ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

Υλοποίηση Αλγόριθμου Ανίχνευσης Ακμών σε προγραμματιζόμενη ψηφίδα Xilinx

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Χ. Αναστασιάδης

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

Υλοποίηση Αλγόριθμου Ανίχνευσης Ακμών σε προγραμματιζόμενη ψηφίδα Xilinx

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Χ. Αναστασιάδης

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την .....

.

……..	……..	……..
Κ. Πεκμεστζή	Δ. Σούντρης	Γ. Οικονομάκος
Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2009

Αναστασιάδης Χ. Νικόλαος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Μέσα από αυτή την σελίδα θέλω να ευχαριστήσω όσους με βοήθησαν να φτάσω ως εδώ, ξεκινώντας από τους γονείς μου που πάντα με στήριξαν με αγάπη από τα παιδικά μου χρόνια, και συνεχίζουν να το κάνουν μέχρι και σήμερα με περίσσεια υπομονή.

Επίσης τους δασκάλους μου από τα σχολικά μέχρι και τα ακαδημαϊκά μου χρόνια για την γνώση και τα κίνητρα που μου έδωσαν.

Ένα μεγάλο ευχαριστώ οφείλω φυσικά στον επιβλέπων της διπλωματικής αυτής, κ. Κιαμάλ Πεκμεστζή, καθηγητή της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, και στον κ. Ισίδωρο Σιδέρη, Διδάκτορα της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ του Εθνικού Μετσόβιου Πολυτεχνείου για την πολύτιμη βοήθεια και καθοδήγησή τους κατά την εκπόνηση της διπλωματικής.

Τέλος θα κλείσω με ένα μεγάλο ευχαριστώ στους φίλους μου, σε όλους αυτούς τους ξεχωριστούς ανθρώπους που όλα αυτά τα χρόνια με στηρίζουν και με ανέχονται.

Περίληψη

Ο σκοπός αυτής της διπλωματικής εργασίας ήταν η υλοποίηση ενός αλγόριθμου ανίχνευσης ακμών χρησιμοποιώντας την προγραμματιζόμενη ψηφίδα Xilinx Spartan 3E. Μελετήθηκαν θεωρητικά πολλοί αλγόριθμοι ανίχνευσης ακμών με χρήση του περιβάλλοντος Matlab και μετά από αξιολόγηση επιλέχθηκαν σαν κατάλληλα για υλοποίηση σε ενσωματωμένο σύστημα το Laplacian of Gaussian φίλτρο και η Bi-level προσέγγιση του.

Αρχικά το σύστημα υλοποιήθηκε αποκλειστικά σε λογισμικό και μετρήθηκε η επίδοσή του για τους δύο επιλεγμένους αλγόριθμους. Στην συνέχεια υλοποιήθηκε συνεπεξεργαστής για την επιτάχυνση του συστήματος. Ο συνεπεξεργαστής αυτός περιλαμβάνει μονοπάτι δεδομένων που περατώνει τους υπολογισμούς για το Laplacian-of-Gaussian φίλτρο ή για την Bilevel προσέγγιση του, και μηχανή καταστάσεων για τον έλεγχό του. Επίσης υλοποιήθηκε η επικοινωνία του συνεπεξεργαστή με τον δίαυλο Fast Simplex Link που χρησιμοποιήθηκε για την επικοινωνία με τον κύριο επεξεργαστή του συστήματος, τον Microblaze.

Τα δύο διαφορετικά φίλτρα που υλοποιήθηκαν συγκρίθηκαν ως προς την επίδοση και την επιφάνεια που καταλαμβάνουν στην προγραμματιζόμενη ψηφίδα με το προσεγγιστικό φίλτρο Bilevel Laplacian of Gaussian να υπερέχει και στους δύο τομείς. Επίσης έγινε σύγκριση επίδοσης μεταξύ των υλοποιήσεων σε λογισμικό και υλικό, με την δεύτερη περίπτωση φυσικά να υπερέχει σε ταχύτητα.

Τα κυκλώματα που υλοποιούν τα δύο φίλτρα υλοποιήθηκαν και σε ASIC τεχνολογία στο περιβάλλον Synopsys και έγινε σύγκριση του κρίσιμου μονοπατιού, της κατανάλωσης ενέργειας και της κάλυψης που έχει το κάθε κύκλωμα στην ψηφίδα, όπου και πάλι το bilevel laplacian of gaussian επιβεβαιώνει την υπεροχή του.

Λέξεις κλειδιά

Ενσωματωμένα συστήματα, Laplacian of Gaussian, LoG, Bilevel Laplacian of Gaussian, BLoG, Xilinx EDK, Xilinx Spartan 3E, Fast Simplex Link, FSL, Xilinx ISE, Ανίχνευση ακμών, Επιταχυντές υλικού, ASIC, Synopsys.

Abstract

Aim of this diploma thesis was to implement and edge detection algorithm using the field programmable gate array Xilinx Spartan 3E. Edge detection algorithms were studied using the Matlab programming environment and after assessment of the algorithms the Laplacian of Gaussian filter and its Bi-level approximation were selected as suitable to implement on the FPGA as embedded systems.

Initially the system was built exclusively on software for both algorithms, and its speed performance was measured. Then a hardware coprocessor was build to accelerate the system. This coprocessor includes a datapath to calculate the output for both the laplacian of Gaussian filter and its approximation, a finite state machine to control the datapath and the communication between the main processor (Microblaze) and the implemented coprocessor. The communication is archived over the Fast simplex Link bus, which is dedicated to the main and the co-processor.

The two different filter circuits were compared in terms of speed and area coverage on the FPGA with the Bilinear approximation surpassing in both fields. The speed performance between the hardware and software implementations was compared with the hardware performing more than 3 times faster.

Both filter circuits were implemented using the ASIC workflow under the Synopsys environment and the area coverage, the power consumption and the critical path were calculated. In every field the bi-level laplacian of Gaussian filter proves its superiority.

Keywords

Embedded Systems, Laplacian of Gaussian, LoG, Bilevel Laplacian of Gaussian, BLoG, Xilinx EDK, Xilinx Spartan 3E, Fast Simplex Link, FSL, Xilinx ISE, Edge Detection, Hardware acceleration, ASIC, Synopsys.

Περιεχόμενα

Ευρετήριο Εικόνων

Υπόβαθρο ανίχνευσης ακμών

Εισαγωγή

Με τον όρο ακμές για μια ασπρόμαυρη εικόνα, αναφερόμαστε σε αλλαγές της φωτεινότητας μεταξύ γειτονικών περιοχών της. Αλλαγές της φωτεινότητας συνήθως αντιστοιχούν σε διαφοροποίηση ιδιοτήτων της απεικόνισης τρισδιάστατων αντικειμένων όπως αλλαγές της υφής, του βάθους, όρια αντικειμένων, διαφορετικό φωτισμό και αντανάκλαση. Έτσι με την ανίχνευση ακμών μπορούμε να αντλήσουμε πληροφορίες για φυσικές ιδιότητες για τα εικονιζόμενα πραγματικά αντικείμενα.

Η ανίχνευση ακμών μιας εικόνας παρουσιάζει αρκετές δυσκολίες. Οι ακμές μπορεί να χαρακτηρίζονται από προοδευτικές ή ακόμα και πολύ μικρές αλλαγές στην φωτεινότητα της εικόνας. Η παρουσία θορύβου σε μια εικόνα μπορεί να οδηγήσει στην ανίχνευση εσφαλμένων ακμών αλλοιώνοντας τα όρια των αντικειμένων. Ο διαφορετικός φωτισμός και η σκίαση μπορεί να ανιχνευτούν σαν ψευδοακμές ενώ δεν αντιστοιχούν σε φυσική ακμή. Ακόμα και αντικείμενα διαφορετικής κλίμακας πιθανό να βρίσκονται στην ίδια εικόνα.

Σε συστήματα βιολογικής όρασης υπάρχουν νευροβιολογικές και ψυχοφυσικές ενδείξεις ότι στα πρώτα στάδια επεξεργασίας της οπτικής πληροφορίας γίνεται κάποιο είδος ανίχνευσης ακμών. Αυτή η επεξεργασία μοιάζει με ζωνοπερατά επιλεκτικά φίλτρα ή ισοδύναμα με συνέλιξη της οπτικής πληροφορίας με νευρικές αποκρίσεις. Αυτά τα φίλτρα έχουν μοντελοποιηθεί με κάποιες διαφορές από Gabor ή Gaussian φίλτρα.

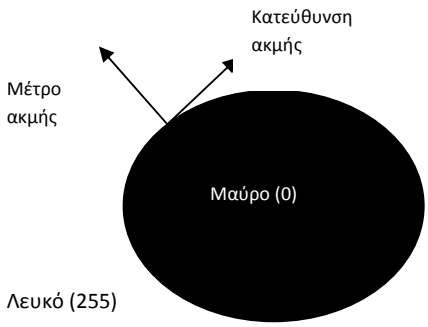
Η ανίχνευση ακμών αποτελεί την βάση για μετέπειτα επεξεργασία μια εικόνας ή ακολουθίας εικόνων με αλγορίθμους υπολογιστικής όρασης, όπως ανάλυση υφής, τμηματοποίησης, ανίχνευσης κίνησης, στερέοψης και αναγνώρισης προτύπων. Γι’ αυτό πρέπει να δίνει αξιόπιστα αποτελέσματα και να υλοποιείται αποδοτικά.

Αναφορές

1. Σημειώσεις, Όραση Υπολογιστών, Πέτρος Μαραγκός, Ε.Μ.Π. 2005 και 2009.

Τύποι και χαρακτηριστικά ακμών

Υπολογιστικά οι ακμές (αλλαγές στην συνάρτηση της έντασης) για συνεχείς συναρτήσεις μπορούν να υπολογιστούν με τον υπολογισμό της πρώτης παραγώγου και εντοπισμό των τοπικών μέγιστων. Μια δεύτερη μέθοδος με πλεονεκτήματα σε αξιοπιστία στηρίζεται στις διελεύσεις της δεύτερης παραγώγου από το μηδέν (zero crossing). Φυσικά επειδή έχουμε συναρτήσεις δύο μεταβλητών (x,y συντεταγμένη) θα υπολογίζουμε τις μερικές παραγώγους. Μια μεταβολή της συνάρτησης της εικόνας μπορεί να περιγραφεί με την βάθμωση (gradient) προς την κατεύθυνση της μέγιστης μεταβολής. Μια ακμή είναι ιδιότητα του κάθε εικονοστοιχείου ξεχωριστά και υπολογίζεται από την συμπεριφορά της συνάρτησης της εικόνας σε μια περιοχή γειτονικών εικονοστοιχείων. Πρόκειται για διανυσματική μεταβλητή με μέτρο και κατεύθυνση [εικόνα 1.1].



Εικόνα 1. Κατεύθυνση και μέτρο ακμής

Το μέτρο της ακμής μας δείχνει πόσο μεγάλη είναι μεταβολή της συνάρτησης φωτεινότητας (ισχυρή, αδύναμη ακμή) και η κατεύθυνση μας δίνει τον προσανατολισμό της ακμής στην εικόνα, και υπολογίζονται ως εξής.

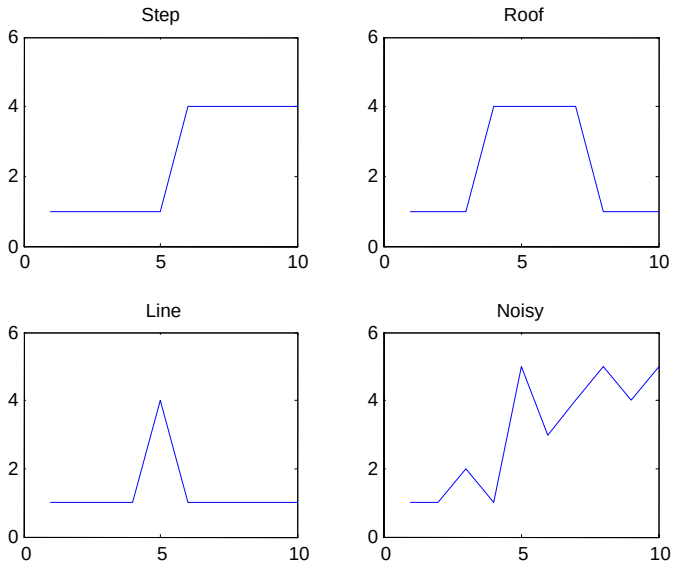
Για το μέτρο της ακμής,

$$|grad(I(x,y)| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

Και για την κατεύθυνση της ακμής

$$\varphi = \arg\left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

Τέλος υπάρχουν διάφορα είδη ακμών. Μερικά από αυτά εικονίζονται στην εικόνα 2 που ακολουθεί στην επόμενη σελίδα.



Εικόνα 2. Είδη ακμών σε grayscale εικόνες.

Η ακμή τύπου στέγης ανταποκρίνεται σε λωρίδες ίδιας έντασης στην εικόνα, και η ακμή τύπου γραμμής αναφέρεται σε μικρότερο εύρος. Η βηματική ακμή είναι η διαχωριστική επιφάνεια δύο αντικειμένων ή ενός αντικειμένου και του περιβάλλοντα χώρου. Η θορυβώδης ακμή είναι μια βηματική ακμή αλλά με τα εικονοστοιχεία να λαμβάνουν ανομοιόμορφες τιμές φωτεινότητας κατά τη μετάβαση μεταξύ των δύο επιπέδων.

Όταν δεν μας ενδιαφέρει η κατεύθυνση παρά μόνο το μέτρο των ακμών τότε με ανιχνεύση των διελεύσεων της δεύτερης παραγώγου από το μηδέν επιτυγχάνουμε καλύτερα αποτελέσματα σε αξιοπιστία και υπολογιστικό κόστος. Ο υπολογισμός της δεύτερης παραγώγου επιτυγχάνεται χρησιμοποιώντας μικρά μητρώα συνέλιξης που λειτουργούν σαν ψηφιακοί πυρήνες λαπλασιανών φίλτρων. Υπολογίζουμε δηλαδή,

$$Laplacian = \nabla^2 \mathrm{I}(\mathrm{x}, \mathrm{y})$$

Οι διάφοροι ανιχνευτές ακμών συνήθως σχεδιάζονται και είναι αποτελεσματικοί για ένα είδος ακμών. Στην συνέχεια της ανάλυσής μας θα ασχοληθούμε με τις βηματικές ακμές που είναι οι πιο συνηθισμένες και προσφέρουν τις περισσότερες πληροφορίες για μια εικόνα.

Αναφορές

1. Σημειώσεις, Όραση Υπολογιστών, Πέτρος Μαραγκός, *Ε.Μ.Π. 2005 και 2009.*
2. Image processing, analysis, and machine vision. Milan Sonka, Vaclav Hlavac, Roger Boyle. *Pacific Grove, Calif. : PWS Publishing, cop. 1999.*

Γραμμικοί τελεστές ανίχνευσης ακμών προσεγγίζοντας 1η παράγωγο

Ιστορικά η πρώτη απόπειρα ανίχνευσης ακμών, που διήρκεσε περίπου 30 χρόνια (δεκαετία 50 έως δεκαετία 70), έγινε υπολογίζοντας διακριτές προσεγγίσεις των μερικών παραγώγων κατά κατεύθυνση για την υπό επεξεργασία εικόνα. Αυτό γίνεται με την συνέλιξη της εικόνας και ενός μικρού μητρώου που στόχο έχει να ενισχύσει την ένταση των ακμών. Το πιο παλιό από αυτά τα μητρώα προτάθηκε από τον Roberts.

Τελεστές Roberts

Τα μητρώα που προτείνει ο Roberts για τον υπολογισμό της πρώτης παραγώγου της συνάρτησης φωτεινότητας της εικόνας είναι τα εξής:

$$R_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, R_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Για μια εικόνα:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Τα μητρώο R_1 συνελισσόμενο με την εικόνα δίνει στην έξοδο:

$$\begin{bmatrix} (-1)a_{11} & (0)a_{12} & a_{13} \\ (0)a_{21} & (1)a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \overset{*R_1}{\Longrightarrow} \begin{bmatrix} a_{22} - a_{11} & a_{22} - a_{13} & .. \\ a_{32} - a_{21} & a_{33} - a_{22} & .. \\ .. & .. & .. \end{bmatrix}$$

Αντίστοιχα για το μητρώο R_2 παίρνουμε:

$$\begin{bmatrix} (0)a_{11} & (-1)a_{12} & a_{13} \\ (1)a_{21} & (0)a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \overset{*R_2}{\Longrightarrow} \begin{bmatrix} a_{21} - a_{12} & a_{22} - a_{13} & .. \\ a_{31} - a_{22} & a_{32} - a_{23} & .. \\ .. & .. & .. \end{bmatrix}$$

Τώρα με χρήση κάποιας νόρμας μπορούμε να υπολογίσουμε το μέτρο των ακμών και με χρήση κατωφλίωσης να αποφανθούμε για τις ακμές της εικόνας. Οι πιο συνηθισμένες νόρμες που χρησιμοποιούνται είναι οι εξής:

$$\sqrt{f_x^2 + f_y^2} \qquad (1)$$

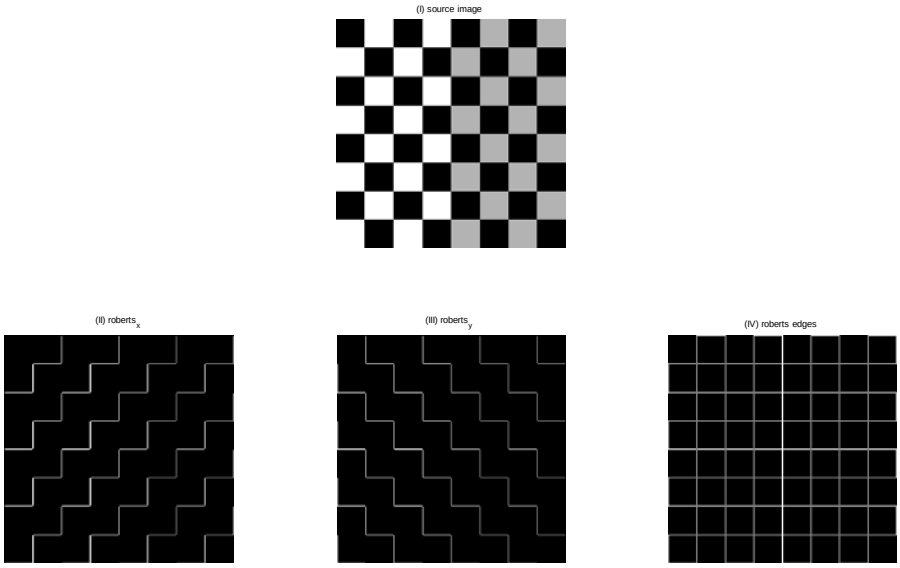
$$|f_x| + |f_y| \qquad (2)$$

$$\max\left(|f_x|, |f_y|\right) \quad (3)$$

Με χρήσης της νόρμας 2 για παράδειγμα προκύπτει ο πίνακας του μέτρου των ακμών. Τα στοιχεία του υπολογίζονται ως εξής:

$$Edge_{i,j} = |I(i,j) - I(i+1,j+1)| + |I(i,j+1) + I(i+1,j)|$$

Μετά τον υπολογισμό του μέτρου της ακμής με την κατάλληλη νόρμα, με την τεχνική της κατωφλίωσης ανιχνεύουμε τα τοπικά μέγιστα της φωτεινότητας της εικόνας και αποφασίζουμε τι θα δεχθούμε ως ακμές. Η κατωφλίωση θα οδηγήσει τα εικονοστοιχεία με τιμή έντασης μικρότερη από το κατώφλι στην δυαδική τιμή '0' και αυτά με μεγαλύτερες τιμές στην δυαδική τιμή '1'(εικονοστοιχείο ακμής). Το αποτέλεσμα του αλγόριθμου φαίνεται στην εικόνα 1.3 που ακολουθεί για μια πολύ απλή εικόνα εισόδου, μια σκακιέρα.



