise.

```
1 `timescale 1 ns / 1 ns
2
3 module Filter_Testbench_no_memory;
4
5     integer file;
6     reg    Clock, Reset;
7     reg    [4:0] s_c;
8     reg    [5:0] s_noise;
9     wire   [20:0] Rom_address; // Allazei analoga mnimi
10    wire    Rom_enable;
11    wire   signed [19:0] Rom_value;
12    wire   signed [21:0] Result;
13
14    Bilateral_zero_padded_no_memory T1(Clock,Result[21:0],Reset,s_c,s_noise,Rom_address[20:0],Rom_value[19:0],Rom_enable);
15
16    Filter_Ram F1(Rom_address[13:0], Clock, , Rom_enable, 1'b0, Rom_value[19:0]); //Ousiastika Ram
17
18    always #10 Clock <= Clock + 1;
19
20    always @(posedge Clock) $fdisplay(file,"%d", Result);
21
22    initial begin
23
24        Clock    <= 0;
25        Reset    <= 0;
26        s_c      <= 5'b00001; //s_c = 1
27        s_noise  <= 6'b111100; //s_noise = 60
28        file = $fopen("C:/Users/CHRIS/Desktop/thesis/system_tasks/Filter_Results_Mdlsn.txt", "w");
29
30        #1280020 begin // Allazei analoga mnimi
31
32            $fclose(file);
33            $finish;
34
35        end
36    end
37 endmodule
```

Εικόνα 21: Κώδικας Verilog Testbench για φίλτρο χωρίς ενσωματωμένη μνήμη

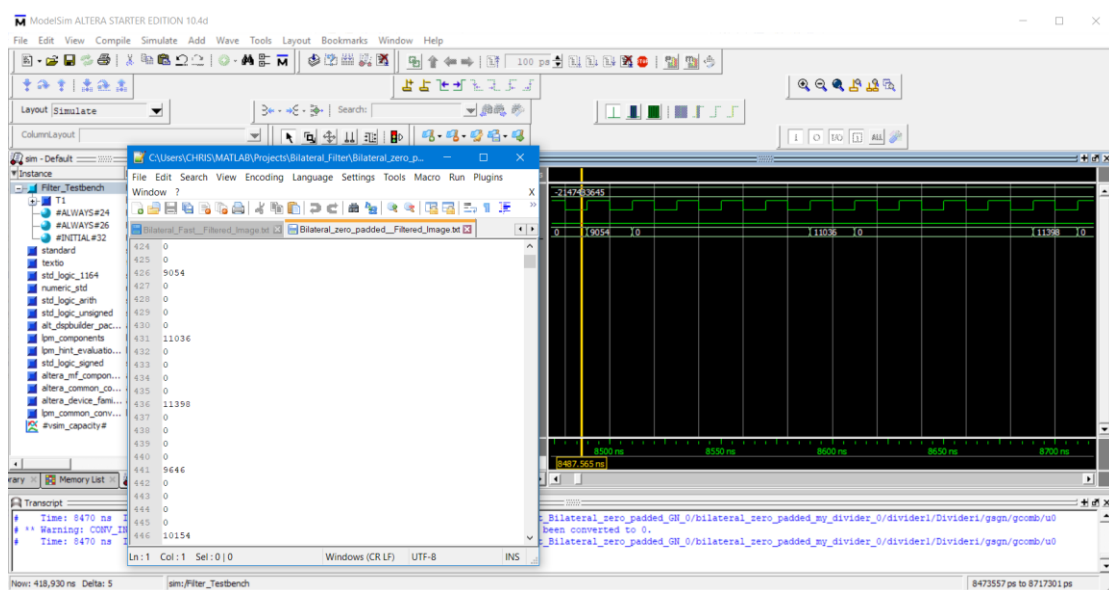




5.3 Εξομοίωση Φίλτρου

Αφού έχουμε περάσει από Compilation στο Quartus το project του φίλτρου, σε οποιοδήποτε από τα δύο σενάρια, το επόμενο βήμα είναι να τρέξουμε την εξομοίωση χρησιμοποιώντας το ανάλογο Testbench.

Αν έχουμε ορίσει τον φάκελο στον οποίο βρίσκεται το Testbench και το αντίστοιχο όνομα αρχείου τότε μας μένει να διαλέξουμε από το μενού Tools -> Run Simulation Tool -> RTL Simulation.



Εικόνα 22: Εξομοίωση στο Modelsim και επιβεβαίωση των αποτελεσμάτων





Κεφάλαιο 6

Διασταύρωση αποτελεσμάτων Modelsim-Simulink

Για να διαπιστώσουμε την ταύτιση ή τη διαφορά μεταξύ των αποτελεσμάτων μας, θα παρουσιάσουμε παρακάτω τον τρόπο με τον οποίο αρχικά συλλέγουμε τα δεδομένα αυτά, πώς τα ομαδοποιούμε ως μία εικόνα και τέλος θα δείξουμε και τη λογική πίσω από τον υπολογισμό όλων των ποσοτήτων που χρησιμοποιήσαμε κατά την εκτίμηση των αποτελεσμάτων.

6.1 Τρόπος αποθήκευσης αποτελεσμάτων φίλτρου

Η εκτέλεση του σεναρίου μας, τόσο στο Simulink όσο και στο Modelsim, παράγει ένα αρχείο τύπου .txt το οποίο περιλαμβάνει όλα τα σήματα εξόδου του φίλτρου. Καθώς το φίλτρο δίνει την έξοδο που θέλουμε να κρατήσουμε σε συγκεκριμένες χρονικές στιγμές, όλες τις υπόλοιπες θα έχει μηδενική έξοδο.

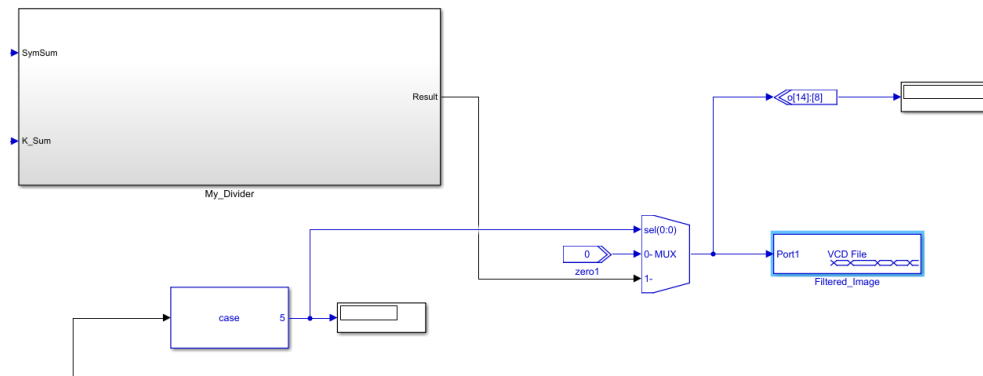
```
90 0
91 0
92 0
93 10702
94 0
95 0
96 0
97 17689
98 0
99 0
100 0
101 20693
102 0
103 0
104 0
105 15316
106 0
107 0
108 0
109 14219
110 0
111 0
112 0
```

Εικόνα 23: Ο τρόπος αποθήκευσης των εξαγόμενων δεδομένων από το Simulink





Λόγω του ότι στο Simulink το μοντέλο μας δημιουργείται με blocks του DSP Builder, η αποθήκευση της εξόδου γίνεται διαφορετικά από το Modelsim, στο οποίο γράφουμε εντολές σε Verilog.



Εικόνα 24: Blocks για έλεγχο τιμής και αποθήκευσης εξόδου

6.2 Ανάγνωση Αποτελεσμάτων στο MATLAB

Καθώς το αρχείο εξόδου των δεδομένων αποθηκεύεται πάντα στον ίδιο φάκελο με το Simulink μοντέλο, μπορούμε εύκολα να το διαβάσουμε με τις ανάλογες εντολές διαχείρισης αρχείων του MATLAB. Για την περίπτωση του Modelsim πρέπει να δώσουμε την πλήρη διαδρομή αν θέλουμε να προσπελαστεί το αρχείο. Η διαδρομή μας, είτε πρόκειται για το αρχείο εξόδου του Simulink είτε του Modelsim, αποθηκεύεται στη μεταβλητή fileID. Παρακάτω εξηγούμε πως επιβεβαιώνουμε την ομοιότητα των δεδομένων των δύο αρχείων.

```
%Image Input
im=double(rgb2gray(imread('dog_professor.jpg')));
%im=noisy_im(1:7,1:7);
[x,y]=size(im);

fileID = fopen( 'Bilateral_zero_padded_Filtered_Image.txt' , 'r');
%fileID = fopen( 'C:\Users\CHRIS\Desktop\thesis\system_tasks\FILTER_Results_Mdlsm.txt' , 'r');
formatSpec= '%d';

Results= fscanf( fileID , formatSpec);
fclose(fileID);

Results=nonzeros(Results);
```

Εικόνα 25: Κώδικας ανάγνωσης κύριας εικόνας και τιμών από αρχείο κειμένου





6.3 Δημιουργία της Φιλτραρισμένης Εικόνας-Simulink ή Modelsim

Έχοντας αποθηκευμένες όλες τις τιμές εξόδου του φίλτρου στον πίνακα Results και έχοντας αφαιρέσει όλες τις μηδενικές τιμές που περιλαμβάνονται στα αρχεία εξόδου, σειρά έχει η τοποθέτηση τους στις κατάλληλες θέσεις της τελικής εικόνας. Καθώς η έξοδος του φίλτρου μας είναι 22 bit fixed point, με 14 bit ακέραιο μέρος και 8 δεκαδικό, πρέπει να διαιρέσουμε την έξοδο με το 256 διότι κάθε αριθμός διαβάζεται από το .txt αρχείο σαν ένας δυαδικός ακέραιος εύρους 22bit.

```
for i=1:x
    for j=1:y

        Remade(i,j)=Results(count)/256;
        count=count+1;

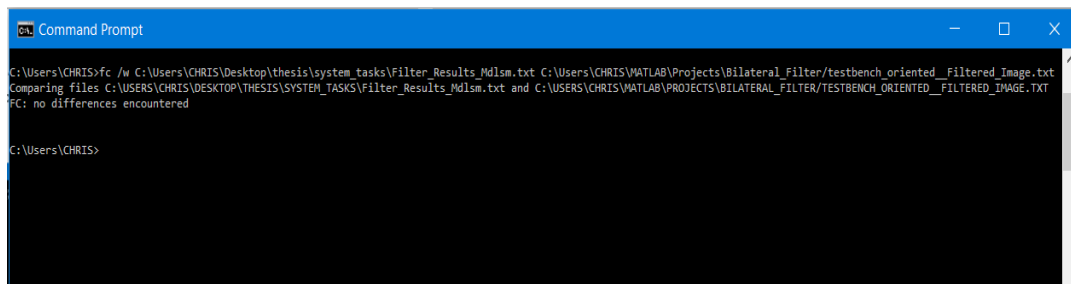
    end
end
```

Εικόνα 26: Κώδικας δημιουργίας τελικής εικόνας

6.4 Έλεγχος τελικών αποτελεσμάτων Simulink – Modelsim

Αφού έχουμε έτοιμα τα δύο μας αρχεία, από την εξομίωση στο Simulink και στο Modelsim, πρέπει να ελέγξουμε αν τελικά είναι ίδια. Για τον έλεγχο αυτό χρησιμοποιούμε μία εντολή γραμμής εντολών του λειτουργικού συστήματος, την εντολή fc. Η συγκεκριμένη εντολή μας εξυπηρετεί αφού συγκρίνουμε τις δύο εικόνες ως προς την ομοιότητα των δεδομένων τους.