Εικόνα 3. Εφαρμογή του τελεστή Roberts.

Η εικόνα (I) είναι η αρχική μας εικόνα προς επεξεργασία. Στις (II) και (III) βλέπουμε το αποτέλεσμα της συνέλιξης με τους δύο τελεστές Roberts. Στην ουσία αυτό που κάνουν οι δύο τελεστές είναι να ενισχύουν τις ακμές της εικόνας κατά τις κατευθύνσεις 45^o και 135^o. Στην εικόνα (IV) βλέπουμε το τελικό αποτέλεσμα του αλγόριθμου χρησιμοποιώντας μια από τις νόρμες που προαναφέρθηκαν για τον υπολογισμό του μέτρου της ακμής και τέλος εφαρμόζοντας το κατώφλι που επιλέγουμε για της επιλογή των περιοχών που συνιστούν ακμή.

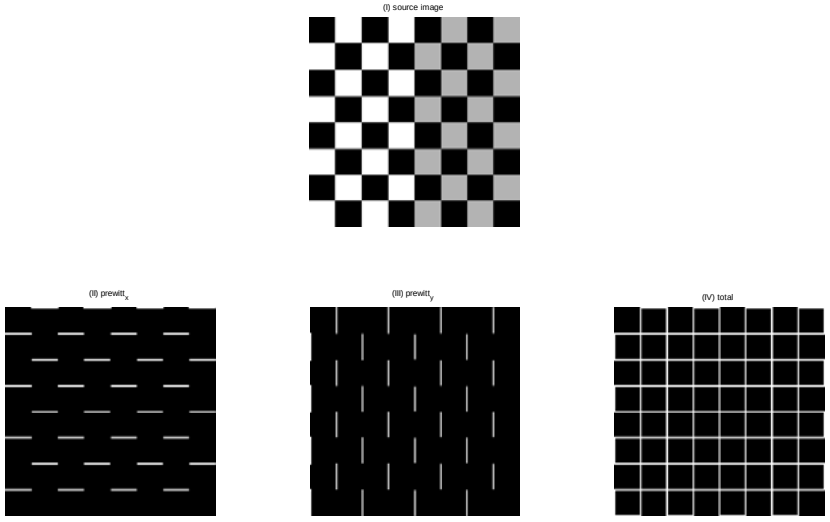
Τελεστές Prewitt

Οι τελεστές prewitt προσεγγίζουν την μερική παράγωγο πρώτης τάξης κατά κατεύθυνση για την εικόνα. Υπάρχουν 8 διαφορετικές κατευθύνσεις για τις οποίες μπορούμε να υπολογίσουμε την μερική παράγωγο, δύο όμως αρκούν για να εντοπίσουμε τις ακμές στην περίπτωση που μας ενδιαφέρει μόνο το μέτρο της ακμής.

$$P_{1=}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, P_{1=}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Οι νόρμες για τις υπόλοιπες κατευθύνσεις μπορούν να προκύψουν με απλή περιστροφή των περιφερειακών στοιχείων της P_1.

Η διαδικασία εντοπισμού των ακμών παραμένει ίδια με αυτή για τον τελεστή Roberts και τα αποτελέσματα ακολουθούν στην εικόνα 1.4

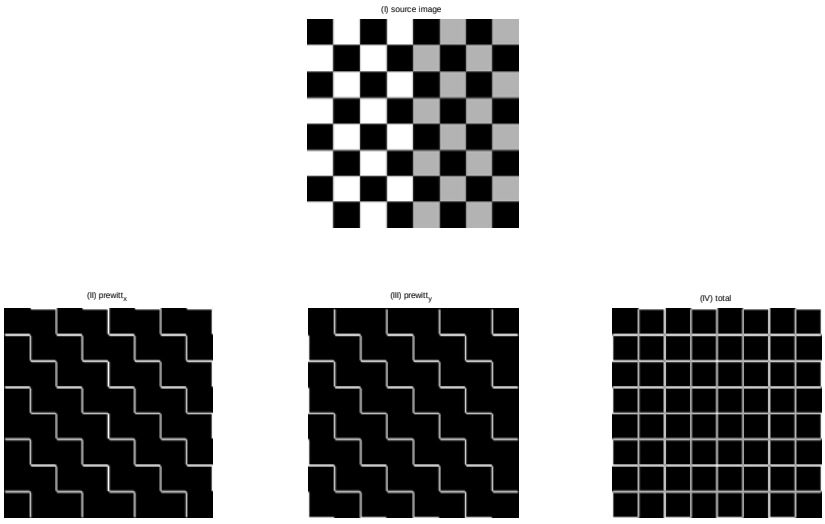


Εικόνα 4. Εφαρμογή τελεστή Prewitt.

Τα ενδιάμεσα αποτελέσματα συνέλιξης της εικόνας με τους δύο τελεστές δίνουν διαφορετικά αποτελέσματα σε σχέση με αυτά που πήραμε από τους τελεστές Roberts όμως το τελικό αποτέλεσμα των ακμών είναι το ίδιο. Αυτό συμβαίνει γιατί τα μητρώα Roberts που χρησιμοποιήσαμε προηγούμενα ενισχύουν τις ακμές της εικόνας κατά διαφορετική κατεύθυνση απ' ότι γίνεται με τους τελεστές Prewitt που εδώ ενισχύουν τις ακμές στις κατευθύνσεις 0^o και 90^o .

Κάτι αντίστοιχο θα συνέβαινε αν χρησιμοποιούσαμε δυο άλλα μητρώα prewitt που προκύπτουν με απλή περιστροφή αυτών που δώσαμε παραπάνω. Προϋπόθεση είναι όμως τα δυο μητρώα να είναι κάθετα μεταξύ τους. Για παράδειγμα

$$P_{3=}\begin{bmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{bmatrix}, P_{4=}\begin{bmatrix}0 & -1 & -1\\ 1 & 0 & -1\\ 1 & 1 & 0\end{bmatrix}$$



Εικόνα 5. Εφαρμογή τροποποιημένου τελεστή Prewitt.

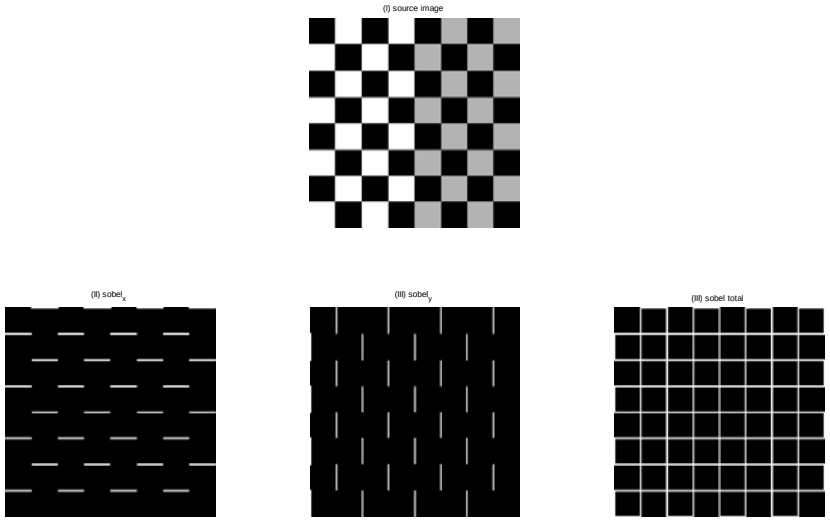
Το τελικό αποτέλεσμα για τις ακμές είναι ακριβώς το ίδιο με τις δύο προηγούμενες μήτρες που χρησιμοποιήθηκαν, κάτι αναμενόμενο αφού το μέτρο των ακμών της εικόνας παραμένει ίδιο.

Τελεστές Sobel

Και οι τελεστές Sobel, όπως και οι επόμενοι που θα αναφέρουμε, προσεγγίζουν την πρώτη μερική παράγωγο κατά κατεύθυνση. Και αυτά τα μητρώα συνέλιξης (convolution kernels) είναι τρία επί τρία, και η διαδικασία για την ανίχνευση των ακμών ίδια με αυτή που χρησιμοποιήθηκε παραπάνω. Και σε αυτή την περίπτωση υπάρχουν οκτώ διαφορετικές κατευθύνσεις που μπορούμε να ανιχνεύσουμε ακμές. Δύο από αυτά τα μητρώα συνέλιξης είναι:

$$S_{1=}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}, S_{2=}\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}$$

Και η εφαρμογή τους στην εικόνα δίνει τα ακόλουθα αποτελέσματα που παρατίθενται στην εικόνα 6.

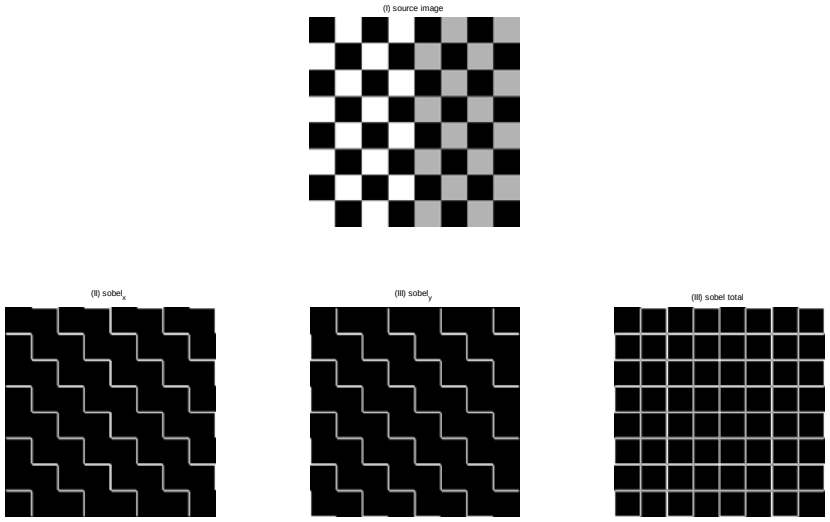


Εικόνα 6. Εφαρμογή τελεστή Sobel

Και περιστρέφοντας τα στοιχεία των παραπάνω μητρώων παίρνουμε τα δυο εναλλακτικά

$$S_{3=}\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, S_{4=}\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

Το αποτέλεσμα για το μέτρο των ακμών όμως και πάλι είναι το ίδιο όπως περιμέναμε.



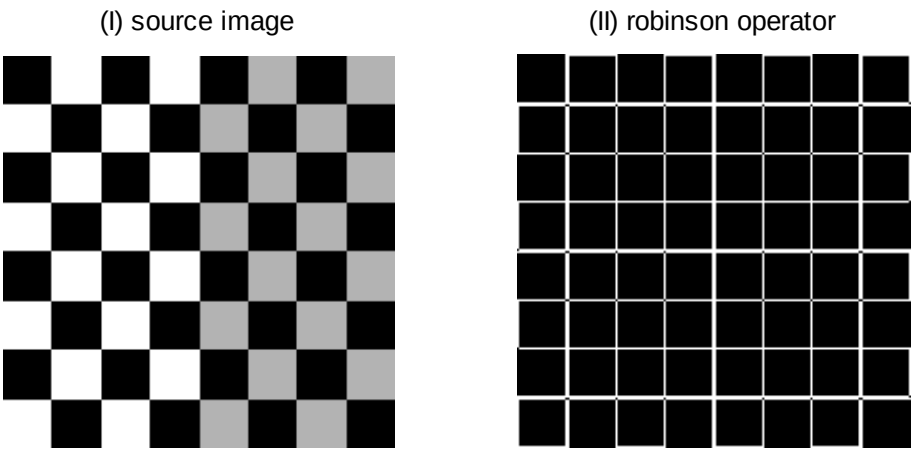
Εικόνα 7. Εφαρμογή τροποποιημένου τελεστή Sobel

Τελεστές Kirch, Robinson

Και οι τελεστές Kirch και Robinson προσεγγίζουν την πρώτη παράγωγο. Τα μητρώα τους επίσης υπολογίζουν κατευθυντικές παραγώγους και έχουν τις ίδιες ιδιότητες με αυτές που έχουμε προαναφέρει. Οι πυρήνες τους είναι οι ακόλουθοι

$$K = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}$$

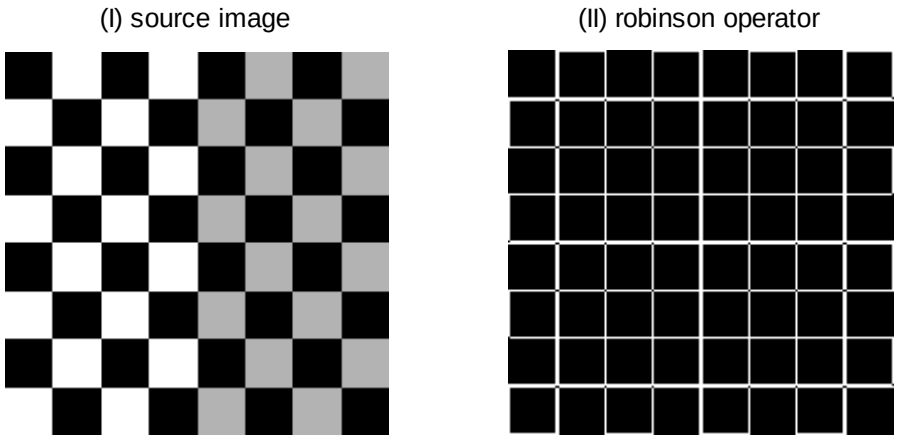
Η έξοδος που παίρνουμε με την εφαρμογή του είναι



Εικόνα 8. Εφαρμογή τελεστή Kirch

Και αντίστοιχα για τον robinson convolution kernel:

$$R = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix}$$



Εικόνα 9. Εφαρμογή τελεστή Robinson

Σχόλια για τους τελεστές προσέγγισης της πρώτης παραγώγου

Εκτός του τελεστή robinson όλοι οι άλλοι έχουν διαστάσεις 3 επί 3. Παρά το μικρό τους μέγεθος εισάγουν αρκετά μεγάλη πολυπλοκότητα. Για τον υπολογισμό ενός pixel εξόδου χρειάζονται 6 πολλαπλασιασμοί και 5 προσθέσεις για κάθε μια από τις κατευθύνσεις που υπολογίζουμε την πρώτη παράγωγο. Μια επιπλέον πρόσθεση χρειάζεται για να πάρουμε το τελικό μέτρο της ακμής. Συνολικά 12 πολλαπλασιασμοί και 11 προσθέσεις, για ένα και μόνο εικονοστοιχείο. Φυσικά παραγοντοποιώντας μπορούμε να μειώσουμε τους πολλαπλασιασμούς σε 2 καθώς όλα τα μητρώα έχουν μόλις 2 μη μηδενικές τιμές για τα στοιχεία τους.

Μια επίσης σημαντική παρατήρηση είναι ότι το άθροισμα των στοιχείων του κάθε μητρώου είναι πάντα μηδέν. Έτσι πάντα όταν βρίσκεται σε εσωτερική περιοχή ενός αντικειμένου (φωτεινότητα σταθερή) η έξοδος είναι πάντα μηδέν. Όταν βρεθούμε όμως σε ακμή η έξοδος παίρνει μεγάλες τιμές. Αυτή είναι η ενίσχυση της ακμής και με αυτό τον τρόπο λειτουργούν τα μητρώα συνέλιξης που προσεγγίζουν την πρώτη παράγωγο.

Αναφορές

1. Σημειώσεις, Όραση Υπολογιστών, Πέτρος Μαραγκός, *Ε.Μ.Π. 2005 και 2009.*
2. Image processing, analysis, and machine vision. Milan Sonka, Vaclav Hlavac, Roger Boyle. *Pacific Grove, Calif. : PWS Publishing, cop. 1999.*

Γραμμικοί τελεστές ανίχνευσης ακμών που προσεγγίζουν την 2η παράγωγο.

Όπως αναφέραμε και προηγούμενα στην εισαγωγή του ίδιου κεφαλαίου ένας εναλλακτικός τρόπος εύρεσης ακμών είναι με τον εντοπισμό των διελεύσεων της δεύτερης παραγώγου από το μηδέν (zero crossing). Οι εικόνες είναι συναρτήσεις δυο μεταβλητών κι έτσι η λαπλασιανή υπολογίζει το μέτρο (magnitude) της δεύτερης παραγώγου, και χωρίς να δίνει πληροφορία για την κατεύθυνση της ακμής. Αυτό όμως δεν μας δημιουργεί πρόβλημα αναφορικά με την εύρεση των ακμών, καθώς αυτό που μας ενδιαφέρει στις περισσότερες εφαρμογές είναι το μέτρο των ακμών και μόνο.

Λαπλασιανός τελεστής (Laplacian operator)

Για μια συνεχή συνάρτηση η λαπλασιανή δίνεται από τον τύπο,

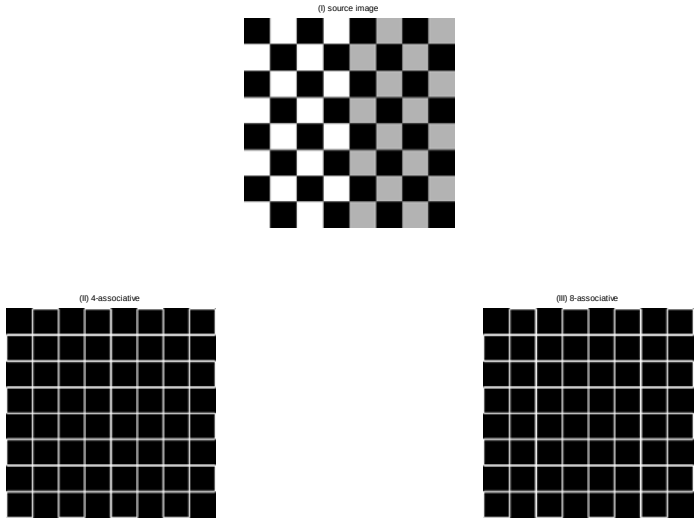
$$Laplacian = \nabla^2 \mathrm{I}(\mathrm{x}, \mathrm{y})$$

Για μια διακριτή συνάρτηση όπως η εικόνα, μπορεί να προσεγγιστεί από μικρά μητρώα συνέλιξης. Τα πιο δημοφιλή είναι :

$$L_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Η διαφορά των δύο μητρώων είναι η συσχετιστικότητα με τα γειτονικά εικονοστοιχεία. Το πρώτα λέμε ότι έχει συσχετιστικότητα 8, δηλαδή η έξοδος μετά την πράξη της συνέλιξης εξαρτάται από τα 8 γειτονικά εικονοστοιχεία του εξεταζόμενου. Ενώ για τον το δεύτερο μητρώο η συσχετιστικότητα είναι 4 καθώς εκτός του κεντρικού εικονοστοιχείου μόνο 4 ακόμη έχουν μη μηδενικές τιμές. Στην εικόνα που ακολουθεί βλέπουμε το αποτέλεσμα της συνέλιξης μεταξύ εικόνας και των μητρώων.



Εικόνα 9. Εφαρμογή τελεστή Laplace

Και με τους δύο πυρήνες οι ακμές ανιχνεύονται πανομοιότυπα.

Σχόλια για τον λαπλασιανό τελεστή

Με την χρήση αυτού του τελεστή μειώνουμε την πολυπλοκότητα υπολογισμού των ακμών σε σχέση με του τελεστές που προσεγγίζουν την πρώτη παράγωγο που προαναφέραμε. Με χρήση του τελεστή 8 συσχετιστικότητας, για κάθε εικονοστοιχείο ακμών χρειαζόμαστε 9 πολλαπλασιασμούς και 8 προσθέσεις, ενώ για το μητρώο με συσχετιστικότητα 4 ο αριθμός πέφτει σε 5 πολλαπλασιασμούς και 4 προσθέσεις. Φυσικά με παραγοντοποίηση και στις δύο περιπτώσεις η απαίτηση για πολλαπλασιαστές πέφτει στους 2.

Σε μερικές περιπτώσεις όμως ο λαπλασιανός τελεστής υπολείπεται αξιοπιστίας των τελεστών πρώτης παραγώγου. Εδώ έχουμε ένα trade-off μεταξύ πολυπλοκότητας και αξιοπιστίας που πρέπει να το αξιολογήσουμε.

Αναφορές

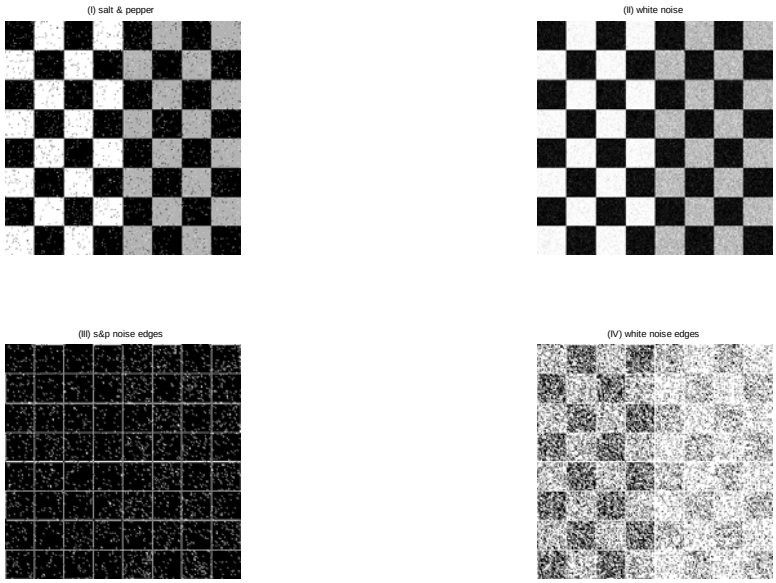
1. Σημειώσεις, Όραση Υπολογιστών, Πέτρος Μαραγκός, *Ε.Μ.Π. 2005 και 2009.*
2. Image processing, analysis, and machine vision. Milan Sonka, Vaclav Hlavac, Roger Boyle. *Pacific Grove, Calif. : PWS Publishing, cop. 1999.*

Συμπεριφορά τελεστών σε θορυβώδεις εικόνες.

Στα πραγματικά συστήματα είναι πολύ πιθανό να συναντήσουμε θόρυβο στις υπό επεξεργασία εικόνες. Παρακάτω εισάγουμε τεχνητά λευκό θόρυβο καθώς και salt & pepper στην αρχική εικόνα, για να δούμε την συμπεριφορά των τελεστών παρουσία θορύβου.

Τελεστής Roberts

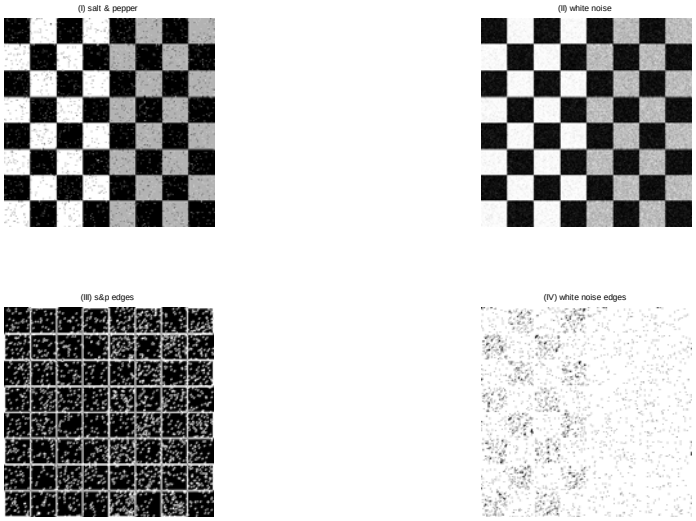
Στην εικόνα που ακολουθεί βλέπουμε ότι ο τελεστής Roberts ανιχνεύει πάρα πολλά ψευδή εικονοστοιχεία που θεωρεί ότι συνιστούν ακμές λόγω του θορύβου salt & pepper, και με την παρουσία λευκού θορύβου αποτυγχάνει τελείως να ανιχνεύσει ακμές..



Εικόνα 10. Συμπεριφορά τελεστή Robinson παρουσία Θορύβου

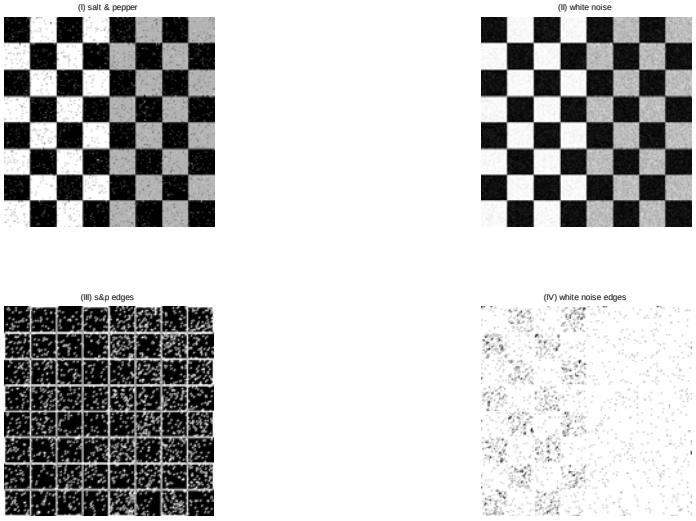
Παρόμοια συμπεριφορά παρουσιάζουν και οι υπόλοιποι τελεστές που αναφέραμε στην προηγούμενη ενότητα.

Τελεστής Prewitt



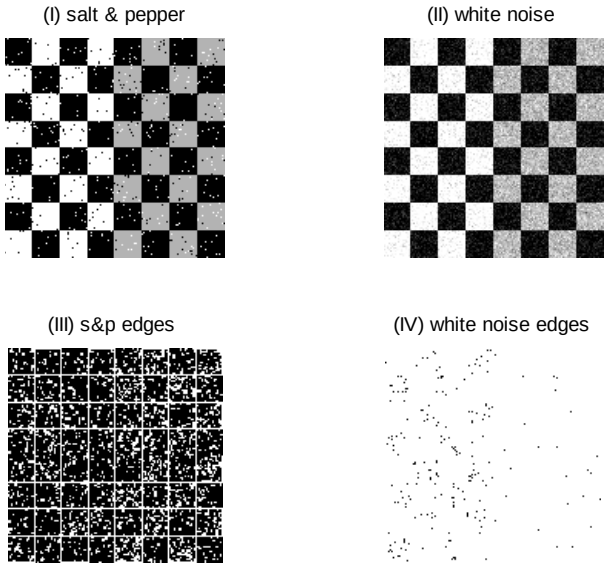
Εικόνα 11. Συμπεριφορά τελεστή Prewitt παρουσία Θορύβου

Τελεστής Sobel



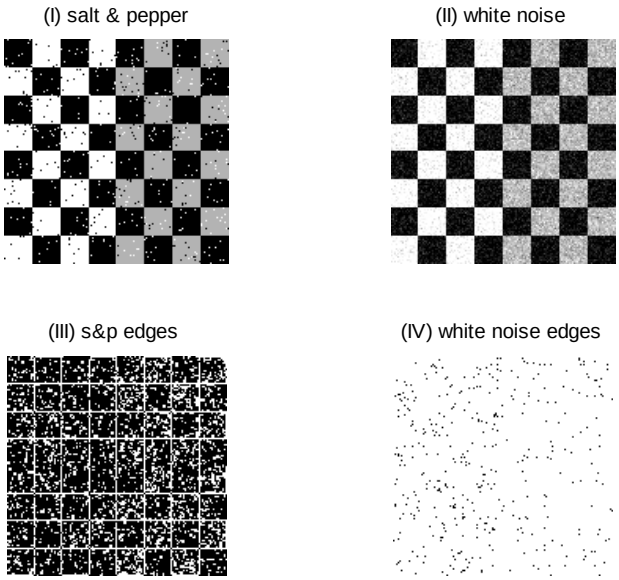
Εικόνα 12. Συμπεριφορά τελεστή Sobel παρουσία Θορύβου

Τελεστής Kirch



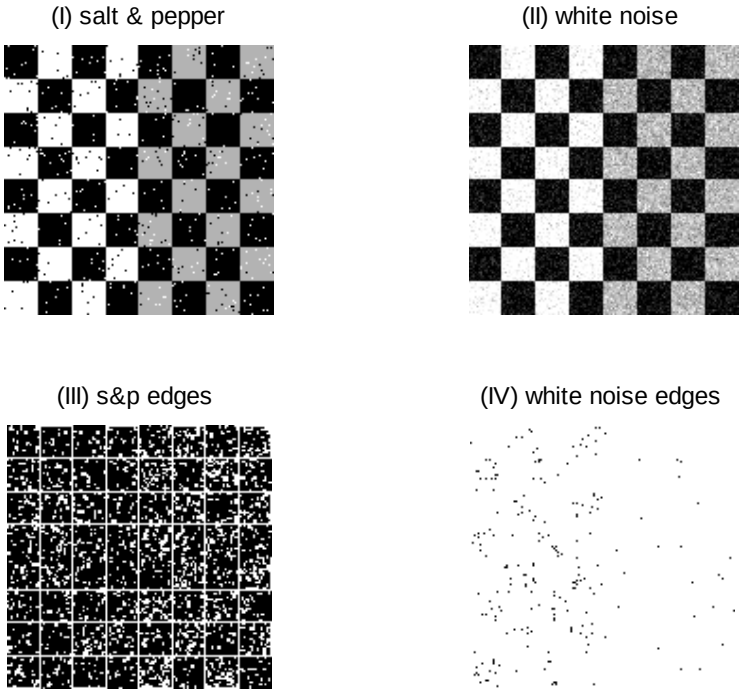
Εικόνα 13. Συμπεριφορά τελεστή Kirch παρουσία Θορύβου

Τελεστής Robinson



Εικόνα 14. Συμπεριφορά τελεστή Robinson παρουσία Θορύβου

Τελεστής Laplace



Εικόνα 15. Συμπεριφορά τελεστή Laplace παρουσία Θορύβου

Σχόλια για την συμπεριφορά των τελεστών παρουσία θορύβου

Όλοι οι τελεστές δεν έχουν την επιθυμητή συμπεριφορά παρουσία θορύβου και ειδικά για τον λευκό θόρυβο. Οι τελεστές Robinson,Kirch και Laplacian δεν πλησιάζουν καν τις πραγματικές ακμές, ακόμα και για μια τόσο απλή εικόνα.

Ο λόγος που συμβαίνει αυτό είναι ότι οι μητρώα συνέλιξης που χρησιμοποιήσαμε, στην ουσία αποτελούν ψηφιακά υψιπερατά φίλτρα. Έτσι, ενισχύουν τον υψίσυχνο θόρυβο οδηγώντας την έξοδο μακριά από τα επιθυμητά αποτελέσματα. Μια λύση είναι να χρησιμοποιήσουμε μη γραμμικά φίλτρα πριν την συνέλιξη της εικόνας με τα μητρώα ανίχνευσης ακμών. Για παράδειγμα ο salt and pepper θόρυβος μπορεί να εξαλειφθεί με ένα φίλτρο μέσης τιμής, όμως δεν θα έχει την ίδια επίδραση και για τον λευκό θόρυβο. Καταλαβαίνουμε πως δεν είναι μια λύση που θα δίνει πάντα αξιόπιστα αποτελέσματα.

Μια πολύ αποτελεσματική λύση πρότεινε ο Canny, χρησιμοποιώντας φιλτράρισμα με ένα γκαουσιανό φίλτρο, και κατόπιν χρησιμοποιεί κανονικά τα μητρώα συνέλιξης που προαναφέραμε. Θα περιγράψουμε τον αλγόριθμό του στην ακόλουθη ενότητα.

Canny edge detector

Ο αλγόριθμος που πρότεινε ο Canny για ανίχνευση ακμών σε εικόνες θεωρείται ο βέλτιστος που μπορούμε να ακολουθήσουμε για ανίχνευση ακμών παρουσία λευκού θορύβου. Για την υλοποίησή του απαιτούνται συγκεκριμένα βήματα όπως αναφέρει ο ίδιος στην δημοσίευσή του, *A Computational Approach to Edge Detection.*

Πρόθεση του Canny ήταν να βελτιώσει τους ήδη υπάρχοντες αλγόριθμους όταν ερευνούσε την περιοχή της ανίχνευσης ακμών. Για να το πετύχει αυτό όρισε κάποια κριτήρια για να αξιολογήσει την αποτελεσματικότητα των αλγόριθμων αυτών.

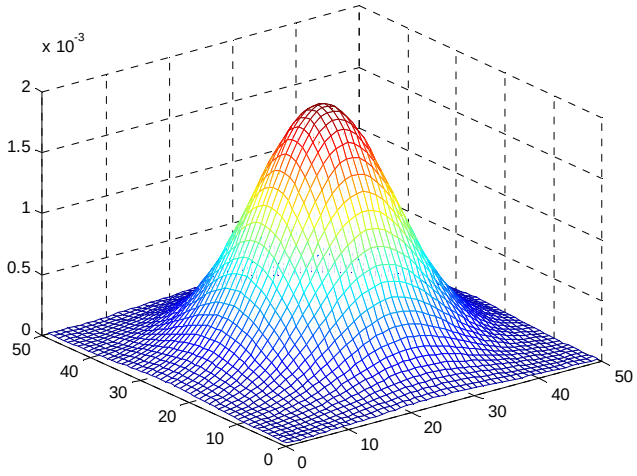
Πρώτο και πιο προφανές κριτήριο ήταν η ελαχιστοποίηση του σφάλματος. Είναι πολύ σημαντικό να ανιχνεύονται όλες οι πραγματικές ακμές (πραγματική είναι μια ακμή που υφίσταται και στον τρισδιάστατο πραγματικό κόσμο), και ταυτόχρονα να μην ανιχνεύονται ακμές που δεν υπάρχουν, ή να έχουμε «διπλές» αποκρίσεις σε μια ακμή.

Δεύτερο κριτήριο ήταν οι ακμές να είναι σωστά τοποθετημένες τοπικά. Η απόσταση μεταξύ της πραγματικής ακμής και της ακμής που εντοπίζει ο αλγόριθμος πρέπει να ελαχιστοποιηθεί. Επίσης η ακμή πρέπει να ορίζεται σαφώς και όχι να παίρνει εκτεταμένες διαστάσεις.