Για την εκτέλεσή της εντολής απαιτείται και η προσθήκη της πλήρους διαδρομής του φακέλου αποθήκευσης καθενός εκ των δύο αρχείων εξόδων πριν την εκτέλεσή της. Η παραμικρή διαφορά ανάμεσα στα δύο αρχεία, τα κενά μεταξύ τιμών λόγου χάρη, θα δώσει αποτέλεσμα μη ομοιότητας των αρχείων. Η λύση σε αυτό το θέμα, είναι η χρήση της εντολής fc /w [pathname1] [pathname2], η οποία αγνοεί τα κενά που προηγούνται ή είναι μετά τα δεδομένα της εξόδου.



Εικόνα 27: Εντολή ελέγχου της ομοιότητας των εξαγόμενων αποτελεσμάτων





6.5 Υπολογισμός του MSE

Για την μέτρηση της ομοιότητας μεταξύ της εικόνας αναφοράς και της εικόνας με θόρυβο ή της εικόνας μετά την αποθορυβοποίηση υπολογίζουμε το μέσο τετραγωνικό σφάλμα (*Mean Squared Error-MSE*). Όταν δύο εικόνες είναι πανομοιότυπες τότε το *MSE* ισούται με μηδέν. Επιπλέον, το μέσο τετραγωνικό σφάλμα θα μας χρειαστεί στην συνέχεια για την εύρεση του σηματοθορυβικού δείκτη $PSNR_{dB}$.

$$MSE = \frac{1}{X \cdot Y} \sum_{x=1}^X \sum_{y=1}^Y (im(x, y) - altered_im(x, y))^2$$

$y \rightarrow$ αριθμός στήλης πίνακα

$x \rightarrow$ αριθμός γραμμής πίνακα

$im(x, y) \rightarrow$ δισδιάστατος πίνακας αρχικής εικόνας

$altered_im(x, y) \rightarrow$ δισδιάστατος πίνακας αλλαγμένης εικόνας

6.6 Υπολογισμός του PSNR

Ο μέγιστος λόγος σήματος προς θόρυβο ή αλλιώς $PSNR_{dB}$ (*Peak Signal-to-Noise Ratio*) είναι ένας από τους δύο δείκτες που παίρνουμε υπόψιν μας και μας βοηθά να εκτιμήσουμε την απόδοση της αποθορυβοποίησης. Ο μέγιστος λόγος σήματος προς θόρυβο είναι αντιστρόφως ανάλογος της τετραγωνικής ρίζας του μέσου τετραγωνικού σφάλματος (*MSE*). Πιο συγκεκριμένα:

$$PSNR_{dB} = 20 \cdot \log_{10} \left(\frac{GV_{max}}{\sqrt{MSE}} \right)$$

$MSE \rightarrow$ μέσο τετραγωνικό σφάλμα

$GV_{max} \rightarrow$ μέγιστη τιμή εικονοστοιχείου δηλαδή $2^n - 1$ για n bit αναπαράσταση





6.7 Υπολογισμός του MSSIM

Η μέση δομική ομοιότητα είναι ένας από τους σημαντικότερους δείκτες που μπορούμε να λάβουμε υπόψιν μας στις μετρήσεις μας ώστε να μας βοηθήσει να εκτιμήσουμε και να συγκρίνουμε την ποιότητα των εικόνων μας μετά την αποθρομβοποίηση

6.6.1 SSIM

Αρχικά, χρησιμοποιώντας παράθυρα μεγέθους $N \times N$, στην εικόνα αναφοράς x και στην εικόνα μετά την επεξεργασία y , υπολογίζουμε την τοπική δομική ομοιότητα *SSIM*. Πιο συγκεκριμένα:

$$SSIM(x, y) = l(x, y) \cdot c(x, y) \cdot s(x, y)$$

$$s(x, y) \rightarrow \text{διαφορά δομής παραθύρων}$$

$$c(x, y) \rightarrow \text{διαφορά αντίθεσης παραθύρων}$$

$$l(x, y) \rightarrow \text{διαφορά φωτεινότητας παραθύρων}$$

Για να βρούμε τη δομική ομοιότητα, το σύστημά μας πρέπει να συγκρίνει τις εικόνες ως προς τις τρεις προαναφερθείσες παραμέτρους. Μία σημαντική λεπτομέρεια είναι πως οι παράμετροι είναι σχετικά ανεξάρτητες αναμεταξύ τους. Επομένως, μία αλλαγή σε κάποιο παράμετρο δεν σημαίνει αλλαγή στις άλλες δύο.

6.6.2 Luminance

Η πρώτη παράμετρος είναι η φωτεινότητα. Ως σύγκριση της φωτεινότητας δύο παραθύρων ορίζουμε:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}$$





$$c_1 \rightarrow k_1 L, k_1 \rightarrow 0.01$$

$$L \rightarrow 2^n - 1, n \rightarrow \text{bit ανά pixel}$$

$$\mu_x, \mu_y \rightarrow \text{μέση τιμή παραθύρου εικόνων } x, y$$

6.6.3 Contrast

Η δεύτερη παράμετρος είναι η αντίθεση. Η αντίθεση όπως και η φωτεινότητα δύσκολα εκτιμάται στο σύνολο της εικόνας για αυτό και η σύγκριση μέσω μικρότερων παραθύρων είναι πιο αποτελεσματική. Ο τύπος για την σύγκριση δύο παραθύρων ορίζεται ως:

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$$

$$c_2 \rightarrow k_2 L, k_2 \rightarrow 0.03$$

$$\sigma_x \rightarrow \left(\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2 \right)^{\frac{1}{2}}$$

$$\sigma_x, \sigma_y \rightarrow \text{τυπική απόκλιση παραθύρου εικόνων } x, y$$

6.6.4 Structure

Η τρίτη παράμετρος που συγκρίνουμε είναι η δομή. Ορίζουμε την δομική πληροφορία μιας εικόνας, ως τα χαρακτηριστικά που αναπαριστούν τα αντικείμενα στην εικόνα αυτή, ανεξάρτητα από την μέση φωτεινότητα και αντίθεση. Συνεπώς την σύγκριση της δομής δύο παραθύρων την ορίζουμε ως εξής:

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x + \sigma_y + c_3}$$

$$c_3 \rightarrow \frac{c_2}{2}$$

$$\sigma_{xy} \rightarrow \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$





$\sigma_{xy} \rightarrow$ συνδιασπορά παραθύρου x και y

6.6.5 Mean SSIM

Αφού υπολογίσουμε τις τιμές του $SSIM$ όλων των παραθύρων της εικόνας βρίσκουμε τον μέσο όρο τους, ο οποίος θα μας δώσει τη μέση δομική ομοιότητα. Συνεπώς, αν έχουμε J παράθυρα τότε:

$$MSSIM \rightarrow \frac{1}{J} \sum_{j=1}^J SSIM(x_j, y_j)$$

$x_j, y_j \rightarrow$ παράθυρο j από τα J παράθυρα που έχουμε χωρίσει την εικόνα x, y

Για τον υπολογισμό των MSE , $PSNR_{dB}$ και $MSSIM$, χρησιμοποιήθηκαν οι έτοιμες συναρτήσεις του MATLAB.























Κεφάλαιο 7

Σύγκριση αποτελεσμάτων MATLAB-Simulink

Σε αυτό το σημείο, καθώς είμαστε σίγουροι πως το μοντέλο μας στο Simulink παράγει σωστά αποτελέσματα και ταυτόχρονα αποτελέσματα που είναι ίδια με αυτά του Modelsim, δεν έχουμε παρά να το συγκρίνουμε με την υλοποίηση μας στο MATLAB (κινητής υποδιαστολής) ώστε να δούμε αν υπάρχουν αποκλίσεις. Για αυτόν τον λόγο, για ένα σετ από εικόνες και για ένα πλήθος από διαφορετικές τιμές της παραμέτρου σ_{noise} , θα παρουσιάσουμε τις εικόνες και τις τιμές των $PSNR_{dB}$ και $MSSIM$ που προκύπτουν από τις μετρήσεις μας.

7.1 Φιλτράρισμα εικόνας 100x100

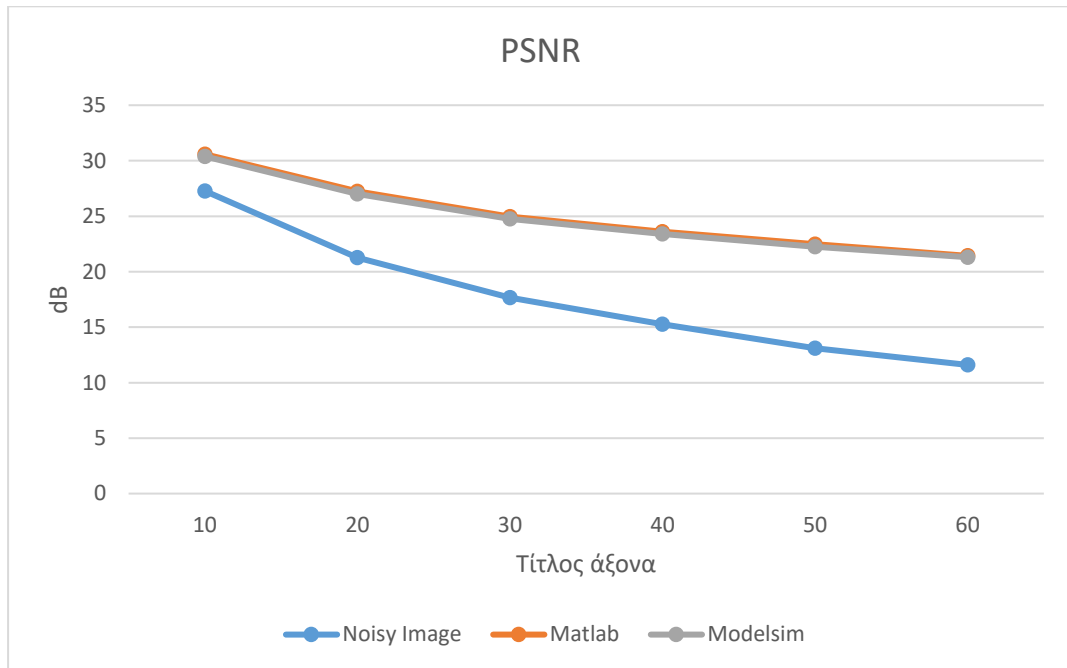
Image 100x100	Αρχική Εικόνα	Εικόνα με Θόρυβο	Φιλτραρισμένη Εικόνα
$\sigma_{noise} = 10$			
$\sigma_{noise} = 20$			
$\sigma_{noise} = 30$			
$\sigma_{noise} = 40$			
$\sigma_{noise} = 50$			
$\sigma_{noise} = 60$			