Βασιζόμενος σε αυτά τα κριτήρια ο Canny κατέληξε σε έναν αλγόριθμο όπου αρχικά στην εικόνα εφαρμόζεται ένα γκαουσιανό ψηφιακό φίλτρο (gaussian). Αυτό στοχεύει στην ελαχιστοποίηση της επίδρασης του θορύβου, και η διαδικασία ονομάζεται ομαλοποίηση της εικόνας (smoothing). Η ψηφιακή μορφή του φίλτρου είναι ένα τετραγωνικό μητρώο συνέλιξης. Όσο μεγαλώνει η διάσταση του φίλτρου και η τυπική απόκλιση (σ) της γκαουσιανής δυσδιάστατης κατανομής, τόσο περισσότερο εξομαλύνεται η εικόνα και μειώνεται η επίδραση του λευκού θορύβου. Οι τιμές του γκαουσιανού φίλτρου δίνονται από την σχέση :

$$G(x,y) = \frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{\sum_{\forall x}\sum_{\forall y} e^{\frac{-(x^2+y^2)}{2\sigma^2}}}$$

και έχει την μορφή του σχήματος στην εικόνα 16:



Εικόνα 16. Δισδιάστατο γκαουσιανό φίλτρο

Στην συνέχεια εφαρμόζεται τελεστής διαφόρισης στην εξομαλυμένη εικόνα. Μια βηματική ακμή χαρακτηρίζεται από την τοποθεσία της, την διεύθυνσή της και το μέτρο της. Ανιχνεύεται με την κατευθυντική παράγωγο της εικόνας (directional operator).

Αν υποθέσουμε ότι G είναι ένα δυσδιάστατο φίλτρο γκαουσιανής κατανομής και θέλουμε να υπολογίσουμε την συνέλιξη της εικόνας με την πρώτη παράγωγο κατά κατεύθυνση **n**.

$$G_{\vec{n}} = \frac{\partial G}{\partial n} = \vec{n}\,\nabla G \qquad (I)$$

Η διεύθυνση **n** πρέπει να είναι κάθετη στην κατεύθυνση της ακμής, παρόλο που αυτή η διεύθυνση δεν είναι δυνατόν να είναι γνωστή από την αρχή, μπορούμε να την προσεγγίσουμε για την εικόνα f ως εξής:

$$\vec{n} = \frac{\nabla(G*f)}{|\nabla(G*f)|} \qquad (II)$$

Οι ακμές τότε βρίσκονται από τα τοπικά μέγιστα της συνέλιξης μεταξύ της εικόνας f και του κατευθυνόμενου διαφορικού τελεστή G_n .

$$\frac{\partial}{\partial n}(G * f) = 0 \qquad (III)$$

Και συνδυάζοντας αυτή την σχέση με την (III) παίρνουμε

$$\frac{\partial^2}{\partial x^2}(G * f) = 0 \qquad (IV)$$

Από αυτή την εξίσωση προκύπτουν τα τοπικά μέγιστα σε κάθετη διεύθυνση από αυτή των ακμών. Ο τελεστής αυτός αναφέρεται στην βιβλιογραφία σαν *non-maxima suppression*.

Με βάση μια ελάχιστη τιμή του μέτρου των ακμών ($|G_n * f|$) (V) αποφασίζουμε την ύπαρξη ή όχι της ακμής. Για να αποφύγουμε την ανίχνευση ανύπαρκτων ακμών χρησιμοποιούμε

κατωφλίωση με υστέρηση. Αν εντοπίσουμε κάποια περιοχή με ένταση ακμών πάνω από ένα ισχυρό κατώφλι τις λαμβάνουμε σαν ακμές. Χαμηλότερες εντάσεις από αυτό το κατώφλι αγνοούνται εκτός και αν είναι γειτονικά συνδεδεμένες με περιοχές μεγάλης έντασης και ξεπερνούν ένα ελάχιστο κατώφλι. Τότε αυτές μάλλον είναι ακμές εξασθενημένες από τον θόρυβο και μετρούνται κανονικά.

Αλγόριθμος ανίχνευσης ακμών του Canny
1.Συνέλιξη της εικόνας με γκαουσιανή κατανομή τυπικής απόκλισης σ
2.Προσέγγιση τοπικών κατευθυνόμενων ακμών με την εξίσωση (III)
3.Εύρεση των περιοχών που συνιστούν ακμές με την χρήση της (IV)
4.Υπολογισμός της έντασης των ακμών με την εξίσωση (V)
5.Κατωφλίωση των ακμών με υστέρηση

Συμπεριφορά του ανιχνευτή ακμών του Canny

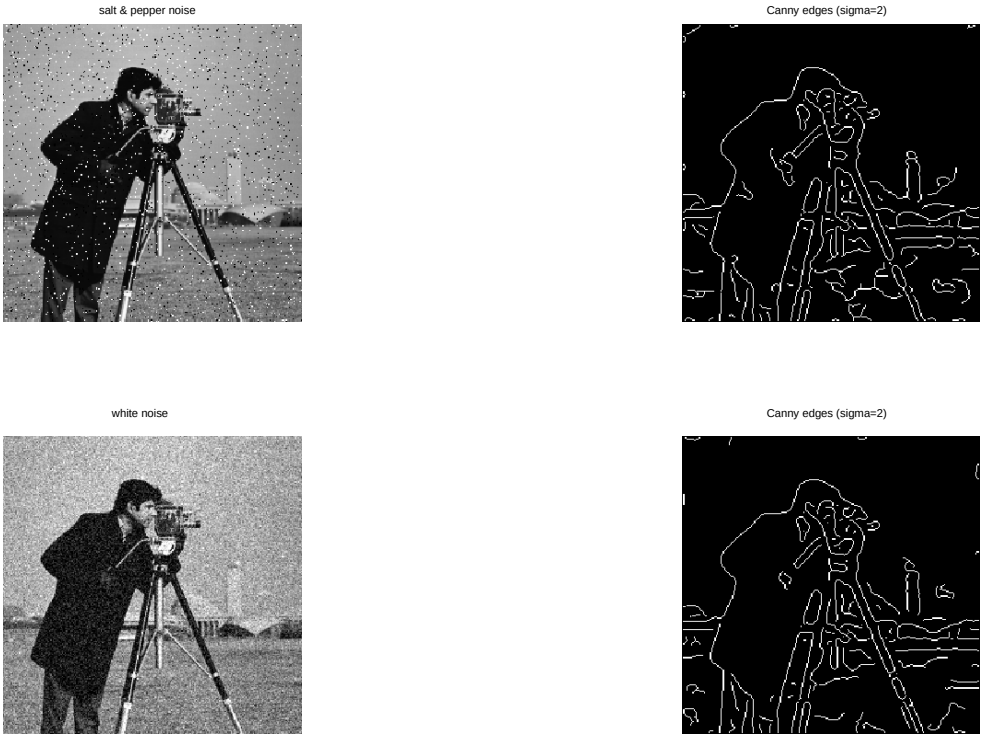
Στον ανιχνευτή ακμών που πρότεινε ο Canny μας δίνεται η δυνατότητα, ανάλογα με την τιμή της τυπικής απόκλισης που διαλέγουμε για το γκαουσιανό φίλτρο, να ανιχνεύσουμε λεπτομερείς ή γενικότερες ακμές. Στην εικόνα που ακολουθεί μπορούμε να παρατηρήσουμε την επίδραση της αύξησης της τυπικής απόκλισης στην ανίχνευση ακμών μιας εικόνας για τιμές από 0.5 έως 3.

Εικόνα 17. Εφαρμογή Αλγόριθμου ανίχνευσης ακμών του Canny.

Η διαφορά στο αποτέλεσμα του αλγόριθμου για διάφορες τιμές τυπικής απόκλισης οφείλεται στο ότι μεγαλώνοντας η τιμή της τυπικής απόκλισης του φίλτρου τόσο περισσότερο ομαλοποιεί την εικόνα και ακμές με πλάτος μικρότερο (τύπου γραμμής και στέγης) από αυτό του πυρήνα της συνέλιξης ουσιαστικά εξαλείφονται από το φίλτρο.

Συμπεριφορά παρουσία θορύβου

Όπως αναφέρθηκε και σε προηγούμενη παράγραφο σημαντικότερο πλεονέκτημα του ανιχνευτή ακμών που προτάθηκε από τον Canny είναι η συμπεριφορά του παρουσίας λευκού θορύβου. Στην εικόνα που ακολουθεί εφαρμόζουμε το αλγόριθμο του Canny σε δυο εικόνες που έχουμε εισάγει salt & pepper και λευκό θόρυβο.



Εικόνα 18. Συμπεριφορά του Canny edge detector παρουσία θορύβου.

σφάλματα στην ανίχνευση ακμών, στην περίπτωση της δεύτερης εικόνας όπου έχουμε εισάγει αρκετή ποσότητα λευκού θορύβου το αποτέλεσμα είναι σχεδόν ταυτόσημο με την «καθαρή» αρχική εικόνα.

Σχόλια για τον αλγόριθμο του Canny

Παρόλο που ο αλγόριθμος του Canny βελτιώνει αρκετά την ποιότητα των ανιχνευόμενων ακμών, είναι σαφές ότι αυξάνει κατά πολύ την πολυπλοκότητα αφού απαιτεί την συνέλιξη της εικόνας με δύο μητρώα συνέλιξης ένα εκ των οποίων μάλιστα(γκαουσιανό φίλτρο) πολύ μεγάλης διάστασης όσο αυξάνεται η τιμή της τυπικής απόκλισης *σ.* Επίσης και η κατωφλίωση με υστέρηση που ακολουθεί για την επιλογή των εικονοστοιχείων που απαρτίζουν ακμές εισάγει πρόσθετη πολυπλοκότητα και απαιτήσεις μνήμης.

Αναφορές

1. Canny, J., *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.

Ακμές Marr-Hildreth (zero crossings of Laplacian-of-Gaussian)

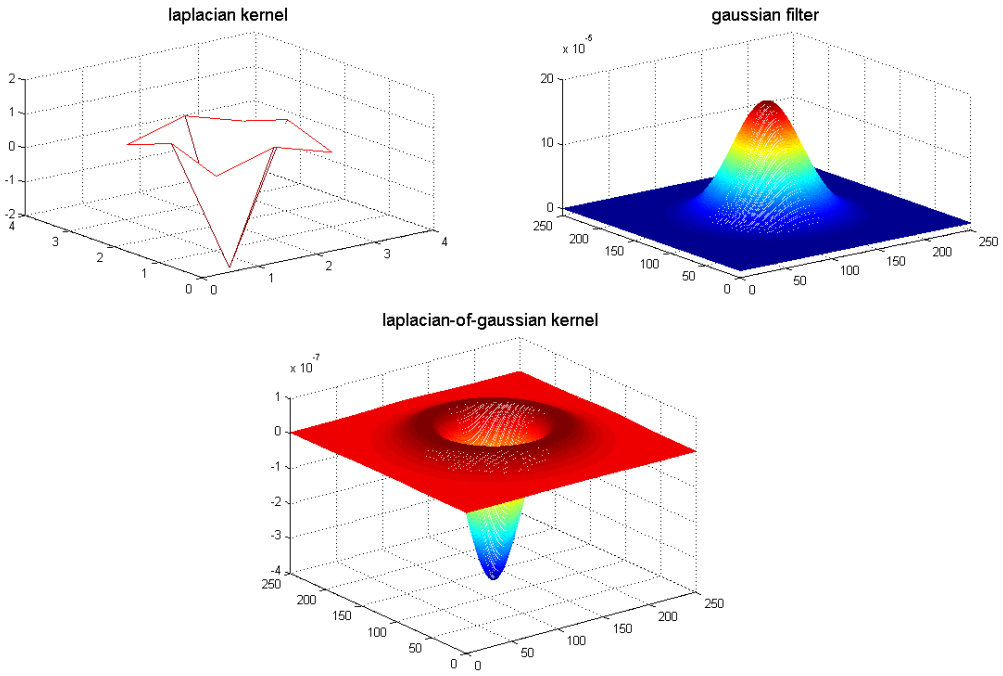
Οι Marr και Hildreth (1980) ανέπτυξαν την θεωρία τους σχετικά με την ανίχνευση ακμών στηριζόμενοι σε στοιχεία από βιολογικά συστήματα όρασης και ιδέες από την θεωρία σημάτων. Κατέληξαν λοιπόν στο ότι η ομαλοποίηση της εικόνας (smoothing) πρέπει να γίνεται με εφαρμογή ενός φίλτρου περιορισμένου κατά βέλτιστο τρόπο τόσο στο χωρικά όσο και στο πεδίο της συχνότητας. Ο χωρικός περιορισμός θα εξασφαλίζει την τοπική ανίχνευση ακμών και την αποφυγή λανθασμένου εντοπισμού ακμών που δεν υφίστανται (echoing edges). Ο συχνοτικός περιορισμός από την άλλη εξασφαλίζει ότι θα ανιχνεύονται ακμές συγκεκριμένης, επιθυμητής, έντασης. Οι δύο αυτοί περιορισμοί τους οδήγησαν στην επιλογή του φίλτρου γκαουσιανής κατανομής για την ομαλοποίηση εικόνων.

Άλλη μια βασική αρχή στην οποία κατέληξαν αφορά το πρώτο στάδιο της ανίχνευσης ακμών, που πρέπει να γίνεται ισοτροπικά, δηλαδή χωρίς να μας ενδιαφέρει η κατεύθυνση της ακμής. Ο λαπλασιανός τελεστής χρησιμοποιείται γι' αυτό το λόγο, καθώς όπως είδαμε παραπάνω πρόκειται για ένα ισοτροπικό δυσδιάστατο φίλτρο για ανίχνευση ακμών που στηρίζεται στην δεύτερη παράγωγο της εικόνας. Καθώς ο λαπλασιανός τελεστής είναι ένα μητρώο συνέλιξης καταλήγουμε ότι τα στάδια της ομαλοποίησης και της ανίχνευσης ακμών μπορούν να περατωθούν με την συνέλιξη της εικόνας και του πυρήνα laplacian-of-gaussian, που στην ουσία είναι η συνέλιξη του γκαουσιανού φίλτρου και του λαπλασιανού τελεστή για την δεύτερη παράγωγο.

$$(\nabla^2 \mathrm{G}_\sigma)(x,y) = \frac{\exp\left(-\frac{r^2}{2\sigma^2}\right)}{2\pi\sigma^4}\left(2-\frac{r^2}{\sigma^2}\right) \ , \qquad r=\sqrt{x^2+y^2}$$

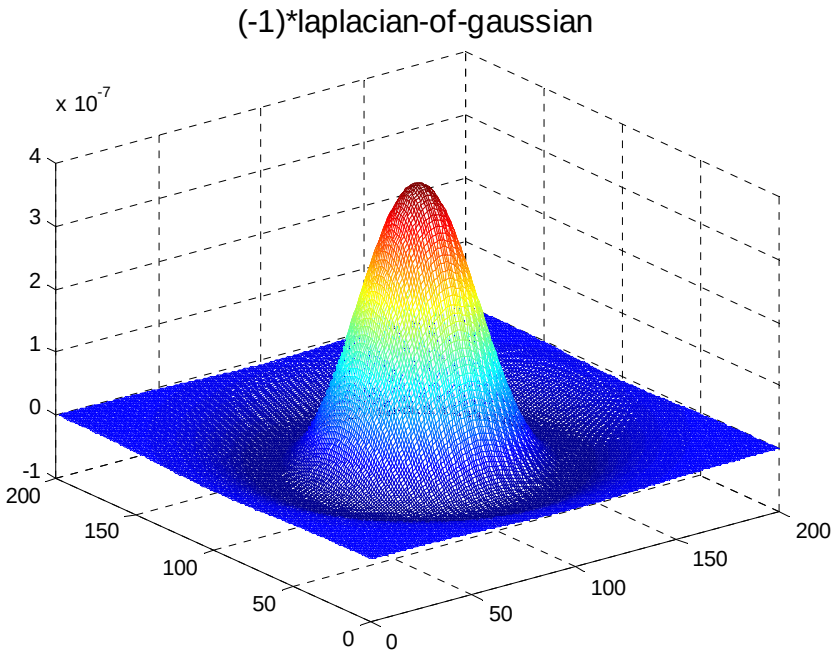
Με την χρήση του τελεστή LoG δηλαδή, δεν χρειάζεται πρώτα να ομαλοποιήσουμε την εικόνα με χρήση του gaussian blooring φίλτρου και μετά να εφαρμόσουμε τον λαπλασιανό τελεστή για την ανίχνευση της δεύτερης παραγώγου της εικόνας. Αυτό που γίνεται ουσιαστικά είναι συνέλιξη του γκαουσιανού φίλτρου με τον λαπλασιανό τελεστή ώστε να προκύψει ο επιθυμητός LoG τελεστής. Με αυτό τον τρόπο μειώνεται η πολυπλοκότητα του προβλήματος ανίχνευσης ακμών καθώς χρειαζόμαστε μια μόνο συνελικτική πράξη με όλη την εικόνα.

Στην εικόνα 19 που ακολουθεί στην επόμενη σελίδα παρουσιάζονται σχηματικά ο λαπλασιανός τελεστής, το γκαουσιανό φίλτρο ομαλοποίησης και το αποτέλεσμα της συνέλιξής τους.



Εικόνα 19. Κατασκευή του δισδιάστατου φίλτρου laplacian of gaussian

Το LoG φίλτρο είναι ένα ζωνοπερατό φίλτρο που μοιάζει με ανεστραμμένο μεξικάνικο καπέλο. Στην επόμενη εικόνα το παρουσιάζουμε ανεστραμμένο για καλύτερη επίδειξη.



Εικόνα 20. Παρουσίαση του ανεστραμμένου φίλτρου LoG

Λειτουργία του Laplacian-of-gaussian αλγόριθμου ανίχνευσης ακμών

Αμέσως καταλαβαίνουμε πως ο τελεστής laplacian-of-gaussian ανιχνεύει τα zero-crossings της δεύτερης παραγώγου της εικόνας, καθώς στηρίζεται στον λαπλασιανό τελεστή. Με την εφαρμογή του τελεστή στην εικόνα δημιουργούνται μηδενικές τιμές μακριά από ακμές, θετικές τιμές από την μια πλευρά της ακμής και αρνητικές από την άλλη. Ακριβώς πάνω στην ακμή οι τιμές που λαμβάνουμε είναι μηδενικές, αλλά αυτό εξαρτάται από το είδος της ακμής και το μέγεθος του τελεστή LoG που έχουμε επιλέξει. Ο εντοπισμός των μηδενικών αποτελεσμάτων του τελεστή πάνω στις ακμές μας δίνει την ακριβή θέση των ακμών. Επίσης και εδώ μπορούμε να χρησιμοποιήσουμε την τεχνική της κατωφλίωσης ώστε να ανιχνεύουμε ακμές με βάση τοπικά μέγιστα ή ελάχιστα.

Πλεονεκτήματα του τελεστή LoG

Με την εισαγωγή του τελεστή LoG οι Marr και Hildreth κατάφεραν να μειώσουν πολύ το υπολογιστικό κόστος για την ανίχνευση ακμών καθώς τώρα αρκεί μια συνέλιξη με μια και μόνο μάσκα που ταυτόχρονα ομαλοποιεί την εικόνα και υπολογίζει την δεύτερη παράγωγο. Αυτό αποτελεί σημαντική βελτίωση σε σχέση με τον αλγόριθμο του Canny που μελετήσαμε παραπάνω.

Ταυτόχρονα όμως διατηρούμε και τα κυριότερα πλεονεκτήματα που μελετήσαμε και στον αλγόριθμο του Canny.

Μεταβάλλοντας την τυπική απόκλιση του γκαουσιανού φίλτρου μεταβάλλουμε τις συχνότητες διέλευσης του ζωνοπερατού φίλτρου και με αυτόν τον τρόπο ανιχνεύουμε ακμές διαφορετικής κλίμακας. Όσο μεγαλώνει η τιμή της τυπικής απόκλισης *(σ)* τόσο πιο «γενικές» ακμές ανιχνεύονται, ενώ για μικρότερες τιμές του *(σ)* τόσο πιο λεπτομερείς ακμές ανιχνεύονται.

Για διάφορα μεγέθη του πυρήνα LoG κάνουμε το φίλτρο ανθεκτικότερο στην παρουσία θορύβου. Η διάσταση του φίλτρου βέβαια και πάλι συνδέεται με την τιμή της τυπικής απόκλισης που αναφέραμε στην προηγούμενη παράγραφο. Υπάρχει ένα tradeoff που πρέπει να λάβουμε υπ’ όψιν μας ανάμεσα στην λεπτομέρεια των ακμών που ανιχνεύουμε και την ευαισθησία του φίλτρου στον θόρυβο.

Αποτελέσματα του LoG τελεστή

Στις εικόνες που ακολουθούν μπορούμε να δούμε αποτελέσματα ανίχνευσης ακμών με χρήση του τελεστή laplacian of gaussian για διαφορετικές τιμές στην τυπική απόκλιση του γκαουσιανού φίλτρου και διαφορετικά μεγέθη του φίλτρου. Το μέγεθος του φίλτρου υπολογίζεται από την σχέση

$$R_{filter} = ceil(sigma * 3) * 2 + 1$$

Για κάθε εφαρμογή του αλγόριθμου παρουσιάζουμε

1. το αποτέλεσμα της ομαλοποίησης με γκαουσιανό φίλτρο αντίστοιχης τυπικής απόκλισης.
2. την συνέλιξη της εικόνας με τον τελεστή laplacian-of-gaussian
3. τα zero-crossings που ανιχνεύονται με κατώφλι μηδέν.

- **σ=2**

gauss conv *lapl-of-gaus.* *zero crossings*

- **σ=4**

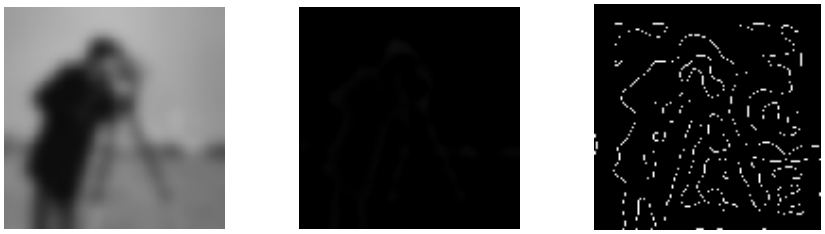
gauss conv *lapl-of-gaus.* *zero crossings*

- **σ=6**

gauss conv *lapl-of-gaus.* *zero crossings*

- **σ=8**

gauss conv *lapl-of-gaus.* *zero crossings*

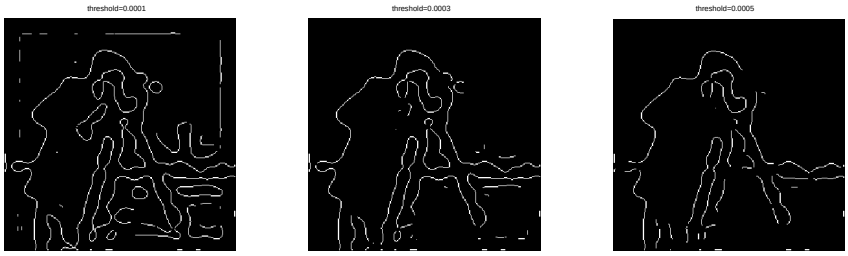


Εικόνα 21. Συμπεριφορά του φίλτρου LoG μεταβάλλοντας την τυπική απόκλιση

Παρατηρούμε την ίδια αρχή που ισχύει και στην περίπτωση του ανιχνευτή του Canny. Όσο μεγαλώνει η τιμή της τυπικής απόκλισης του γκαουσιανού φίλτρου, τόσο γενικότερες είναι οι ακμές που ανιχνεύονται και αγνοούνται οι πιο λεπτομερείς.

Άλλος ένας παράγοντας όμως που επηρεάζει τις ακμές που ανιχνεύονται είναι η χρήση του κατωφλίου. Λαμβάνουμε δηλαδή ως ακμή τα zero-crossings εκείνα που στις πλευρές της ακμής ξεπερνούν μια ελάχιστη θετική τιμή, ή πέφτουν χαμηλότερα από μια αρνητική.

Στην εικόνα που ακολουθεί μπορούμε να παρατηρήσουμε την επίδραση της χρήσης κατωφλίου για την ανίχνευση ακμών. Και στις τρεις εφαρμογές χρησιμοποιούμε τον τελεστή laplacian of gaussian με τυπική απόκλιση $\sigma=6$, και σταδιακά αυξάνουμε το κατώφλι. Με αυτό τον τρόπο περιορίζουμε την ανίχνευση σε ισχυρότερες ακμές.



Εικόνα 22. Συμπεριφορά του LoG φίλτρου μεταβάλλοντας το κατώφλι για αποδοχή pixel ως μέρος ακμής.

Αναφορές

1. Σημειώσεις, Όραση Υπολογιστών, Πέτρος Μαραγκός, *Ε.Μ.Π. 2005 και 2009.*

Bi-level Laplacian of Gaussian filter

Όπως είδαμε στο προηγούμενο κεφάλαιο με την χρήση του laplacian of Gaussian φίλτρου πετύχαμε σημαντική μείωση στο κόστος εύρεσης της ακμής μιας εικόνας. Παρόλα αυτά αν παρατηρήσουμε το μητρώο συνέλιξης 7x7 που προκύπτει όπως περιγράψαμε προηγούμενα, για τυπική απόκλιση *σ=1,*

0.0005	*0.0028*	*0.0088*	*0.0125*	*0.0088*	*0.0028*	*0.0005*
0.0028	*0.0177*	*0.0394*	*0.0433*	*0.0394*	*0.0177*	*0.0028*
0.0088	*0.0394*	*0.0002*	*-0.0964*	*0.0002*	*0.0394*	*0.0088*
0.0125	*0.0433*	*-0.0964*	*-0.3183*	*-0.0964*	*0.0433*	*0.0125*
0.0088	*0.0394*	*0.0002*	*-0.0964*	*0.0002*	*0.0394*	*0.0088*
0.0028	*0.0177*	*0.0394*	*0.0433*	*0.0394*	*0.0177*	*0.0028*
0.0005	*0.0028*	*0.0088*	*0.0125*	*0.0088*	*0.0028*	*0.0005*

Βλέπουμε ότι έχει 10 διαφορετικούς συντελεστές. Για τον υπολογισμό δηλαδή ενός και μόνο pixel απαιτούνται 10 πολλαπλασιασμοί. Με σκοπό την περαιτέρω μείωση του κόστους υπολογισμού της συνέλιξης μιας εικόνας με το laplacian of gaussian φίλτρο, οι Pei και Horng προτείνουν στην δημοσίευσή τους έναν τρόπο προσέγγισης του LoG φίλτρου με δύο μόνο διαφορετικές τιμές συντελεστών.

Το BLoG σε μία διάσταση

Οι Pei και Horng ασχολούνται με την προσέγγιση του laplacian of gaussian φίλτρου σαν ένα πρόβλημα βελτιστοποίησης. Κατασκευάζουν ένα φίλτρο που έχει μόνο δύο διαφορετικές μη μηδενικές τιμές συντελεστών. Οι επιλεγόμενες τιμές για τους συντελεστές πρέπει να οδηγούν στην ελαχιστοποίηση της νόρμας του σφάλματος για την προσέγγιση. Το BLoG φίλτρο είναι απλά μια προσέγγιση του LoG αλλά μπορεί να υπολογιστεί ιδιαίτερα αποδοτικά, καθώς απαιτεί μόλις δύο πολλαπλασιασμούς ανεξάρτητα από το μέγεθος του φίλτρου. Θα ξεκινήσουμε μελετώντας το φίλτρο στην μία διάσταση για να εξηγήσουμε την κατασκευή και την λειτουργία του.

Το φίλτρο Laplacian of gaussian, για μία διάσταση, παίρνει τις τιμές του από την σχέση

$$LoG(n,\sigma) = (\nabla^2 \mathrm{G}_\sigma)(n,\sigma) = \frac{\exp\left(-\frac{n^2}{2\sigma^2}\right)}{\sqrt{2\pi}\sigma^3}\left(1-\frac{n^2}{\sigma^2}\right)$$

Ορίζουμε προσέγγιση BLoG του φίλτρου Laplacian of Gaussian την συνάρτηση

$$BLoG(n,N_1,N_2,F_1,F_2) = \begin{cases} F_1, & |n| \le N_1 \\ F_2, & N_1 < |n| \le N_1 \\ 0, & |n| > N_2 \end{cases}$$

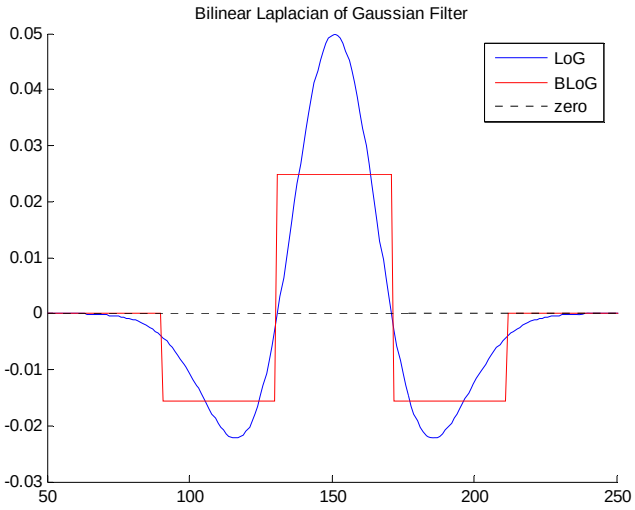
Στόχος μας είναι να ορίσουμε τις τιμές των μεταβλητών για την συνάρτηση BLoG ώστε αυτή να προσεγγίζει όσο το δυνατόν καλύτερα το φίλτρο laplacian of gaussian. Η συνάρτηση σφάλματος της προσέγγισης είναι:

$$E_p = \|BLoG(n, N_1, N_2, F_1, F_2) - LoG(n, \sigma)\|_p$$

Οι τρεις πρώτες μεταβλητές του φίλτρου θα χρησιμοποιηθούν για την ελαχιστοποίηση του σφάλματος και η τελευταία θα μας βοηθήσει να τηρήσουμε την προδιαγραφή ότι η DC απόκριση του φίλτρου πρέπει να είναι μηδέν. Έτσι λοιπόν επιλέγουμε για τις μεταβλητές τις τιμές:

- $N_1=\sigma$ και αν η τυπική απόκλιση δεν είναι ακέραιος τότε στρογγυλοποιούμε στον πλησιέστερα.
- Η F_1 ορίζεται από διαφορετικά κριτήρια βελτιστοποίησης
  - L_1-νόρμα : $F_1 = LoG(\frac{N_1}{2}, \sigma)$
  - L_2-νόρμα : $F_1 = \frac{1}{2N_1+1}\sum_{n=-N_1}^{N_1} LoG(n, \sigma)$
  - L_∞-νόρμα : $F_1 = \frac{1}{2}LoG(0, \sigma)$
- $N_2=3N_1$
- $F_2 = \frac{-F_1(2N_1+1)}{2(N_2-N_1)}$

Για παράδειγμα αν χρησιμοποιήσουμε την L_∞ για την προσέγγιση της συνάρτησης laplacian of gaussian με τυπική απόκλιση $\sigma=2$, παίρνουμε την συνάρτηση που εικονίζεται με κόκκινο στην παρακάτω γραφική παράσταση.



Εικόνα 23. Προσέγγιση του μονοδιάστατου λαπλασιανού φίλτρου.

Το φίλτρο BLoG σε δύο διαστάσεις

Οι ίδιες ιδέες που παρουσιάστηκαν σε μία διάσταση για το BLoG φίλτρο μπορούν να επεκταθούν και στις δύο διαστάσεις. Οι Pei και Horng στην δημοσίευσή τους προτείνουν δύο τρόπους για την επέκταση του φίλτρου σε δύο διαστάσεις. Ο πρώτος, και αυτός που θα ακολουθήσουμε εμείς, αφορά την γενίκευση του μονοδιάστατου φίλτρου. Η δεύτερη κάνει

χρήση του μετασχηματισμού McClellan για την κατασκευή του μητρώου συνέλιξης για το BLoG φίλτρο, εκμεταλλευόμενο την κυκλική συμμετρία του.

Γενικεύοντας το μονοδιάστατο φίλτρο σε δύο διαστάσεις

Όπως είδαμε και σε προηγούμενο κεφάλαιο οι τιμές του φίλτρου LoG δίνονται από την συνάρτηση:

$$\text{LoG(x,y,σ)}=(\nabla^2 G_\sigma)(x,y,\sigma)=\frac{\exp\left(-\frac{r^2}{2\sigma^2}\right)}{2\pi\sigma^4}\left(2-\frac{r^2}{\sigma^2}\right)\ ,\ r=\sqrt{x^2+y^2}$$

Και ορίζουμε την προσέγγιση του φίλτρου αυτού με την συνάρτηση

$$BLoG(x,y,R_1,R_2,F_1,F_2)=\begin{cases}F_1, & x^2+y^2\le R_1^2\\ F_2, & R_1^2<x^2+y^2\le R_2^2\\ 0, & x^2+y^2>R_2^2\end{cases}$$

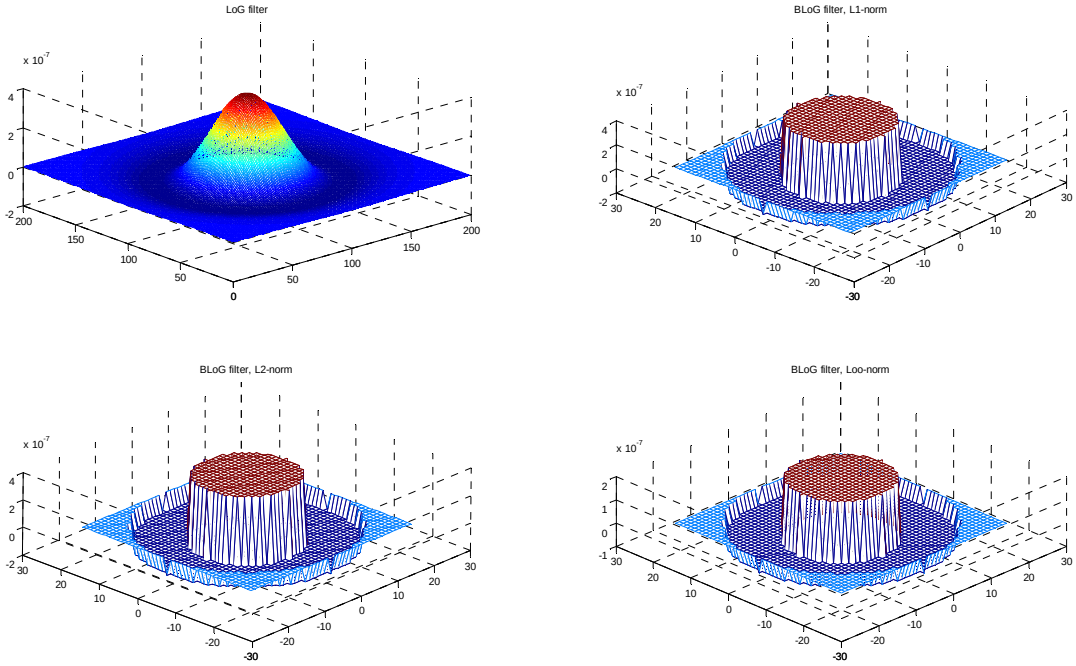
Σε αυτή την περίπτωση η συνάρτηση σφάλματος της προσέγγισης είναι :

$$E_p=\|BLoG(x,y,R_1,R_2,F_1,F_2)-\text{LoG(x,y,σ)}\|_p$$

Με τις ίδιες παραδοχές που γίνανε για την περίπτωση των τριών διαστάσεων καταλήγουμε στις παρακάτω τιμές για τις μεταβλητές του BLoG φίλτρου.