Εικόνα 28: Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 100x100

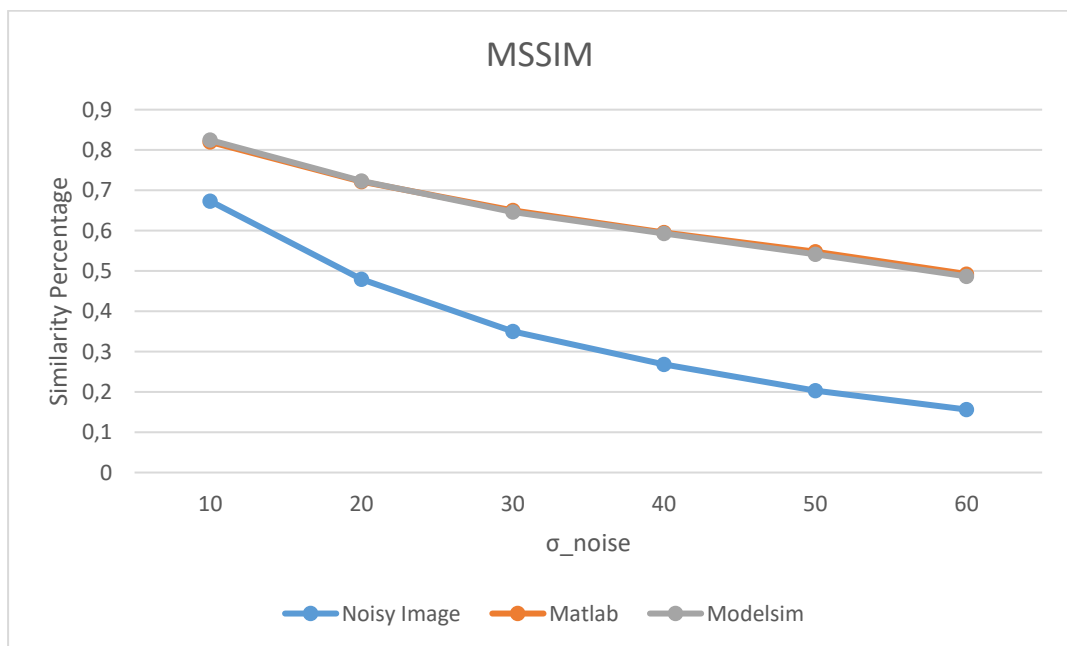




7.2 Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 100x100



Γράφημα 1: PSNR για εικόνα 100x100



Γράφημα 2: MSSIM για εικόνα 100x100





Πίνακας 2: Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 100x100

σ_{noise} 100x100 Dog	Noisy Image PSNR _{db}	Noisy Image MSSIM	MATLAB PSNR _{db}	MATLAB MSSIM	Modelsim PSNR _{db}	Modelsim MSSIM
10	27,2715	0,6733	30,5846	0,8199	30,3894	0,8245
20	21,2651	0,4794	27,2363	0,7214	27,0150	0,7229
30	17,6591	0,3498	24,9561	0,6504	24,7630	0,6465
40	15,2764	0,2679	23,6093	0,5954	23,3997	0,5931
50	13,1154	0,2030	22,4882	0,5481	22,2336	0,5412
60	11,5991	0,1563	21,4273	0,4925	21,3071	0,4866

7.3 Φιλτράρισμα εικόνας 150x150

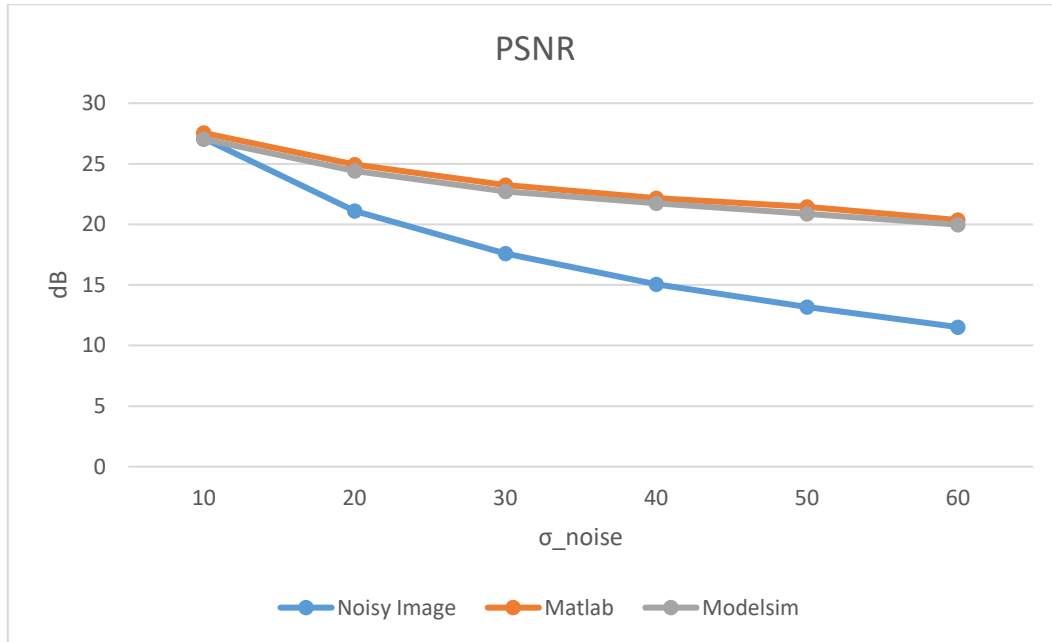
Image 150x150	Αρχική Εικόνα	Εικόνα με Θόρυβο	Φιλτραρισμένη Εικόνα
$\sigma_{noise} = 10$			
$\sigma_{noise} = 20$			
$\sigma_{noise} = 30$			
$\sigma_{noise} = 40$			
$\sigma_{noise} = 50$			
$\sigma_{noise} = 60$			

Εικόνα 29: Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 150x150

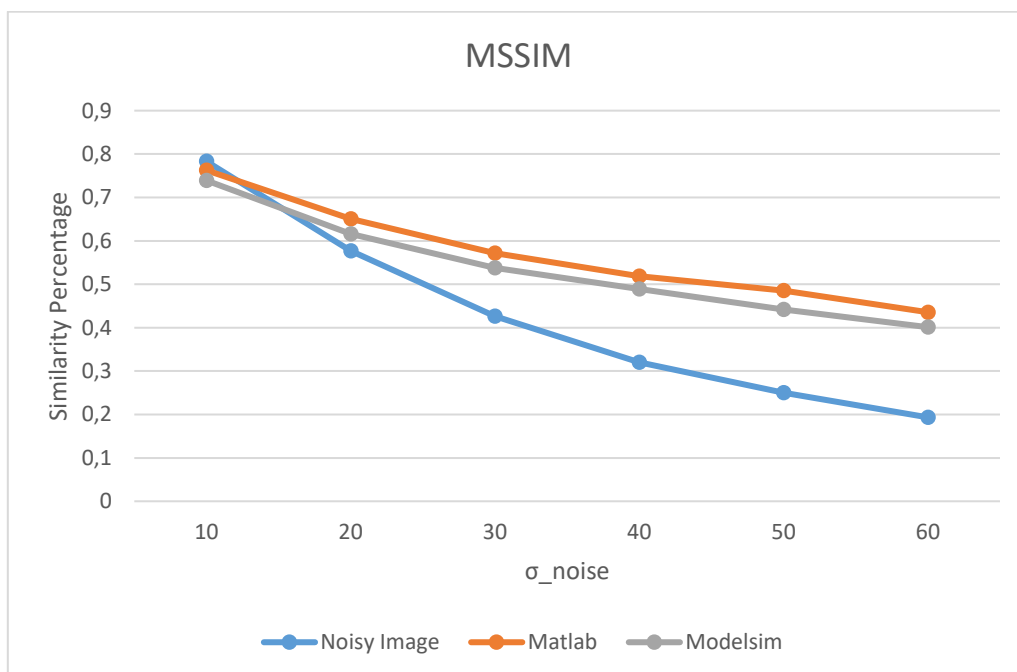




7.4 Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 150x150



Γράφημα 3: PSNR για εικόνα 150x150



Γράφημα 4: MSSIM για εικόνα 150x150














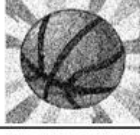








Πίνακας 3: Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 150x150

σ_{noise} 150x150 raccoons	Noisy Image PSNR _{db}	Noisy Image MSSIM	MATLAB PSNR _{db}	MATLAB MSSIM	Modelsim PSNR _{db}	Modelsim MSSIM
10	27,1024	0,7831	27,5446	0,7626	27,0382	0,7391
20	21,1075	0,5770	24,9510	0,6509	24,4072	0,6157
30	17,5945	0,4262	23,2452	0,5715	22,7254	0,5375
40	15,0698	0,32	22,1615	0,5187	21,73	0,4893
50	13,1799	0,2502	21,4486	0,4855	20,8780	0,4421
60	11,5220	0,1934	20,3590	0,4356	19,9817	0,4012