- $R_1=\sigma\sqrt{2}$, και αν δεν προκύψει ακέραιος,τότε στρογγυλοποιούμε στον πλησιέστερο.
- Η παράμετρος F_1 και στην περίπτωση των δύο διαστάσεων καθορίζεται από τρεις διαφορετικές νόρμες.
  - L_1-νόρμα : $F_1=LoG(\frac{R_1}{2},0,\sigma)$
  - L_2-νόρμα : $F_1=\frac{\sum\sum_{x^2+y^2\le R_1^2} LoG(x,y,\sigma)}{\sum\sum_{x^2+y^2\le R_1^2} 1}$
  - L_∞-νόρμα : $F_1=\frac{1}{2}LoG(0,0,\sigma)$
- $R_2=2R_1$
- Και η παράμετρος F_2 δίνεται από την σχέση $F_2=\frac{-F_1\sum\sum_{x^2+y^2\le R_1^2}1}{\sum\sum_{R_1^2<x^2+y^2\le R_2^2}1}$

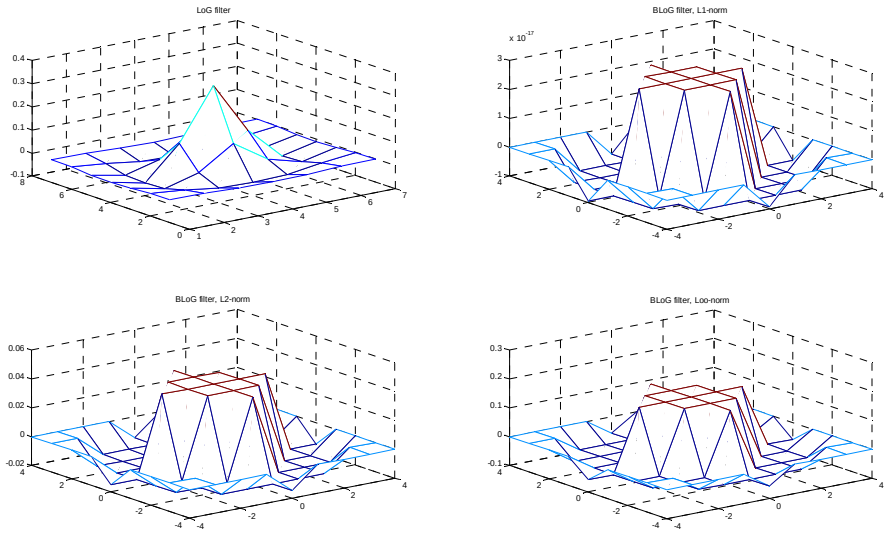
Χρησιμοποιώντας αυτές τις νόρμες μπορούμε να σχεδιάσουμε το δυσδιάστατο φίλτρο laplacian of gaussian. Τα τρία φίλτρα που προσεγγίζουν το αρχικό εικονίζονται σε γραφήματα στην εικόνα 24 που ακολουθεί.



Εικόνα 24. Προσέγγιση του λαπλασιανού φίλτρου με τις τρεις νόρμες του BloG φίλτρου

Κατασκευή ενός BloG Φίλτρου

Θα κατασκευάσουμε ένα φίλτρο BLoG που θα προσεγγίζει το αντίστοιχο LoG με τυπική απόκλιση *σ=1*. Για τις τρεις νόρμες που μπορούμε να χρησιμοποιήσουμε προκύπτουν τα παρακάτω φίλτρα που απεικονίζονται γραφικά.



Εικόνα 25. Προσέγγιση ενός 5x5 laplacian of Gaussian φίλτρου.

Στην συγκεκριμένο παράδειγμα θα χρησιμοποιήσουμε το φίλτρο που προκύπτει για την L_∞ νόρμα.

Παράμετροι BLoG φίλτρου για σ=1	
R_1	2
R_2	4
F_1	-0.1592
F_2	0.0575

Και το πλήρες φίλτρο 9x9 είναι:

0	0	0	0	0.0575	0	0	0	0
0	0	0.0575	0.0575	0.0575	0.0575	0.0575	0	0
0	0.0575	0.0575	0.0575	-0.1592	0.0575	0.0575	0.0575	0
0	0.0575	0.0575	-0.1592	-0.1592	-0.1592	0.0575	0.0575	0
0.0575	0.0575	-0.1592	-0.1592	-0.1592	-0.1592	-0.1592	0.0575	0.0575
0	0.0575	0.0575	-0.1592	-0.1592	-0.1592	0.0575	0.0575	0
0	0.0575	0.0575	0.0575	-0.1592	0.0575	0.0575	0.0575	0
0	0	0.0575	0.0575	0.0575	0.0575	0.0575	0	0
0	0	0	0	0.0575	0	0	0	0

Παρατηρούμε ότι έχουμε μόλις δύο διαφορετικούς συντελεστές, έτσι μας δίνεται η δυνατότητα να υπολογίσουμε την έξοδο του φίλτρου με μόλις δύο πολλαπλασιασμούς, και λιγότερες προσθέσεις καθώς το φίλτρο έχει και μηδενικές τιμές στις γωνίες του. Αυτό είναι το σημαντικότερο πλεονέκτημα της προσέγγισης αυτής, καθώς απλοποιεί κατά πολύ την μορφή του φίλτρου.

Εφαρμογή του φίλτρου BLoG

Για την ανίχνευση ακμών με το φίλτρο που μόλις ορίσαμε θα χρειαστεί να εκτελεστεί η πράξη της συνέλιξης με την εικόνα. Τα μη μηδενικά στοιχεία του φίλτρου είναι 49, και έχουμε δυο διαφορετικές τιμές συντελεστών. Δώδεκα στοιχεία έχουν την τιμή -0.1592 και 37 την τιμή 0.0575. Επομένως για τον υπολογισμό της τιμής εξόδου ενός εικονοστοιχείου θα χρειαστούν

- 11+35+1 = 47 προσθέσεις
- Και 2 πολλαπλασιασμοί.

Αν συγκρίνουμε αυτή την περίπτωση με το αντίστοιχο φίλτρο LoG ίδιας διάστασης θα παρατηρήσουμε ότι ο αριθμός και το κόστος των πράξεων είναι πολύ μεγαλύτερος.

Ο πυρήνας του LoG φίλτρου 9x9 είναι

0.0020	0.0028	0.0034	0.0037	0.0037	0.0037	0.0034	0.0028	0.0020
0.0028	0.0036	0.0034	0.0024	0.0017	0.0024	0.0034	0.0036	0.0028
0.0034	0.0034	0.0009	-0.0033	-0.0054	-0.0033	0.0009	0.0034	0.0034
0.0037	0.0024	-0.0033	-0.0113	-0.0152	-0.0113	-0.0033	0.0024	0.0037
0.0037	0.0017	-0.0054	-0.0152	-0.0200	-0.0152	-0.0054	0.0017	0.0037
0.0037	0.0024	-0.0033	-0.0113	-0.0152	-0.0113	-0.0033	0.0024	0.0037
0.0034	0.0034	0.0009	-0.0033	-0.0054	-0.0033	0.0009	0.0034	0.0034
0.0028	0.0036	0.0034	0.0024	0.0017	0.0024	0.0034	0.0036	0.0028
0.0020	0.0028	0.0034	0.0037	0.0037	0.0037	0.0034	0.0028	0.0020

Συγκεκριμένα βλέπουμε ότι όλοι οι συντελεστές του φίλτρου (81) είναι μη μηδενικοί, και υπάρχουν 13 διαφορετικές τιμές συντελεστών. Πράγμα που σημαίνει ότι θα χρειαστούν

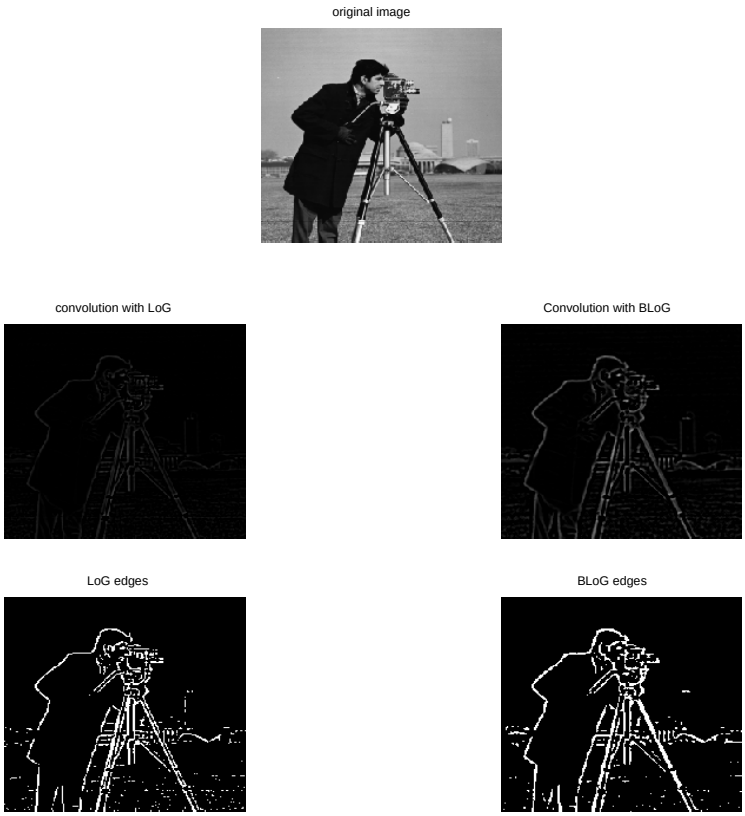
- 80+1 = 81 προσθέσεις
- 13 πολλαπλασιασμοί

Στον πίνακα που ακολουθεί γίνεται μια σύγκρισή του κόστους υπολογισμού για την έξοδο των δύο φίλτρων.

	Προσθέσεις	Μείωση%	Πολ.	Μείωση%
LoG 9x9	81		13	
BLoG 9x9	47	42%	2	85%

Η μείωση του κόστους που πετυχαίνουμε και ειδικά από την πράξη του πολλαπλασιασμού είναι εξαιρετικά σημαντική και θα μας βοηθήσει να καταλήξουμε σε μια πιο αποδοτική υλοποίηση.

Με την εφαρμογή και την σύγκριση των δύο φίλτρων μπορούμε να διαπιστώσουμε ότι το φίλτρο BLoG αποτελεί μια πάρα πολύ καλή προσέγγιση του laplacian of Gaussian. Στο παράδειγμα που ακολουθεί χρησιμοποιήσαμε φίλτρα με τυπική απόκλιση *σ=1* και διαστάσεις 9x9. Η εύρεση των ακμών γίνεται με απλή θεοτική κατωφλίωση.



Εικόνα 26. Σύγκριση του laplacian of Gaussian και της Bilevel προσέγγισής του.

Οι ακμές που ανιχνεύονται στις δύο περιπτώσεις είναι εξαιρετικά συναφείς. Μικρές διαφορές παρατηρούνται στο πάχος των ακμών, πράγμα που οφείλεται στην ομοιομορφία των συντελεστών του BLoG φίλτρου σε σχέση με το laplacian of gaussian.

Σχόλια για το BLoG φίλτρο

Με το φίλτρο αυτό που κατασκευάσαμε προσεγγίζοντας το Laplacian of gaussian φίλτρο πετύχαμε να καταλήξουμε σε ένα φίλτρο που διατηρεί τα πλεονεκτήματα του LoG και ταυτόχρονα μειώνει σημαντικά το κόστος υπολογισμού. Έτσι λοιπόν στα ήδη γνωστά πλεονεκτήματα του LoG:

- Ισοτροπική ανίχνευση ακμών (ανεξάρτητο από την κατεύθυνση της ακμής φίλτρο)
- Πιστή τοπικότητα στην ανίχνευση των ακμών
- Αναισθησία στον θόρυβο
- Ταυτόχρονη εξομάλυνση της εικόνας και ανίχνευση των ακμών με μια και μόνο συνέλιξη
- Απόκριση σε συγκεκριμένη συχνότητα ακμών

Ερχόμαστε τώρα να προσθέσουμε και το:

- Χαμηλότερο κόστος υπολογισμού.

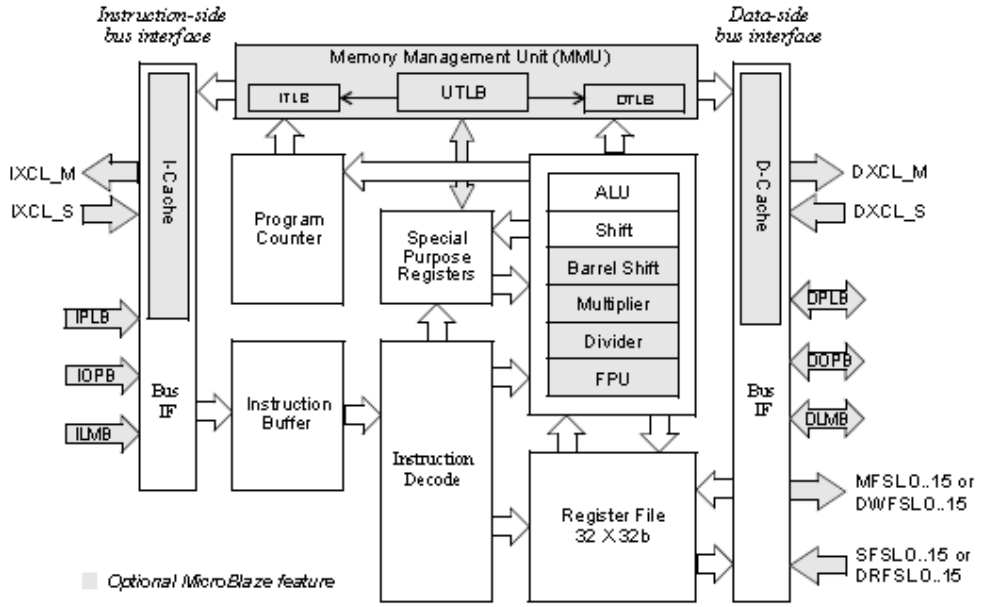
Αναφορές

1. Design of bilevel Laplacian-of-Gaussian filter. Soo-Chang Pei, Ji-Hwei Horng. Elsevier Signal Proccesing. 2002.

Υλοποίηση Αλγορίθμων σε λογισμικό

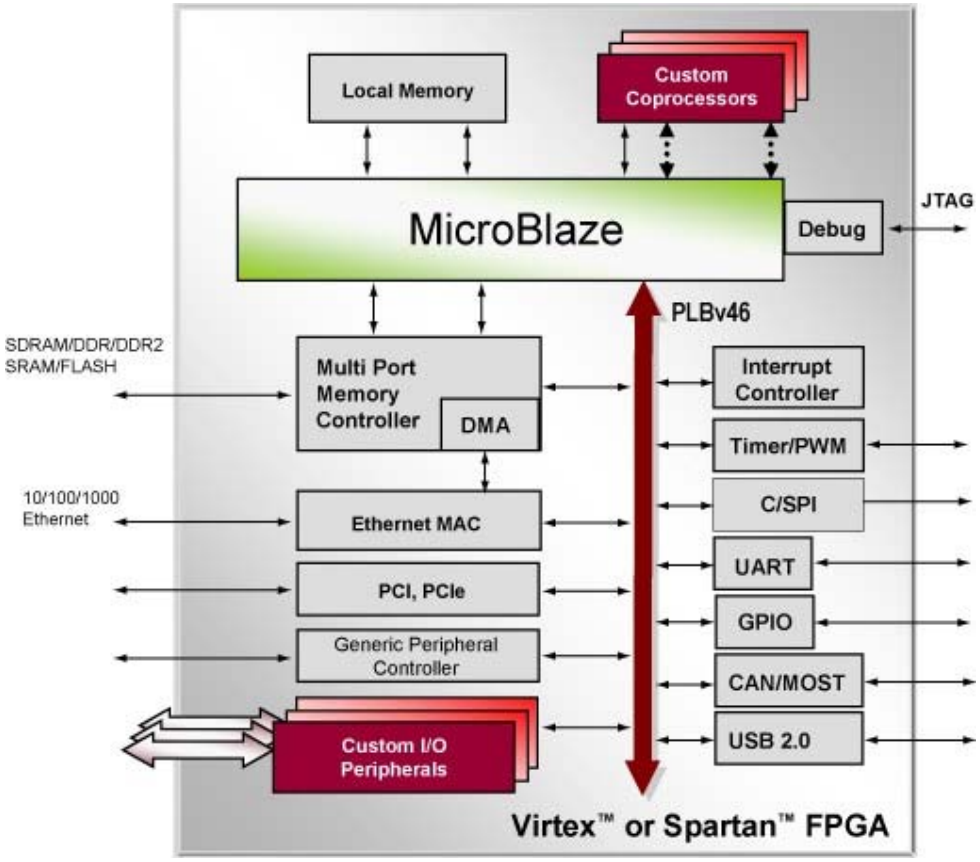
Η πλατφόρμα Xilinx Spartan 3E και ο μικροεπεξεργαστής Microblaze

Για την υλοποίηση του αλγόριθμου που καταλήξαμε στο προηγούμενο κεφάλαιο θα χρησιμοποιήσουμε την πλατφόρμα Xilinx Spartan-3E FPGA, σε συνδυασμό με το περιβάλλον ανάπτυξης εφαρμογών Xilinx EDK 8.1. Στο συγκεκριμένο FPGA μπορούμε να χρησιμοποιήσουμε τον μικροεπεξεργαστή Microblaze και μια σειρά από περιφερειακά όπως σειριακές συνδέσεις, χρονόμετρα, μνήμη RAM ακόμα και περιφερειακά που θα αναπτύξουμε εμείς σε γλώσσα περιγραφής υλικού. Στο μπλοκ διάγραμμα που ακολουθεί παρουσιάζονται οι κυριότερες λειτουργικές μονάδες του microblaze.



Εικόνα 27. Αρχιτεκτονική του microblaze

Ο microblaze μπορεί αν συνδεθεί με τα περιφερειακά μέσω διαύλων, τον OPB (On Chip peripherial bus) και τον PLB (Proccesor Local Bus). Τα περιφερειακά συνδέονται σε slave mode και ο επεξεργαστής είναι πάντα master, και φυσικά υπάρχει πάντα και ο απαραίτητος διαιτητής διαδρόμου που φροντίζει για την ομαλή επικοινωνία των περιφερειακών με τον επεξεργαστή.



Εικόνα 28. Αρχιτεκτονική συστήματος με τον microblaze, τους διαύλους και τα περιφερειακά.