7.5 Φιλτράρισμα εικόνας 200x200

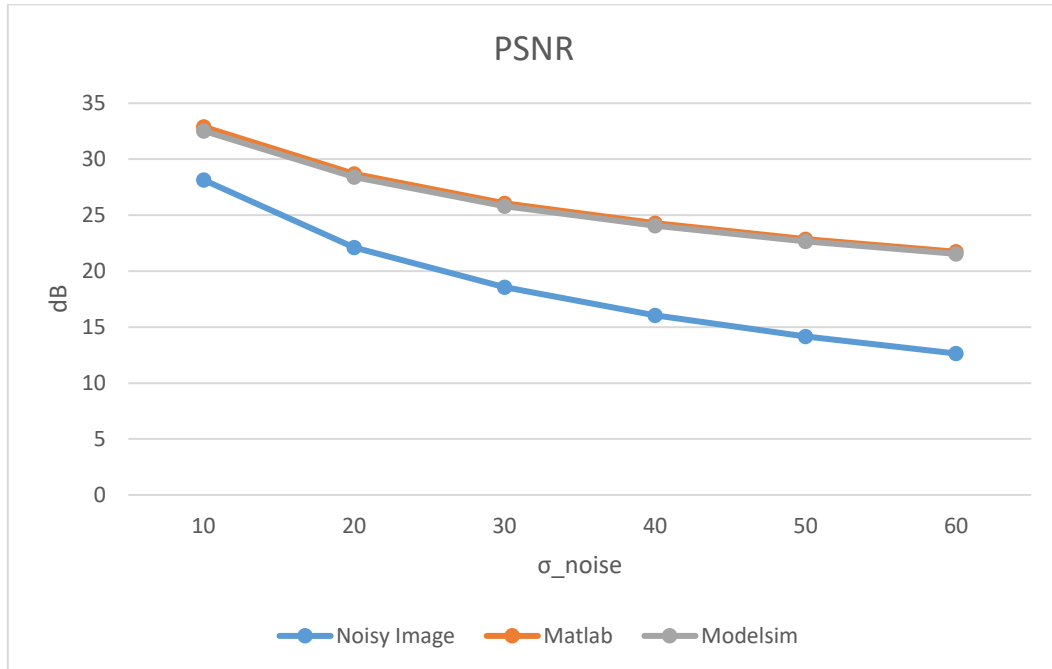
Image 200x200	Αρχική Εικόνα	Εικόνα με Θόρυβο	Φιλτραρισμένη Εικόνα
$\sigma_{noise} = 10$			
$\sigma_{noise} = 20$			
$\sigma_{noise} = 30$			
$\sigma_{noise} = 40$			
$\sigma_{noise} = 50$			
$\sigma_{noise} = 60$			

Εικόνα 30: Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 200x200

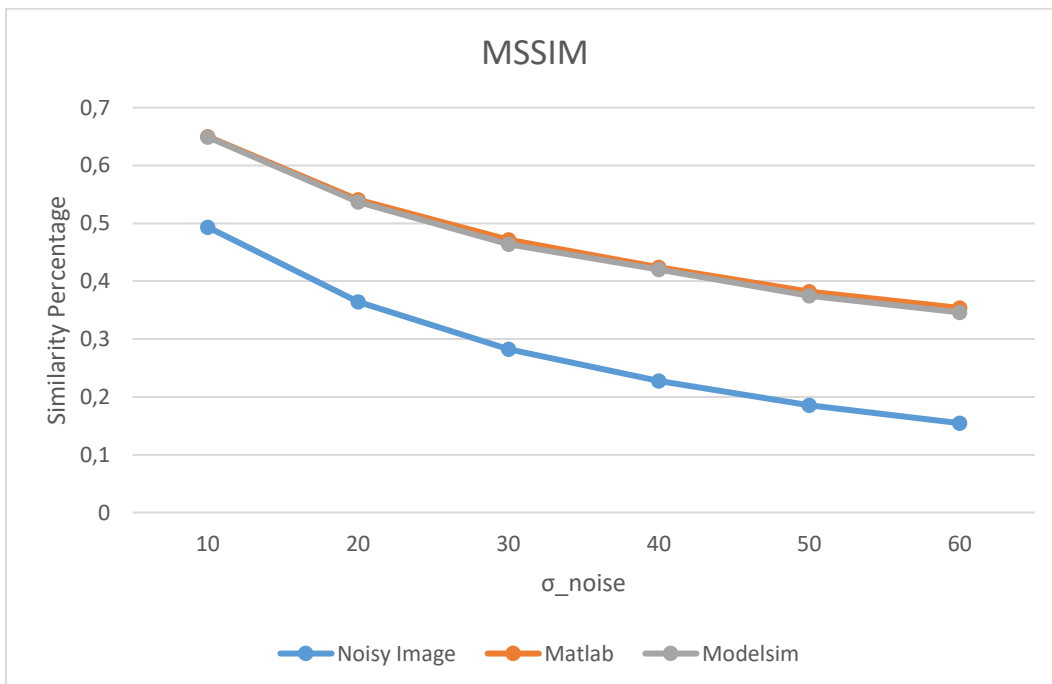




7.6 Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 200x200



Γράφημα 5: PSNR για εικόνα 200x200



Γράφημα 6: MSSIM για εικόνα 200x200





Πίνακας 4: Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 200x200

σ_{noise} 200x200 basketball	Noisy Image PSNR _{db}	Noisy Image MSSIM	MATLAB PSNR _{db}	MATLAB MSSIM	Modelsim PSNR _{db}	Modelsim MSSIM
10	28,1476	0,4934	32,8949	0,6501	32,5306	0,6492
20	22,1205	0,3643	28,7014	0,5406	28,3993	0,5368
30	18,5627	0,2822	26,0716	0,4715	25,8158	0,4641
40	16,0581	0,2275	24,2809	0,4242	24,0591	0,4200
50	14,1558	0,1855	22,8500	0,3820	22,6529	0,3747
60	12,6399	0,1546	21,7288	0,3538	21,5458	0,3461

7.7 Φιλτράρισμα εικόνας 330x330

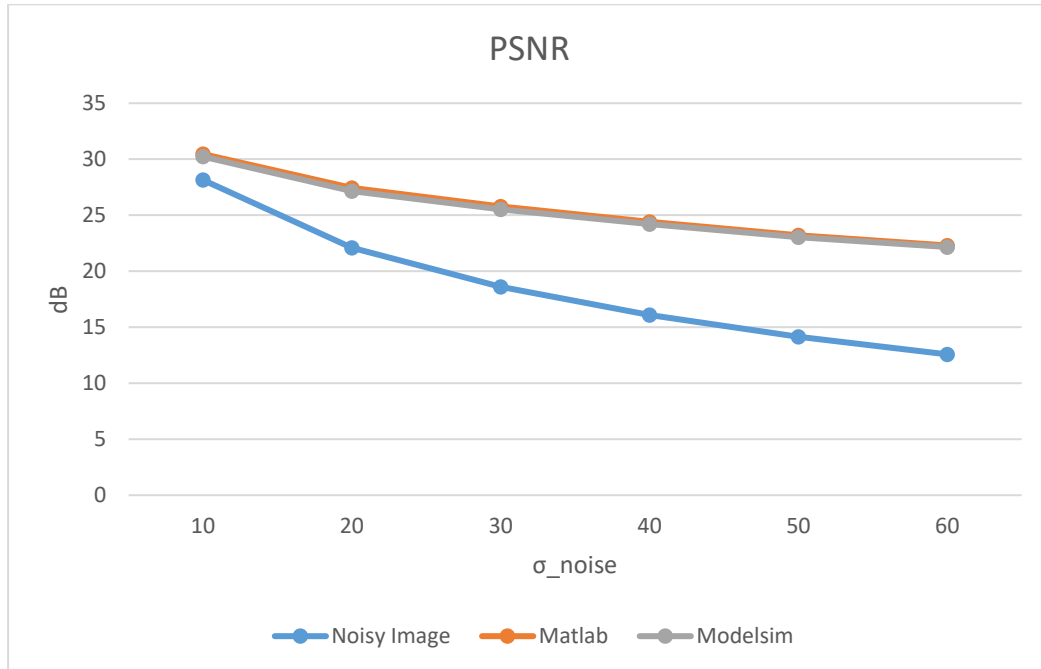
Image 330x330	Αρχική Εικόνα	Εικόνα με Θόρυβο	Φιλτραρισμένη Εικόνα
$\sigma_{noise} = 10$			
$\sigma_{noise} = 20$			
$\sigma_{noise} = 30$			
$\sigma_{noise} = 40$			
$\sigma_{noise} = 50$			
$\sigma_{noise} = 60$			

Εικόνα 31: Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 330x330

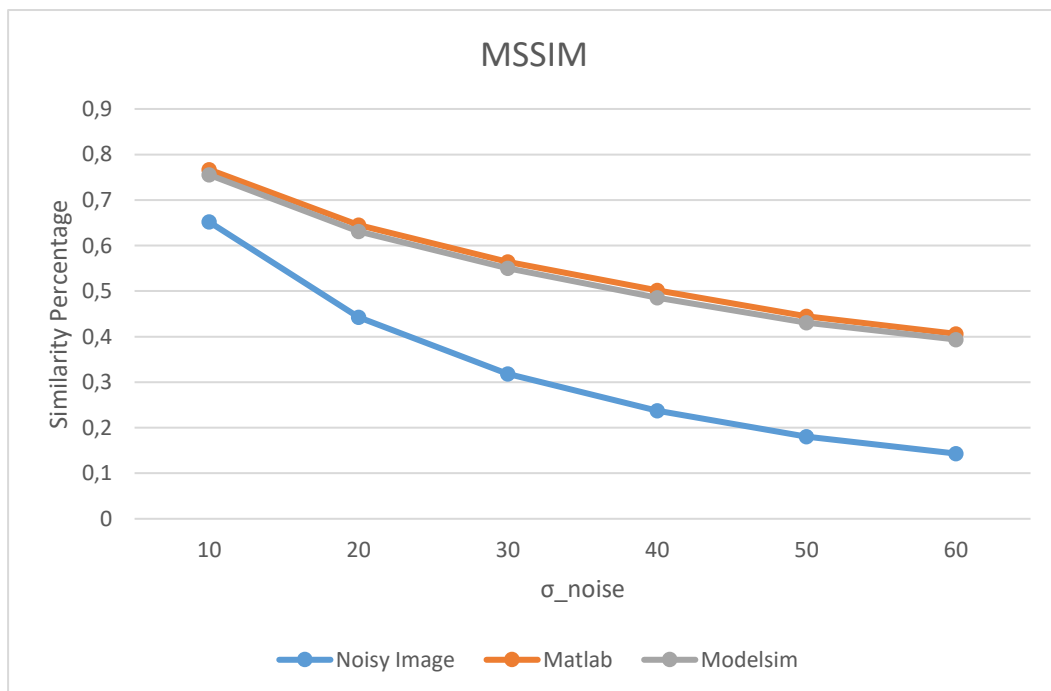




7.8 Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 330x330



Γράφημα 7: PSNR για εικόνα 330x330



Γράφημα 8: MSSIM για εικόνα 33x330





Πίνακας 5: Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 330x330

σ_{noise} 330x330 food	Noisy Image PSNR _{db}	Noisy Image MSSIM	MATLAB PSNR _{db}	MATLAB MSSIM	Modelsim PSNR _{db}	Modelsim MSSIM
10	28,1477	0,6518	30,4591	0,7669	30,2317	0,7552
20	22,0974	0,4428	27,4507	0,645	27,1611	0,631
30	18,6057	0,3185	25,7739	0,5642	25,5288	0,5504
40	16,1057	0,2375	24,4223	0,5015	24,1964	0,4853
50	14,1467	0,1806	23,2129	0,4448	23,0406	0,4308
60	12,5784	0,1433	22,2997	0,4062	22,163	0,3937