Σε περιπτώσεις που η απόδοση είναι το ζητούμενο μπορεί να χρησιμοποιηθεί και η σύνδεση FSL μεταξύ ενός περιφερειακού και του επεξεργαστή. Το περιφερειακό σε αυτή την περίπτωση λειτουργεί σαν συνεπεξεργαστής στο σύστημα. Έτσι πετυχαίνουμε την αδιάκοπη επικοινωνία του επεξεργαστή με τον περιφερειακό ανεξάρτητα αν γίνεται χρήση του διαδρόμου από άλλο περιφερειακό, και μπορούμε να εκμεταλλευτούμε καλύτερα την σωλήνωση (pipeline) του microblaze. Στην συνέχεια θα προχωρήσουμε στην περιγραφή μερικών από τα χαρακτηριστικά ενός συστήματος που αναπτύσσεται στο FPGA Spartan 3E της Xilinx, και θα μας απασχολήσουν και στην ανάπτυξη του συστήματός μας.

Microblaze soft processor

Ο Microblaze είναι ένας soft-processor. Αυτό σημαίνει ότι δεν αποτελεί ξεχωριστό ολοκληρωμένο κύκλωμα πάνω στην πλακέτα του Spartan 3E αλλά πρόκειται για έναν επεξεργαστή υλοποιημένο σε γλώσσα περιγραφής υλικού με σκοπό να υλοποιείται πάνω στην ψηφίδα του FPGA της Xilinx, μαζί με τους διαδρόμους και τα άλλα περιφερειακά του συστήματος. Τα κύρια χαρακτηριστικά του είναι:

- RISC (reduced instruction set) σύνολο εντολών με 32-bit μέγεθος εντολών
- Δίαυλος 32-bit

- Σωλήνωση (pipeline) 3 σταδίων για βελτιστοποίηση ως προς επιφάνεια, και 5 σταδίων για βελτιστοποίηση ως προς απόδοση.
- 32 καταχωρητές 32-bit γενικής χρήσης.
- Υποστήριξη σε hardware πολλαπλασιαστή, διαιρέτη και μονάδα κινητής υποδιαστολής.

Ανάλογα με την soft-core έκδοση του microblaze χρησιμοποιούμε, έχουμε πολλές παραμέτρους που μπορούμε να ελέγξουμε. Επιλογές που έχουν να κάνουν με χρήση μνημών Cache, επιλογή διαύλων και συνδέσεων, χρήση hardware για υπολογισμούς στην αριθμητική λογική μονάδα και τα στάδια της σωλήνωσης ανάλογα με την παράμετρο βελτιστοποίησης που θα επιλέξουμε για τον επεξεργαστή. Από την Xilinx προτείνεται η χρήση της τελευταίας έκδοσης του επεξεργαστή αλλά αυτό εξαρτάται από την έκδοση του EDK που χρησιμοποιούμε. Στον πίνακα που ακολουθεί παραθέτουμε όλες τις παραμετροποιήσιμες δυνατότητες του microblaze.

Feature	**MicroBlaze Versions**				
	v4.00	**v5.00**	**v6.00**	**v7.00**	**v7.10**
Version Status	deprecated	deprecated	deprecated	deprecated	preferred
Processor pipeline depth	3	5	3/5	3/5	3/5
On-chip Peripheral Bus (OPB) data side interface	option	option	option	option	option
On-chip Peripheral Bus (OPB) instruction side interface	option	option	option	option	option
Local Memory Bus (LMB) data side interface	option	option	option	option	option
Local Memory Bus (LMB) instruction side interface	option	option	option	option	option
Hardware barrel shifter	option	option	option	option	option
Hardware divider	option	option	option	option	option
Hardware debug logic	option	option	option	option	option
Fast Simplex Link (FSL) interfaces	0-7	0-7	0-7	0-15	0-15
Machine status set and clear instructions	option	Yes	option	option	option
Instruction cache over IOPB interface	option	No	No	No	No
Data cache over IOPB interface	option	No	No	No	No
Instruction cache over CacheLink (IXCL) interface	option	option	option	option	option
Data cache over CacheLink (DXCL) interface	option	option	option	option	option
4 or 8-word cache line on XCL	4	option	option	option	option
Hardware exception support	option	option	option	option	option
Pattern compare instructions	option	Yes	option	option	option
Floating point unit (FPU)	option	option	option	option	option

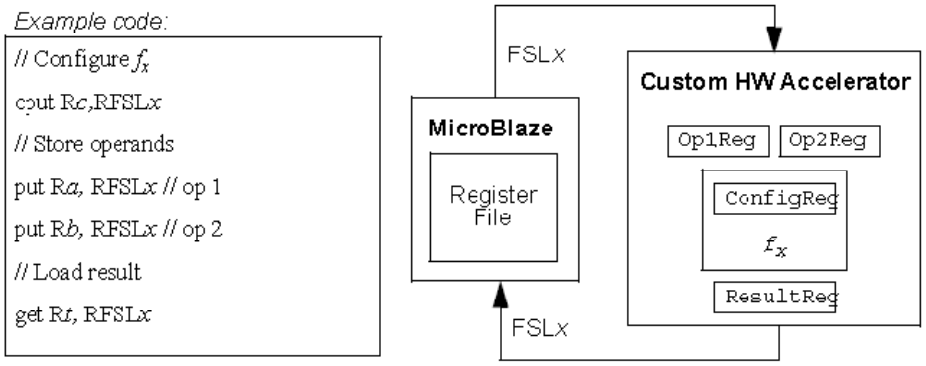
Disable hardware multiplier1	option	option	option	option	option
Hardware debug readable ESR and EAR	Yes	Yes	Yes	Yes	Yes
Processor Version Register (PVR)	-	option	option	option	option
Area or speed optimized	-	-	option	option	option
Hardware multiplier 64-bit result	-	-	option	option	option
LUT cache memory	-	-	option	option	option
Processor Local Bus (PLB) data side interface	-	-	-	option	option
Processor Local Bus (PLB) instruction side interface	-	-	-	option	option
Floating point conversion and square root instructions	-	-	-	option	option
Memory Management Unit (MMU)	-	-	-	option	option
Extended Fast Simplex Link (FSL) instructions	-	-	-	option	option

Fast Simplex Link

Όπως έχουμε αναφέρει και προηγούμενα μια από τις πιθανές συνδέσεις μεταξύ του μικροεπεξεργαστή microblaze και ενός περιφερειακού είναι η σύνδεση FSL. Κάθε επεξεργαστής microblaze μπορεί να διαχειριστεί έως και 16 διαφορετικές συνδέσεις τύπου FSL, κάθε μία από τις οποίες διαθέτει θύρα εισόδου και εξόδου. Έχουμε δηλαδή αμφίδρομη επικοινωνία μεταξύ του επεξεργαστή και του περιφερειακού, και επειδή κάθε σύνδεση είναι αφοσιωμένη σε ένα και μόνο περιφερειακό, αυτό ουσιαστικά, λειτουργεί σαν συνεπεξεργαστής με τον microblaze. Κύρια χρήση της σύνδεσης αυτής είναι για την κατασκευή κάποιου επιταχυντή για έναν αλγόριθμο, ή κομμάτι του, που εισάγει μεγάλη πολυπλοκότητα.

Η σύνδεση έχει μέγεθος μέχρι και 32-bit και προς τις δύο κατευθύνσεις και υλοποιείται με μια FIFO ουρά. Σε κάθε επικοινωνία μεταξύ του επεξεργαστή και του συνεπεξεργαστή, προς οποιαδήποτε κατεύθυνση, τα δεδομένα αποθηκεύονται σε μια ουρά με πλήθος στοιχείων που καθορίζεται από εμάς και μέγεθος λέξης 32-bit. Χρησιμοποιούμαι εντολές *get,put* για να στείλουμε και να λάβουμε δεδομένα από και προς το περιφερειακό αντίστοιχα, και μας δίνεται η δυνατότητα να επιλέξουμε μεταξύ blocking/non blocking data, blocking/non blocking control.

Πλεονέκτημα της σύνδεσης FSL είναι ότι συνδέεται απευθείας στο pipeline του επεξεργαστή και προσφέρει υψηλή απόδοση, γι' αυτό και είναι ιδανική για επέκταση των δυνατοτήτων του επεξεργαστή με χρήση επιταχυντών. Στην εικόνα που ακολουθεί μπορούμε να δούμε το σχηματικό διάγραμμα μιας τέτοιας υλοποίησης.



Βλέπουμε ότι η σύνδεση FSL αντλεί δεδομένα απευθείας από το αρχείο καταχωρητών του microblaze. Αυτό ισοδυναμεί με την επέκταση του συνόλου των εντολών του επεξεργαστή, αλλά ταυτόχρονα με το πλεονέκτημα ότι δεν επηρεάζεται το υπόλοιπο pipeline του επεξεργαστή από την ταχύτητα του επιταχυντή μας.

Floating Point Unit

Ο microblaze διαθέτει ειδική αριθμητική και λογική μονάδα για την εκτέλεση των λογικών και αριθμητικών πράξεων. Οι πράξεις γίνονται ακολουθώντας το πρότυπο *IEEE 754 standard.*

Η FPU (floating point unit) έχει την δυνατότητα για εκτέλεση λογικών, αριθμητικών πράξεων καθώς και μετατροπών.

- Αριθμητικές πράξεις
  - Πρόσθεση
  - Αφαίρεση
  - Πολλαπλασιασμός
  - Διαίρεση
  - Τετραγωνική ρίζα
- Λογικές πράξεις
  - Μικρότερο
  - Μεγαλύτερο
  - Μικρότερο ή ίσο
  - Μεγαλύτερο ή ίσο
  - Όχι ίσο
  - Αδύνατο να ταξινομηθεί (για αποτέλεσμα NaN)
- Μετατροπές
  - Από αριθμό σταθερής υποδιαστολής σε κινητής υποδιαστολής
  - Από αριθμό κινητής υποδιαστολής σε σταθερής.

Ο λόγος που η Xilinx εφοδιάζει τον microblaze με μονάδα κινητής υποδιαστολής είναι η πολυπλοκότητα που εισάγουν αυτές οι πράξεις όταν υλοποιούνται σε software. Υπάρχει όφελος έως και 40 φορές στην περίπτωση που υλοποιήσουμε σε hardware αυτές τις πράξει και απαλλάξουμε το λογισμικό από αυτό τον φόρτο.

Το περιβάλλον Xilinx EDK αναγνωρίζει αυτόματα τις πράξεις κινητής υποδιαστολής μέσα στο πρόγραμμά μας και εκτελεί τις πράξεις αυτές στο εξειδικευμένο υλικό. Αυτό βοηθά σε πιο αποδοτική υλοποίηση των προγραμμάτων μας όταν περιέχουν πολλές πράξεις με αριθμούς κινητής υποδιαστολής, όπως σε περιπτώσεις ψηφιακής επεξεργασίας σημάτων που είναι και η περίπτωσή μας.

Χαρακτηριστικά του Xilinx Spartan 3E

Για την υλοποίηση του συστήματός μας έχουμε διαθέσιμο ένα FPGA, Xilinx Spartan 3E και συγκεκριμένα την έκδοση XC3S500E. Στον παρακάτω πίνακα συνοψίζουμε τις δυνατότητες της συγκεκριμένης έκδοσης.

	XC3S500E
Πύλες συστήματος	500k
Λογικές μονάδες	10476
Block Ram Bits	360k
Κατανεμημένα Block Ram bits	73k
DCMs	4
Πολλαπλασιαστές	20

Ένα σημαντικό χαρακτηριστικό του Spartan 3E είναι η παρουσία 20 πολλαπλασιαστών 18x18 που διαθέτει υλοποιημένους σε υλικό και χρησιμοποιούνται αυτόματα από το fpga όταν αναγνωρίζει την πράξη του πολλαπλασιασμού, είτε αυτή λαμβάνει χώρα στο λογισμικό μας, είτε σε κάποιο περιφερειακό που έχουμε αναπτύξει σε γλώσσα περιγραφής υλικού. Αν λάβουμε υπ' όψη την καθυστέρηση που εισάγει η πράξη του πολλαπλασιασμού, και το μέγεθος του κυκλώματος του πολλαπλασιαστή καταλαβαίνουμε πόσο μπορούμε να ωφεληθούμε από την παρουσία των πολλαπλασιαστών.

Υλοποίηση του φίλτρου LoG στο EDK

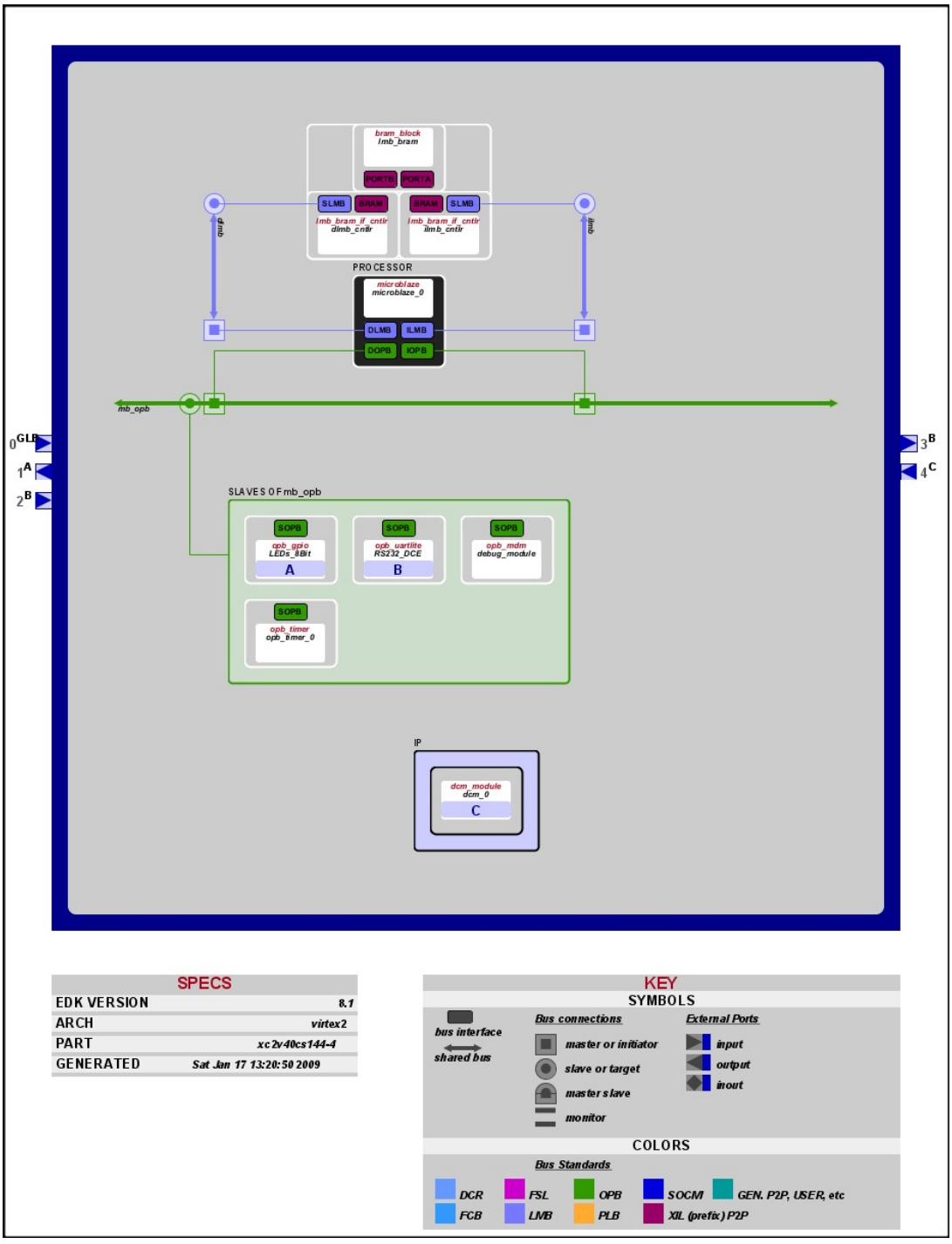
Όπως είδαμε στην θεωρητική περιγραφή των αλγορίθμων ανίχνευσης ακμών σε προηγούμενα κεφάλαια, αυτοί που παρουσιάζουν αξιόλογα αποτελέσματα και ταυτόχρονα έχουν αποδοτική υλοποίηση από πλευρά υπολογιστικού κόστους είναι τα ζωνοπερατά φίλτρα Laplacian of Gaussian και Bilevel Laplacian of Gaussian. Το δεύτερο αποτελεί μια προσέγγιση του πρώτου δίνοντας παραπλήσια αποτελέσματα αλλά με ακόμα χαμηλότερο υπολογιστικό κόστος. Σε αυτό το κεφάλαιο θα μελετήσουμε την υλοποίηση τους ως ενσωματωμένο λογισμικό στην πλακέτα Xilinx Spartan 3E με χρήση του περιβάλλοντος Embedded Development Kit 8.1.

Και για τους δύο αλγόριθμους εκτελούμε την πράξη της συνέλιξης για την εικόνα με ένα μητρώο συνέλιξης που αντιστοιχεί στο κάθε φίλτρο. Έπειτα καλούμαστε να αποφασίσουμε για τις ακμές. Στην υλοποίησή μας θα χρησιμοποιήσουμε μητρώα συνέλιξης μεγέθους 5x5 και θα μετρήσουμε τον χρόνο που χρειάζεται για να ολοκληρωθεί ο κάθε αλγόριθμος για δεδομένη εικόνα.

Παραμετροποίηση του συστήματος

Θα ξεκινήσουμε με την υλοποίηση του φίλτρου laplacian of gaussian. Στο περιβάλλον edk μας δίνεται η δυνατότητα να γράψουμε ένα πρόγραμμα στην γλώσσα προγραμματισμού C και κατόπιν ο compiler θα το μεταγγλωτίσει στην γλώσσα assembly του microblaze. Έχουμε στην διάθεσή μας 20 block rams πάνω στο FPGA από 2 Kb ή κάθε μία, δηλαδή σύνολο 40 Kb για να αποθηκεύσουμε τα δεδομένα και το πρόγραμμα. Εναλλακτικά για τα δεδομένα μπορεί αν χρησιμοποιηθούν και τα 16MB της εξωτερικής DDR Ram αλλά αυτό θα μειώσει την απόδοση του συστήματος και γι αυτό θα το αποφύγουμε. Για την προσομοίωση της κάμερας η οποία παρέχει τις εικόνες προς ανίχνευση ακμών θα χρησιμοποιήσουμε την σειριακή σύνδεση μεταφέροντας εικόνες από έναν υπολογιστή στο FPGA. Για μέτρηση της απόδοσης θα μας βοηθήσει ένα χρονόμετρο υλοποιημένο σε hardware και παρέχεται από την Xilinx. Το τελικό σύστημα που προκύπτει παρουσιάζεται στο block diagram της εικόνας που ακολουθεί.