7.9 Φιλτράρισμα εικόνας 512x512

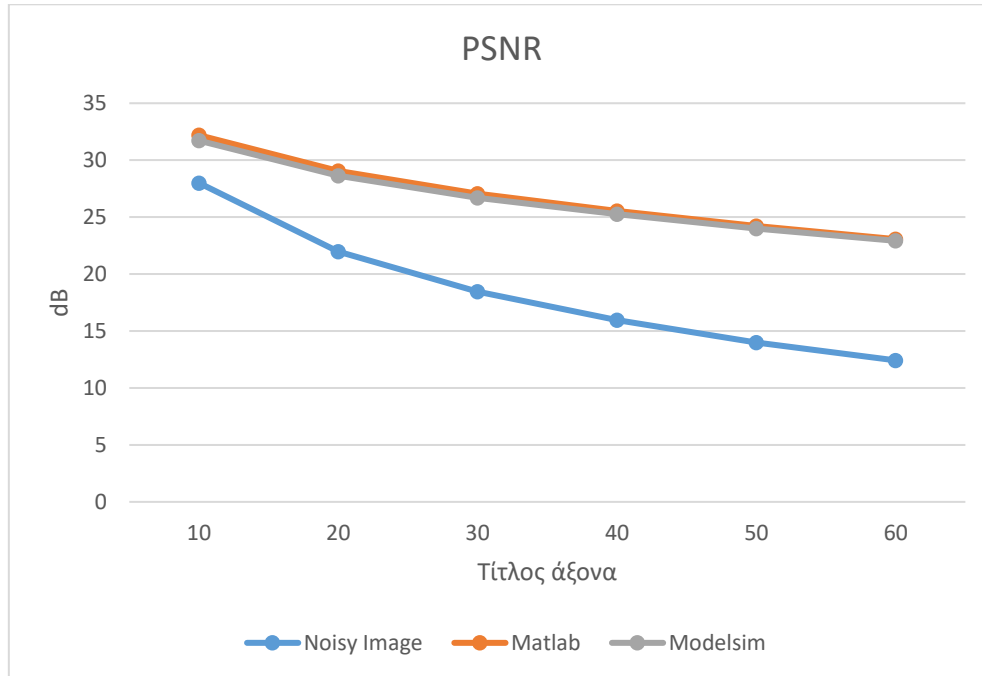
Image 512x512	Αρχική Εικόνα	Εικόνα με Θόρυβο	Φιλτραρισμένη Εικόνα
$\sigma_{noise} = 10$			
$\sigma_{noise} = 20$			
$\sigma_{noise} = 30$			
$\sigma_{noise} = 40$			
$\sigma_{noise} = 50$			
$\sigma_{noise} = 60$			

Εικόνα 32: Πίνακας αποτελεσμάτων Φιλτραρίσματος εικόνας 512x512

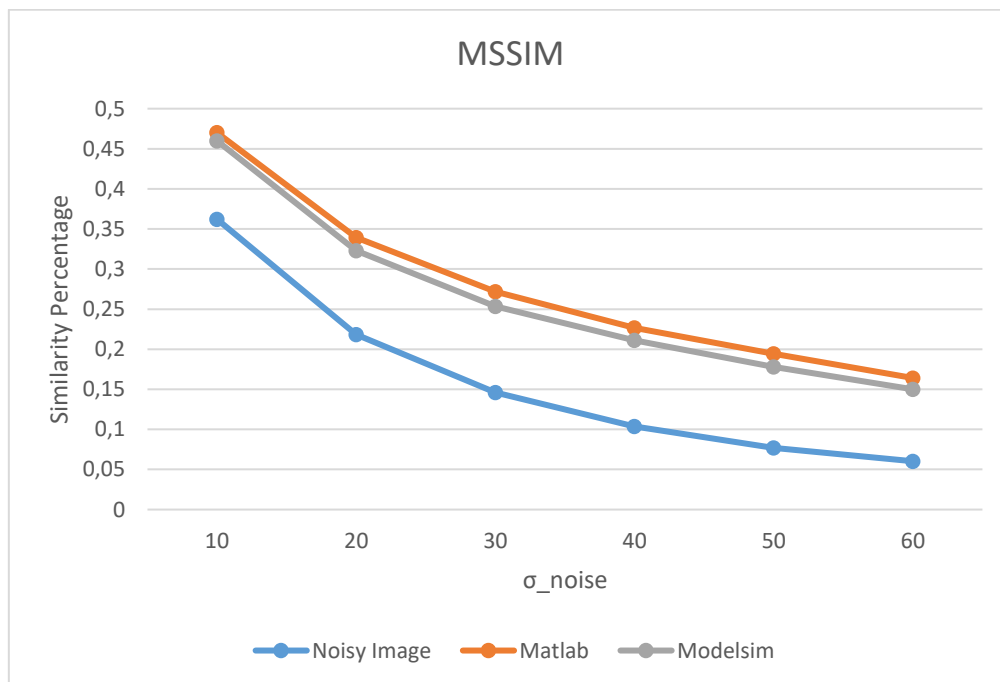




7.10 Ποιοτικά και ποσοτικά αποτελέσματα εικόνας 512x512



Γράφημα 9: PSNR για εικόνα 512x512



Γράφημα 10: MSSIM για εικόνα 512x512





Πίνακας 6: Ποσοτικά αποτελέσματα φιλτραρίσματος εικόνας 512x512

σ_{noise} 512x512 landscape	Noisy Image $PSNR_{db}$	Noisy Image $MSSIM$	MATLAB $PSNR_{db}$	MATLAB $MSSIM$	Modelsim $PSNR_{db}$	Modelsim $MSSIM$
10	28,0008	0,3621	32,1931	0,47	31,7173	0,4599
20	21,9747	0,2184	29,0505	0,3394	28,6356	0,3227
30	18,4533	0,1459	27,0503	0,2716	26,7003	0,2536
40	15,9563	0,1036	25,5335	0,2265	25,2736	0,2110
50	13,9919	0,0768	24,2095	0,1943	24	0,1781
60	12,4235	0,0603	23,0681	0,1641	22,9205	0,1500

7.11 Χρονικές Εκτιμήσεις

Παρακάτω, παραθέτουμε τους χρόνους εφαρμογής της διαδικασίας φιλτραρίσματος για τις διάφορες εικόνες που χρησιμοποιήσαμε. Η μαθηματική σχέση με την οποία υπολογίζουμε τα steps που απαιτούνται στο Simulink για να φιλτραριστεί πλήρως μία εικόνα είναι:

$$Steps_{simulink} = 2[2(y + 4) + 2] + (x - 1)[(cycles \cdot y) + 4] + (cycles \cdot y)$$

$y \rightarrow$ πλήθος γραμμών (rows) δισδιάστατης εικόνας

$x \rightarrow$ πλήθος στηλών (columns) δισδιάστατης εικόνας

$cycles \rightarrow$ κύκλοι ρολογιού για τον υπολογισμό μίας εξόδου του φίλτρου

Έχοντας υπολογίσει τα συνολικά steps φιλτραρίσματος της εικόνας, μπορούμε εν συνεχεία να υπολογίσουμε και τον αντίστοιχο χρόνο συνολικού φιλτραρίσματος. Καθώς στο Simulink η εξομοίωση ξεκινάει από $Steps = 0$, πρέπει να προσθέσουμε την περίοδο ρολογιού άλλη μία φορά. Η πράξη έχει ως εξής:

$$time_{filter} = (Steps_{simulink} + 1)period_{clock}$$

$period_{clock} \rightarrow$ περίοδος ρολογιού





$Steps_{simulink} \rightarrow$ κύκλοι ρολογιού για πλήρη έξοδο του φίλτρου

Μία ακόμα μετρική που μας ενδιαφέρει είναι τα καρέ ανά δευτερόλεπτο (Frames per second-FPS). Ο υπολογισμός των FPS μας δείχνει την ικανότητα του φίλτρου να επεξεργαστεί βίντεο και για περισσότερα από $30 \frac{\text{καρέ}}{\text{δευτερόλεπτο}}$ κάτι τέτοιο είναι εφικτό. Ο υπολογισμός της συγκεκριμένης μετρικής γίνεται ως εξής:

$$FPS = \frac{1}{time_{filter}}$$

$time_{filter} \rightarrow$ χρόνος συνολικού φιλτραρίσματος σε δευτερόλεπτα

Ο αριθμός από steps που χρειαζόμαστε για να μας δώσει το φίλτρο όλα τα αποτελέσματα για κάθε εικόνα βρίσκεται στην τρίτη στήλη του Πίνακα 7. Από την άλλη, αν θέλουμε να τρέξουμε κάποιο από τα δύο Testbenches που έχουν δημιουργηθεί στο Modelsim, τότε πρέπει να χρησιμοποιήσουμε τις τιμές στην τέταρτη στήλη. Τέλος, ο σχεδιασμός μας δίνει έξοδο κάθε 4 κύκλους ρολογιού.

Πίνακας 7: Χρόνοι φιλτραρίσματος ανά εικόνα

Μέγεθος Εικόνας	Pixels	Simulink Steps	Modelsim ns (1 step = 20 ns)	FPS
100x100	10.000	40.816	816.340	1224,979
150x150	22.500	91.216	1.824.340	548,143
200x200	40.000	161.616	3.232.340	309,373
330x330	108.900	438.256	8.765.140	114,088
512x512	262.144	1.052.688	21.053.780	47,497





Κεφάλαιο 8

Απαιτήσεις σε πόρους

Έχοντας παρουσιάσει τα αποτελέσματα χρήσης του φίλτρου και τις χρονικές μετρήσεις, θα κάνουμε μία πιο εκτενή ανάλυση των πόρων που κάθε σενάριο χρησιμοποιεί. Τα blocks που χρησιμοποιήθηκαν έχουν συγκεκριμένο αριθμό, ανάλογα το σενάριο και τη λειτουργία τους, και το μόνο κομμάτι που μεταβάλλεται είναι το πλήθος των πόρων καθώς αυτό έχει άμεση σχέση με το FPGA που θα χρησιμοποιηθεί.

8.1 Φίλτρο με ενσωματωμένη μνήμη

Ολοκληρώνοντας τη σύνθεση στο Quartus, παρουσιάζουμε τους πόρους του ολοκληρωμένου FPGA που το φίλτρο χρησιμοποιεί. Η μεγαλύτερη διαφορά που παρατηρήσαμε βρίσκεται στην ποσότητα της μνήμης του FPGA ολοκληρωμένου που απασχολείται, από το σενάριο με ανάλυση 100x100 στο σενάριο με ανάλυση 330x330.

Πίνακας 8: Χρήση πόρων για την υλοποίηση του φίλτρου με ενσωματωμένη μνήμη

Εικόνα 100x100			Εικόνα 330x330	
FPGA: Cyclone IV E	Αριθμός Πόρων	Συνολικό Ποσοστό	Αριθμός Πόρων	Συνολικό Ποσοστό
Συνολικά Logic Elements	5.100	4%	5.268	4%
Registers	1602	-	1609	-
I/O (bits)	24	5%	24	5%
Συνολική Μνήμη (bits)	207.840	5%	2.204.240	55%
Multipliers	29	5%	29	5%
Συχνότητα: 50MHz				





Η συνολική μνήμη του FPGA ολοκληρωμένου μας είναι 3.981.312 bits. Κατά συνέπεια υπάρχει ένα άνω όριο στην μέγιστη ανάλυση εικόνας που μπορούμε να φιλτράρουμε. Θα επιδιώξουμε να υπολογίσουμε άνω όριο με ίσες τις δύο διαστάσεις. Με δεδομένο ότι κάθε pixel αναπαρίσταται με 20 bit τότε:

$$\text{ανάλυση}_{max} = \sqrt{\frac{3.981.312}{20}} \cong 446$$

Άρα, σχεδόν όλη η μνήμη του FPGA ολοκληρωμένου μπορεί να απασχοληθεί από μία εικόνα με ανάλυση 446x446 και αναπαράσταση 20 bit.

8.2 Φίλτρο χωρίς ενσωματωμένη μνήμη

Παρακάτω, παρουσιάζουμε τους πόρους που χρησιμοποιεί το φίλτρο χωρίς ενσωματωμένη μνήμη. Παρατηρούμε τον διπλασιασμό στις απαιτήσεις για Logic Elements καθώς και την μεγάλη αύξηση στον αριθμό των bits για εισόδους και εξόδους όσο και την πολύ μικρή χρήση της συνολικής μνήμης.

Η αύξηση του συνολικού αριθμού των Logic Elements είναι αναμενόμενη λόγω της προσθήκης της επιλογής μεταξύ των 6 πινάκων look-up, ανάλογα με την τιμή της εισόδου που τροφοδοτούμε με τη σταθερά σ_{ph} στο συγκεκριμένο σχεδιασμό.

Πίνακας 9: Χρήση πόρων για την υλοποίηση φίλτρου χωρίς ενσωματωμένη μνήμη

	Εικόνα 100x100		Εικόνα 330x330	
FPGA: Cyclone IV E	Αριθμός Πόρων	Συνολικό Ποσοστό	Αριθμός Πόρων	Συνολικό Ποσοστό
Συνολικά Logic Elements	9.461	8%	9.402	8%
Registers	1600	-	1604	-
I/O (bits)	77	15%	77	15%
Συνολική Μνήμη (bits)	7.840	< 1%	15.744	< 1%
Multipliers 9-Bit	28	5%	28	5%
Συχνότητα: 50MHz				





8.3 Διαφορές των δύο Υλοποιήσεων

Παρακάτω παρουσιάζουμε συνοπτικά τις πιο διακριτές διαφορές των δύο σεναρίων μας. Θα μπορούσαμε γενικά να πούμε ότι το φίλτρο με την ενσωματωμένη μνήμη ήταν το πρωτότυπο ή το πρώτο στάδιο του σχεδιασμού.

Η διαφορά στους πίνακες look-up προέρχεται από την μετατροπή της σταθεράς σ_{ph} σε είσοδο, στο φίλτρο χωρίς την ενσωματωμένη μνήμη. Αυτή η μετατροπή απαιτεί την προσθήκη πινάκων look-up με περιεχόμενο τις φωτομετρικές συνιστώσες και πλήθος ανάλογο με τις διαφορετικές επιλογές θορύβου. Συγκεκριμένα, προσθέσαμε 6 πίνακες look-up, λόγω των 6 διαφορετικών επιλογών θορύβου ανά procedure (ο συνολικός αριθμός τους υπολογίζεται παρακάτω).

Η παραπάνω ιδιαιτερότητα δεν ισχύει στο φίλτρο με την ενσωματωμένη μνήμη καθώς ο πίνακας look-up, εσωτερικά του κάθε procedure, αρχικοποιείται βάσει της σταθεράς σ_{ph} κατά το compilation του σχεδιασμού.

Πίνακας 10: Διαφορές μεταξύ των δύο σχεδιασμών του φίλτρου

Διαφορές φίλτρων βάση Σχεδίασης	Φίλτρο με ενσωματωμένη Μνήμη	Φίλτρο χωρίς ενσωματωμένη Μνήμη
LUT	$(6 \cdot 1) + 2$	$(6 \cdot 6) + 2$
Μνήμη	Αρχικοποιείται και περιλαμβάνεται στο project κατά την σύνθεση	Δέχεται διαφορετικές μνήμες αλλά πρέπει να είναι ίδιο μέγεθος
Εξομοίωση	Κάθε αλλαγή στη σχεδίαση πρέπει να περάσει από σύνθεση	Κατά την εξομοίωση συνδέουμε τις ανάλογες εισόδους και εξόδους με την εξωτερική μνήμη
Θετικά	Γρήγορη Υλοποίηση και Verification	Μεγαλύτερη ευελιξία
Αρνητικά	Κάθε αλλαγή εικόνας και επιπέδου θορύβου απαιτεί εκ νέου σύνθεση και compilation	Απαιτεί μετατροπή της εικόνας σε .hex και προσθήκη στο project πριν τη σύνθεση





Κεφάλαιο 9

Μελλοντικές Προσθήκες και Αλλαγές

Σε αυτό το κεφάλαιο, θα κάνουμε κάποιες προτάσεις για τη μελλοντική βελτίωση ή αλλαγή στο φίλτρο και τα εκτιμώμενα πιθανά αποτελέσματα που μπορεί να προκύψουν. Οι περισσότερες προτάσεις, είναι σχετικά απλές, δεδομένου ότι η δημιουργία του φίλτρου από την αρχή ήταν το πιο δύσκολο κομμάτι.

9.1 Χρήση Παραθύρου διαφορετικού μεγέθους

Η πρόταση για χρήση ενός παράθυρου φίλτρου διαφορετικού μεγέθους από αυτό που υλοποιούμε, προτείνεται σε συνδυασμό με την πρόταση επαναληπτικού φιλτραρίσματος της εικόνας που θα αναφέρουμε πιο αναλυτικά στη συνέχεια. Καθώς το παράθυρό 5x5 επιφέρει το μεγαλύτερο ποσοστό θολώματος έναντι καλύτερης αποθορυβοποίησης σε σχέση με το παράθυρο 3x3, που επιφέρει μικρότερο θόλωμα αλλά και μικρότερη αποθορυβοποίηση, παρατηρούμε πως το μικρότερο παράθυρο απαιτεί αρκετά λιγότερους πόρους για την υλοποίησή του.

Εκτιμούμε ότι με τη χρήση σχεδόν ίδιου αριθμού από πόρους, 8 procedures για το παράθυρο 3x3 έναντι 6 procedures για το παράθυρο 5x5, θα μας έδινε ταχύτερα αποτελέσματα (1 φιλτραρισμένο pixel/1 κύκλο ρολογιού) καθώς το μικρότερο παράθυρο απαιτεί τον υπολογισμό 8 συνιστωσών σε σχέση με τις 24 συνιστώσες του μεγαλύτερου παραθύρου. Συνεπώς το συμπέρασμα που αρχικά κρατάμε είναι πως μπορούμε να φιλτράρουμε μία εικόνα 4 φορές πιο γρήγορα με kernel 3x3 σε σχέση με ένα kernel 5x5.



Εικόνα με θόρυβο



Kernel 3x3



Kernel 5x5

Εικόνα 33: Αποτελέσματα αποθορυβοποίησης για θόρυβο $\sigma=40$



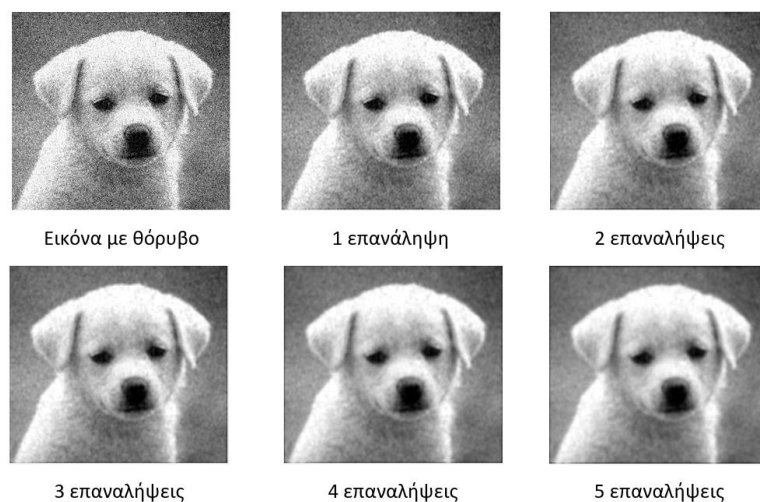


9.2 Επαναληπτικό φιλτράρισμα με kernel 3x3

Καθώς η χρήση παραθύρου 3x3 είναι 4 φορές πιο γρήγορη από τη χρήση του παραθύρου 5x5, δημιουργήσαμε στο MATLAB ένα σενάριο το οποίο πραγματοποιεί μία σειρά από επαναλήψεις, στο τέλος κάθε επανάληψης εισάγεται για φιλτράρισμα η τελική εικόνα του προηγούμενου φιλτραρίσματος. Αν και το παράθυρο 3x3 δεν επέφερε στην αρχή κάποιο θόλωμα, μετά από κάποιες επαναλήψεις και ανάλογα τον θόρυβο της εικόνας, παρατηρήσαμε ότι το φαινόμενο εμφανιζόταν. Σε εικόνες με περισσότερο θόρυβο η επαναληπτική διαδικασία είχε καλύτερα αποτελέσματα.



Εικόνα 34: Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=15$ και kernel 3x3



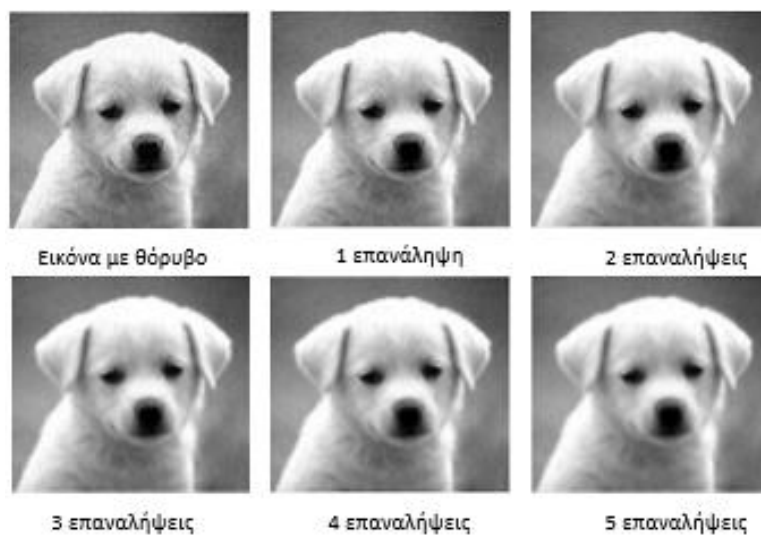
Εικόνα 35: Χρήση επαναληπτικού φιλτραρίσματος για $\sigma_{noise}=35$ και kernel 3x3



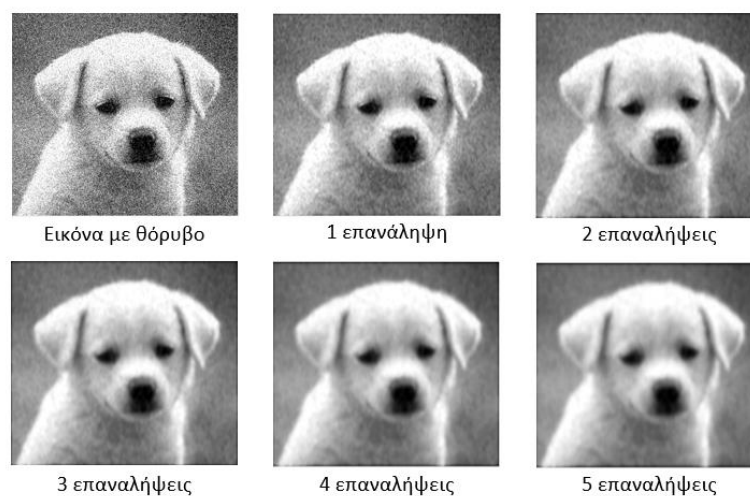


9.3 Επαναληπτικό φιλτράρισμα με kernel 5x5

Για να έχουμε πλήρη εικόνα της επαναληπτικής μεθόδου, δημιουργήσαμε ένα ανάλογο σενάριο για το διμερές μας φίλτρο με παράθυρο 5x5. Αν και η επαναληπτική μέθοδος καταφέρνει να αυξήσει κατά πολύ τη μετρική *MSSIM*, είναι αρκετά εμφανές το θόλωμα αλλά και η σταδιακή μείωση της μετρικής *PSNR*.



Εικόνα 36: Χρήση επαναληπτικού φιλτραρίσματος για $\sigma=15$ και kernel 5x5

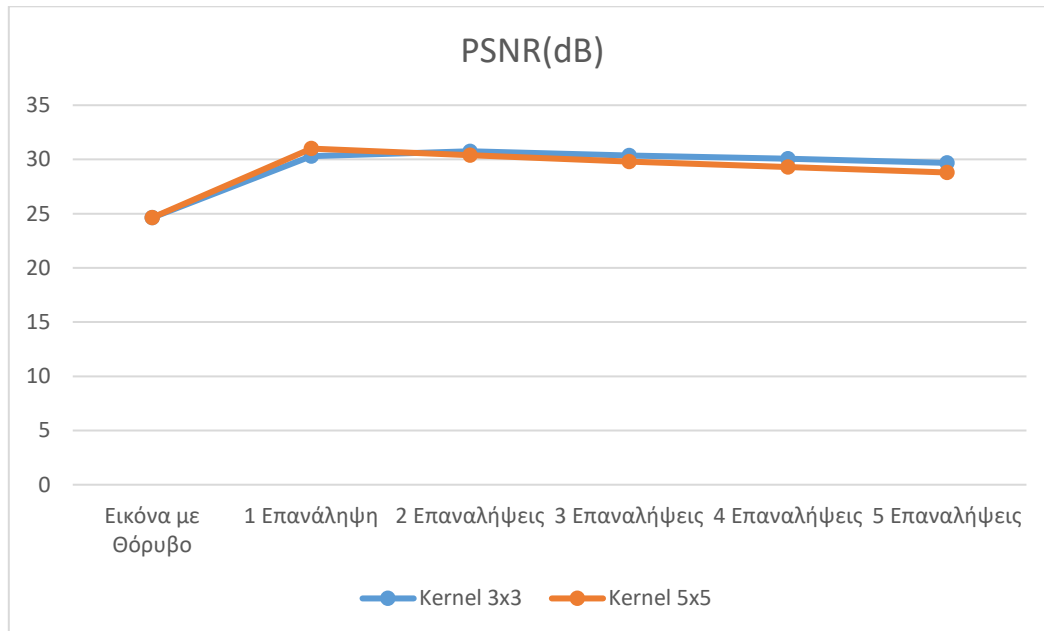


Εικόνα 37: Χρήση επαναληπτικού φιλτραρίσματος για $\sigma=35$ και kernel 5x5

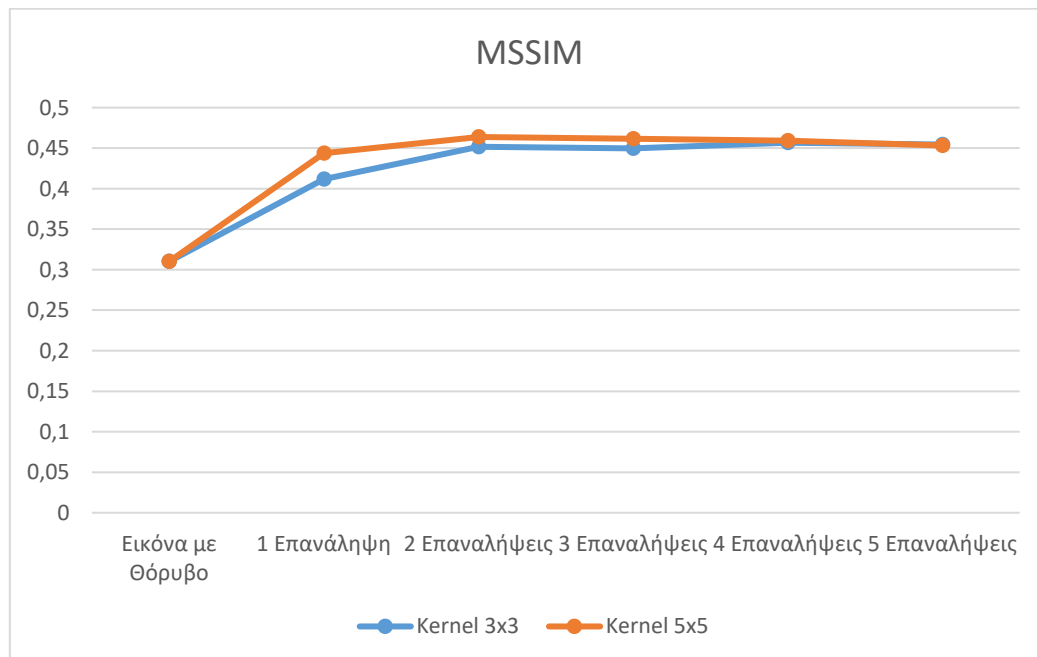




9.4 Μετρήσεις επαναληπτικού φιλτραρίσματος για θόρυβο $\sigma_{noise}=15$



Γράφημα 11: PSNR ανά επανάληψη για θόρυβο $\sigma_{noise}=15$

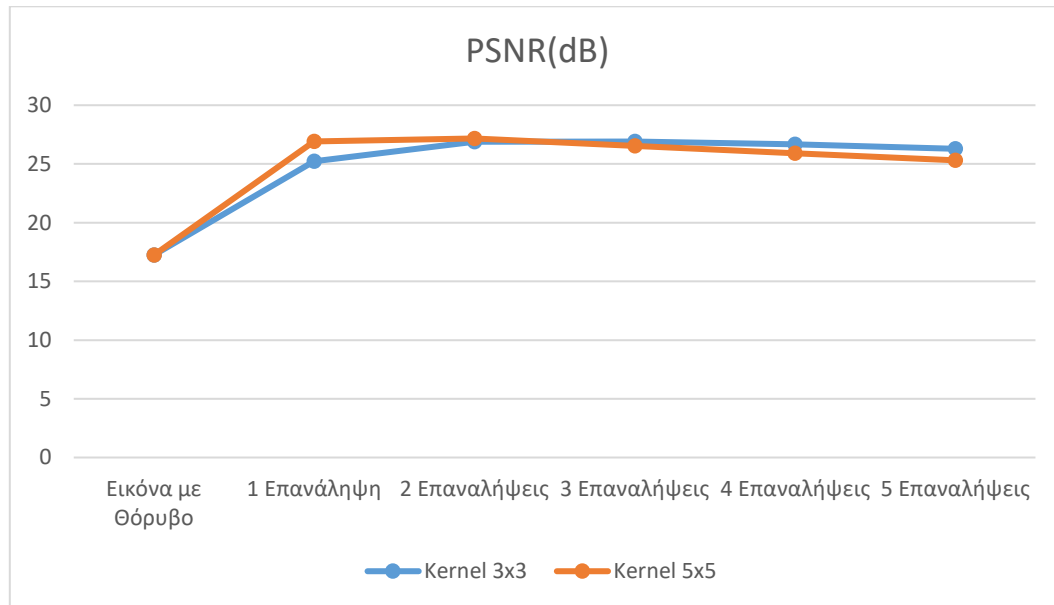


Γράφημα 12: MSSIM ανά επανάληψη για θόρυβο $\sigma_{noise}=15$

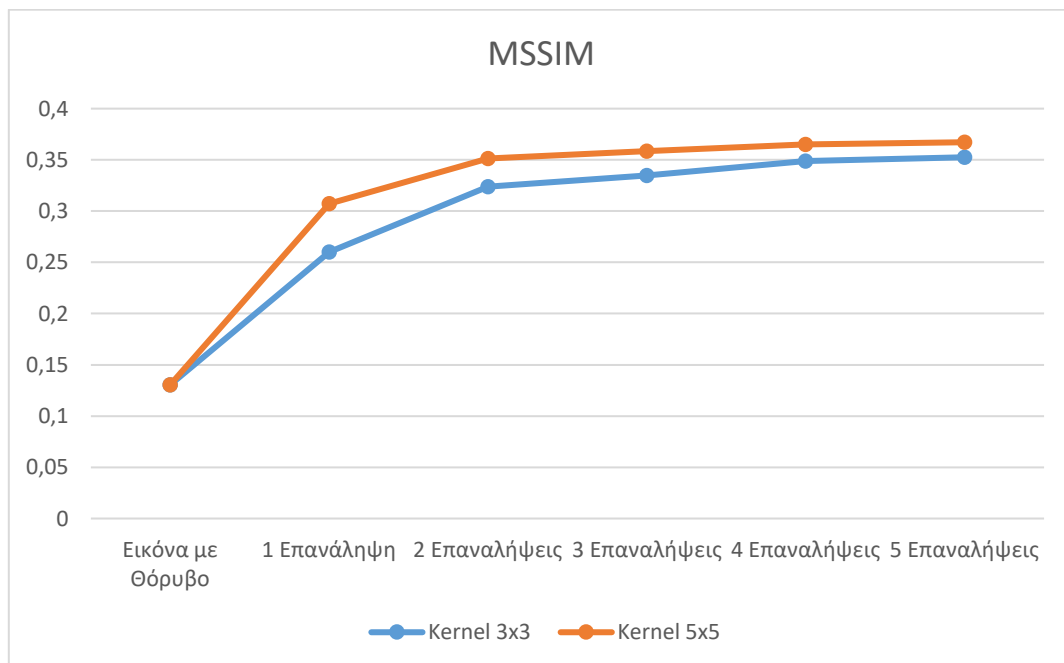




9.5 Μετρήσεις επαναληπτικού φιλτραρίσματος για θόρυβο σnoise=35



Γράφημα 13: PSNR ανά επανάληψη για θόρυβο σnoise=35



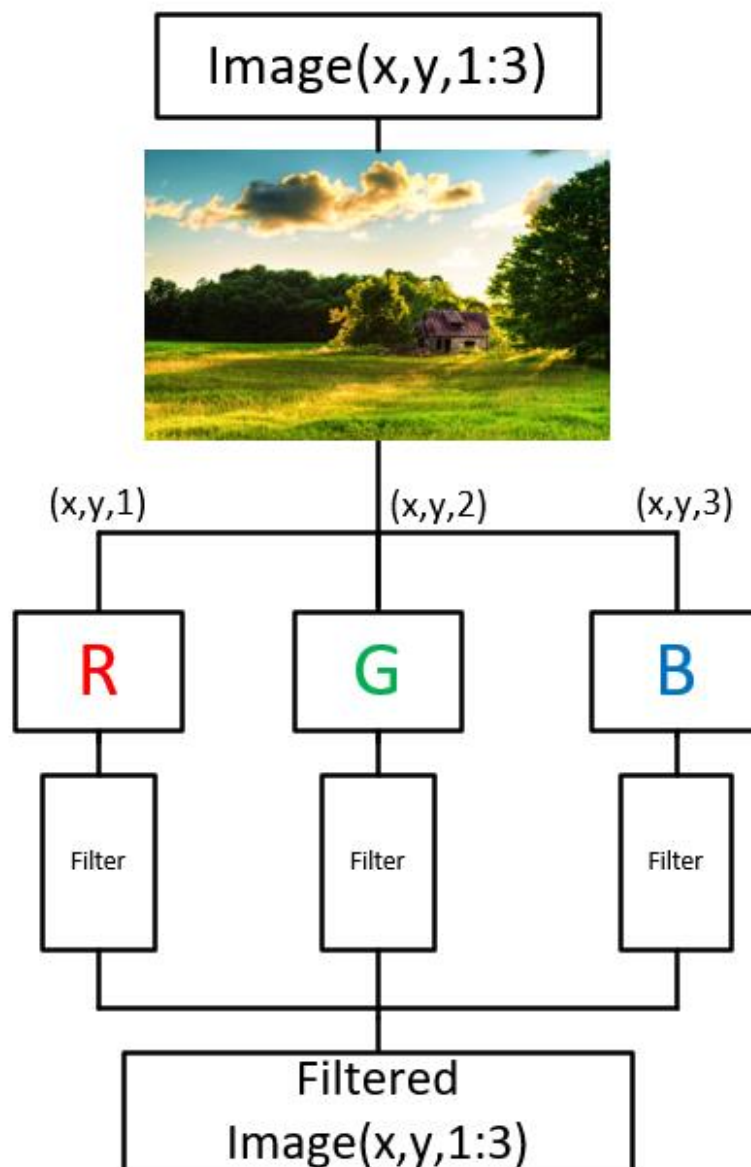
Γράφημα 14: MSSIM ανά επανάληψη για θόρυβο σnoise=35





9.6 Φιλτράρισμα έγχρωμων εικόνων RGB

Αφού σχεδιάστηκε το φίλτρο για grayscale εικόνες και τα αποτελέσματα του είναι ικανοποιητικά, το επόμενο βήμα είναι να σχεδιαστεί μία εκδοχή του, η οποία να φιλτράρει έγχρωμες εικόνες. Οι έγχρωμες εικόνες RGB, σε σχέση με τις αντίστοιχες grayscale, αποτελούνται από 3 διαστάτους πίνακες, έναν για κάθε βασικό πίνακα δεδομένου ότι το όνομα RGB αναφέρεται στον πίνακα του κόκκινου (Red), πράσινου (Green) και μπλε (Blue), από τον συνδυασμό των οποίων προκύπτουν όλα τα υπόλοιπα χρώματα.

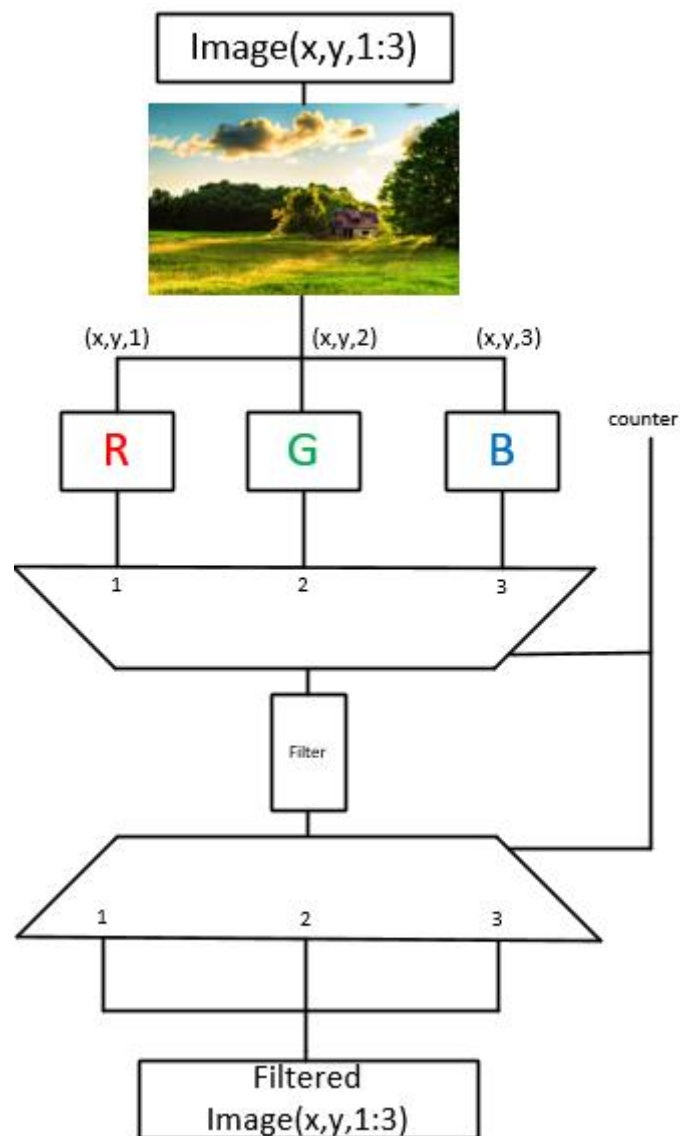


Εικόνα 38: Παράλληλη αρχιτεκτονική φίλτρου για RGB εικόνες





Για την υλοποίηση του φιλτραρίσματος σε έγχρωμες εικόνες RGB απαιτείται ο τριπλάσιος αριθμός πόρων, ώστε για κάθε δισδιάστατο πίνακα να φιλτράρεται ένα pixel τη φορά. Έτσι θα φιλτράρουμε όλα τα pixels της έγχρωμης εικόνας στον ίδιο χρόνο με τη grayscale εκδοχή της. Δεδομένης της διαφοράς των δύο τύπων εικόνας, αν η χρονική καθυστέρηση δεν μας απασχολεί, τότε μπορούμε να χρησιμοποιήσουμε το ένα τρίτο των πόρων και να φιλτράρουμε τον κάθε δισδιάστατο πίνακα μετά τον άλλο (σειριακά), συνολικά στον τριπλάσιο χρόνο.



Εικόνα 39: Σειριακή αρχιτεκτονική φίλτρου για RGB εικόνες





Βιβλιογραφία

- [1] J. Giraldo, Z. Kelm, L. Yu, J. Fletcher, B. Erickson, and C. McCollough, "Comparative study of two image space noise reduction methods for computed tomography: Bilateral filter and nonlocal means," in Proc. Conf. IEEE EMBS, 2009, pp. 3529–3532.
- [2] Z. Wand, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Process, vol. 13, no. 4, pp. 600–612, Apr. 2014.
- [3] Anna Gabiger-Rose, IEEE, Matthias Kube, Robert Weigel and Richard Rose, "An FPGA-Based Fully Synchronized Design of a Bilateral Filter for Real-Time Image Denoising", IEEE Trans. Industrial Electronics, vol. 61, no. 8, Aug. 2014.
- [4] Chandrajit Pal, Avik Kotal, Asit Samanta, Amlan Chakrabarti, Ranjan Ghosh, "Design space exploration for image processing architectures on FPGA targets", University College of Science, Technology and Agriculture, University of Calcutta, 92, APC Road, Kolkata, India, pp. 1-19.
- [5] "DSP Builder Handbook, Volume 1: Introduction to DSP Builder", Altera Corporation, Nov. 2013.
- [6] "DSP Builder Handbook, Volume 2: DSP Builder Standard Blockset", Altera Corporation, Aug. 2016.