Στο σύστημα μπορούμε να διακρίνουμε τον τοπικό διάυλο εσωτερικά του επεξεργαστή (Processor Local Bus) με μπλέ χρώμα, τον πράσινο δίαυλο OPB και τα συνδεδεμένα επάνω του περιφερειακά RS232_DCE (σειριακή θύρα), Leds8bit (φωτοδίοδοι), Debug_Module (μονάδα εκσφαλμάτωσης), opb_timer (χρονόμετρο).



Εικόνα 29. Σύστημα κατασκευασμένο στο Xilinx EDK με τον microblaze και τα περιφερειακά.

Περιγραφή του αλγόριθμου

Το σύστημα που υλοποιήσαμε διαχειρίζεται μια εικόνα μεγέθους QCIF. Όλα τα μεγαλύτερα πρότυπα είναι απλά πολλαπλάσια του QCIF και γι' αυτό αποτελεί κατάλληλη επιλογή. Επίσης το μέγεθος του επιτρέπει την επεξεργασία του στο περιορισμένο μέγεθος μνήμης που προσφέρει το FPGA για την υλοποίηση του συστήματός μας. Μεγαλύτερες εικόνες μπορούν να τμηματοποιηθούν σε τμήματα μεγέθους QCIF και το περιφερειακό να τροφοδοτηθεί περισσότερες από μια φορές με δεδομένα μέχρι την περάτωση του υπολογισμού των ακμών για όλη την εικόνα.

Το φίλτρο laplacian of Gaussian που θα υλοποιήσομε θα έχει διαστάσεις 5x5 και τυπική απόκλιση *σ=0.751.* Το μητρώο συνέλιξης παρουσιάζεται στο πίνακα που ακολουθεί

0.0059	*0.0417*	*0.0744*	*0.0417*	*0.0059*
0.0417	*0.1323*	*-0.0460*	*0.1323*	*0.0417*
0.0744	*-0.0460*	*-1.0005*	*-0.0460*	*0.0744*
0.0417	*0.1323*	*-0.0460*	*0.1323*	*0.0417*
0.0059	*0.0417*	*0.0744*	*0.0417*	*0.0059*

Θα προχωρήσουμε σε συνέλιξή του μητρώου συνέλιξης του laplacian of Gaussian φίλτρου με την εικόνα μεγέθους QCIF που ακολουθεί.

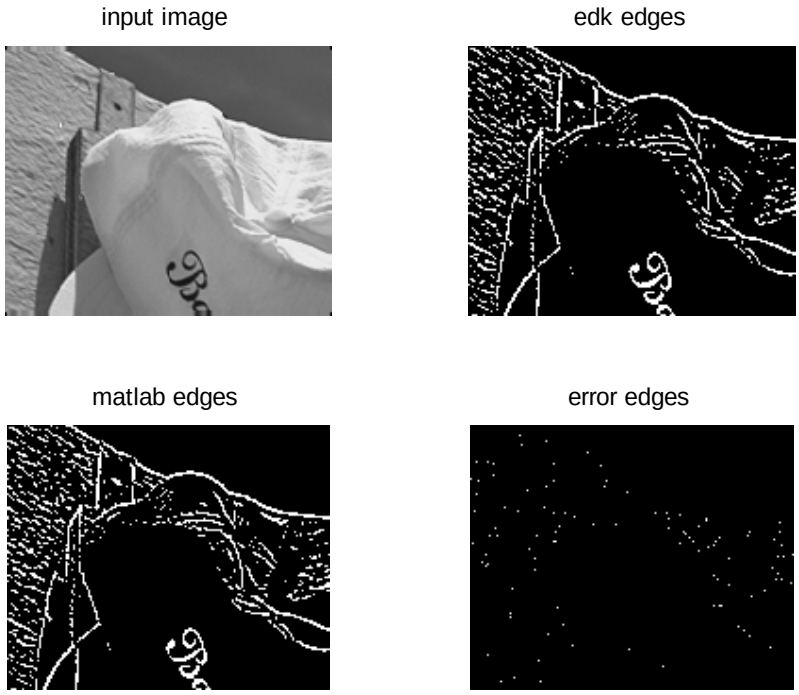
Εικόνα 30. Εικόνα σε CIF format προς επεξεργασία.

Έχουμε υλοποιήσει τον αλγόριθμο τόσο στο περιβάλλον Xilinx EDK όσο και στο matlab για να μπορέσουμε να έχουμε καλή παρουσίαση, σύγκριση και επιβεβαίωση των αποτελεσμάτων. Η εικόνα θα μεταφερθεί από το matlab στο FPGA μέσω του σειριακού καλωδίου που θα προσομοιώσει την κάμερα. Μετά την λήψη της εικόνας εφαρμόζεται το φίλτρο laplacian of Gaussian. Αμέσως πριν την έναρξη των υπολογισμών που αφορούν τον αλγόριθμο ενεργοποιείται ένα χρονόμετρο στο σύστημα και απενεργοποιείται με το τέλος τους. Με αυτόν τον τρόπο θα καταφέρουμε να μετρήσουμε την επίδοση του συστήματος και να την συγκρίνουμε με τις επόμενες υλοποιήσεις. Με το πέρας των υπολογισμών η εικόνα που περιέχει πλέον τα αποτελέσματα της εύρεσης ακμών μεταφέρεται και πάλι πίσω στο matlab μέσω της σειριακής θύρας για σύγκριση με τα θεωρητικά αποτελέσματα της ανίχνευσης ακμών που έχουν υπολογιστεί με την χρήση του αρχικού κώδικα που έχουμε γράψει στο matlab.

Ο κώδικας των προγραμμάτων σε C και σε matlab μπορούν να βρεθούν στο παράρτημα.

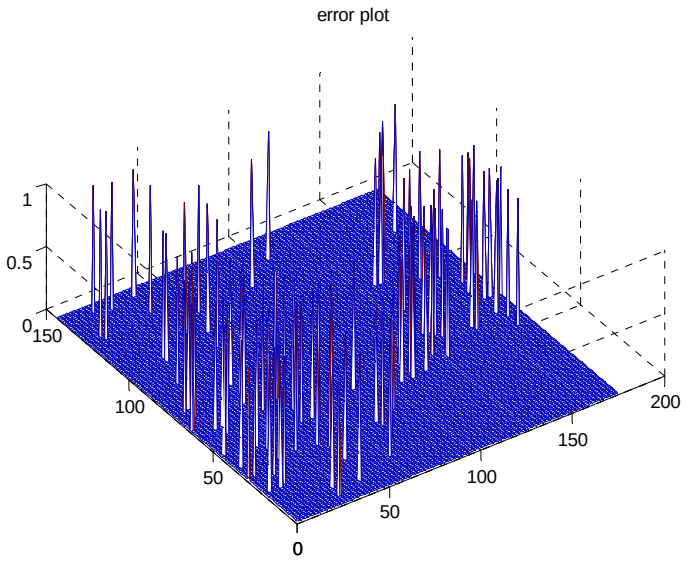
Εκτέλεση και αποτελέσματα

Με την εφαρμογή του αλγόριθμου λαμβάνουμε τα αποτελέσματα της εικόνας που ακολουθεί:



Εικόνα 31. Εκτέλεση του αλγόριθμου ανίχνευσης ακμών LoG σε λογισμικό του EDK

Η πρώτη είναι η αρχική εικόνα, με την εικόνα που επεξεργαστήκαμε στο FPGA να βρίσκεται δεξιά της και την εικόνα που μας δίνει το matlab από κάτω. Αφαιρώντας αυτές τις δύο μεταξύ τους παίρνουμε μια οπτικοποίηση του σφάλματος στην τέταρτη εικόνα. Βλέπουμε ότι υπάρχουν ελάχιστα εσφαλμένα εικονοστοιχεία μεταξύ της εικόνας που επεξεργαστήκαμε στο FPGA και αυτής στο matlab. Ο λόγος εμφάνισης τους είναι κάποια διαφορά στις στρογγυλοποιήσεις που δίνουν τιμές πολύ κοντά στο κατώφλι που ορίζουμε για την ακμή. Ακολουθεί και μια εκτύπωση του σφάλματος.

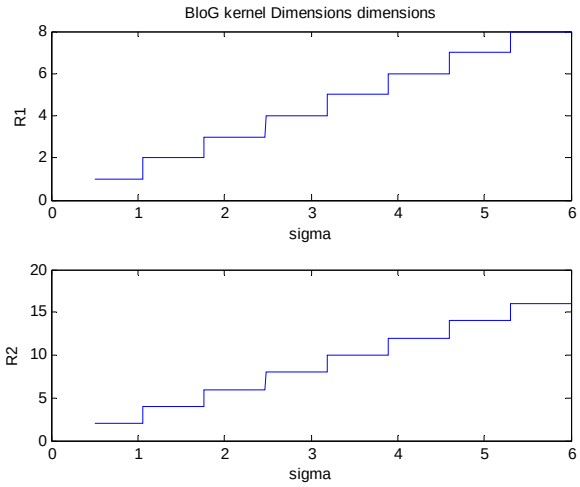


Εικόνα 32. Εκτύπωση σφάλματος μετά από σύγκριση με το matlab.

Για τον υπολογισμό των ακμών χρειάστηκαν 128.668 παλμοί ρολογιού του επεξεργαστή, ο οποίος τρέχει στα 50MHz. Ο χρόνος αυτός αντιστοιχεί σε μια εικόνα μεγέθους CIF την οποία επεξεργαζόμαστε τμηματικά, διαιρώντας την σε 4 υπό-εικόνες QCIF.

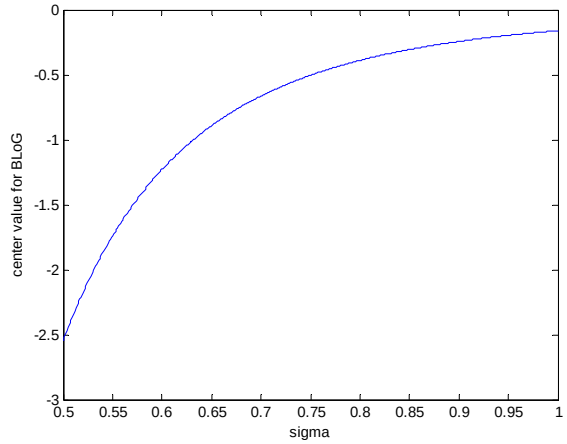
Υλοποίηση του φίλτρου BLoG στο EDK

Όπως είδαμε και προηγούμενα η προσέγγιση του laplacian of gaussian φίλτρου με το BloG θα μας οδηγήσει σε ένα διαφορετικό μητρώο συνέλιξης. Η τιμή της τυπικής απόκλισης που θα διαλέξουμε θα καθορίσει την διάσταση του φίλτρου καθώς και τις τιμές των συντελεστών του φίλτρου. Στο γράφημα που ακολουθεί μπορούμε να δούμε τις διαστάσεις που προκύπτουν για την εσωτερική και την εξωτερική ακτίνα του φίλτρου ανάλογα με την τιμή της τυπικής απόκλισης που θεωρήσαμε.



Εικόνα 33. Εσωτερική και εξωτερική ακτίνα του BloG φίλτρου, συναρτήσει της τυπικής απόκλισης.

Για μικρές τιμές της τυπικής απόκλισης (σ) που μας ενδιαφέρουν περισσότερο (προκειμένου να ανιχνεύσουμε πιο λεπτομερείς ακμές), οι τιμές για τον συντελεστή στην εσωτερική ακτίνα του φίλτρου παρουσιάζεται στο σχήμα που ακολουθεί.



Εικόνα 34. Κεντρική τιμή του BloG φίλτρου, συναρτήσει της τυπικής απόκλισης.

Οι συντελεστές μεταξύ εσωτερικής και εξωτερικής ακτίνας καθορίζονται από την τιμή των συντελεστών στην εσωτερική ακτίνα σε συνάρτηση με την διάσταση του φίλτρου από την σχέση,

$$F_2 = \frac{-F_1 \sum\sum_{x^2+y^2 \le R_1^2} 1}{\sum\sum_{R_1^2 < x^2+y^2 \le R_2^2} 1}$$

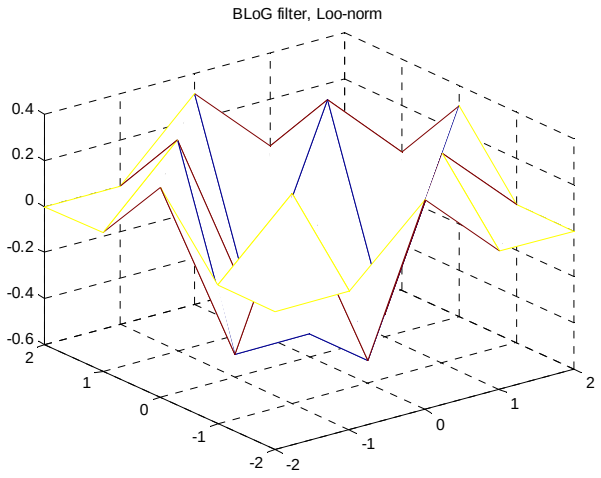
Από το παραπάνω σχήμα βλέπουμε ότι είναι εύκολο να επιλέξουμε τιμές για τους συντελεστές που είναι δυνατόν να παρασταθούν σαν δυνάμεις του 2. Αυτό είναι ένα πολύ ισχυρό πλεονέκτημα που μπορεί να μας οδηγήσει σε μια υλοποίηση απουσία πολλαπλασιαστών, με ασφαλώς μικρότερο κρίσιμο μονοπάτι, μικρότερο συνολικά hardware και χαμηλότερη κατανάλωση ισχύος.

Στην συγκεκριμένη υλοποίηση θα προσεγγίσουμε το φίλτρο laplacian of Gaussian με τυπική απόκλιση σ=0.751, χρησιμοποιώντας την L_∞ νόρμα που οδηγεί στους συντελεστές του φίλτρου BLOG στις παρακάτω τιμές,

$$F_1 = -0.5 = 2^{-1}$$
$$F_2 = \frac{5}{8}0.5 = \frac{(2^2 + 2^0)2^{-1}}{2^3} = 2^{-2} + 2^{-4}$$

Το φίλτρο που προκύπτει είναι το εξής

0	0	0.3125	0	0
0	0.3125	-0.5000	0.3125	0
0.3125	-0.5000	-0.5000	-0.5000	0.3125
0	0.3125	-0.5000	0.3125	0
0	0	0.3125	0	0

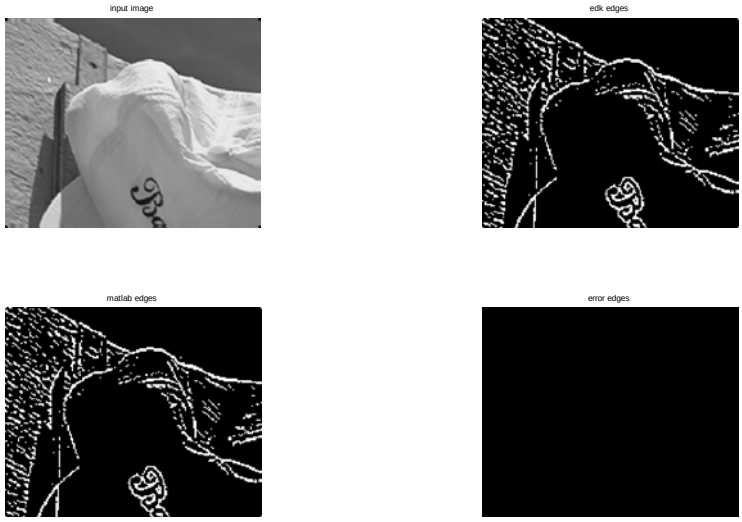


Εικόνα 35. Η τελική μορφή του Bilevel LoG φίλτρου.

Εκτέλεση και αποτελέσματα

Η διαδικασία που ακολουθήσαμε είναι ίδια με αυτή στην περίπτωση του laplacian of gaussian. Έχουμε υλοποιήσει τον αλγόριθμο στο EDK αποκλειστικά σε software και θα συγκρίνουμε τα αποτελέσματά του με το matlab.

Εκτελώντας τον αλγόριθμο στο EDK το αποτέλεσμα συμπίπτει απόλυτα με την υλοποίηση στο matlab. Στο σχήμα που ακολουθεί βλέπουμε ότι δεν υπάρχει το παραμικρό σφάλμα στην ανίχνευση των ακμών. Αυτό συμβαίνει εδώ καθώς δεν υπάρχει διαφορά μεταξύ των τιμών του φίλτρου στο matlab και το EDK και δεν εισάγονται σφάλματα στρογγυλοποίησης.



Εικόνα 36. Εφαρμογή του BloG φίλτρου σε λογισμικό του EDK.

Για την εκτέλεση του αλγόριθμου χρειάστηκαν 128.668 παλμοί ρολογιού και ο επεξεργαστής είναι χρονισμένος στα 50MHz.

Παρατηρούμε ότι είναι ακριβώς ο ίδιος αριθμός παλμών με την υλοποίηση του laplacian of Gaussian παρόλο που χρησιμοποιούμε μόνο δύο πολλαπλασιαστές. Ο λόγος είναι ότι και στις δύο περιπτώσεις η πλακέτα κάνει χρήση των ενσωματωμένων πολλαπλασιαστών που διαθέτει και εκτελεί τις πράξεις σε 1 παλμό ρολογιού. Οπότε τελικά ο χρόνος εκτέλεσης του αλγόριθμου εξαρτάται από το loop overhead που εισάγουν οι επαναληπτικοί βρόχοι με τους οποίους σαρώνουμε την εικόνα. Ουσιαστικά δηλαδή ακόμα και στο software που γράφουμε η πλακετα Spartan εισάγει κάποια επιτάχυνση από εξειδικευμένο υλικό. Σε μια υλοποίηση εξ ολοκλήρου σε software, όπως σε ένα pc, ο χρόνος υπολογισμού θα παρουσίαζε μεγαλύτερη διαφοροποίηση σε χρόνο εκτέλεσης.

Επιτάχυνση των αλγόριθμων με χρήση hardware

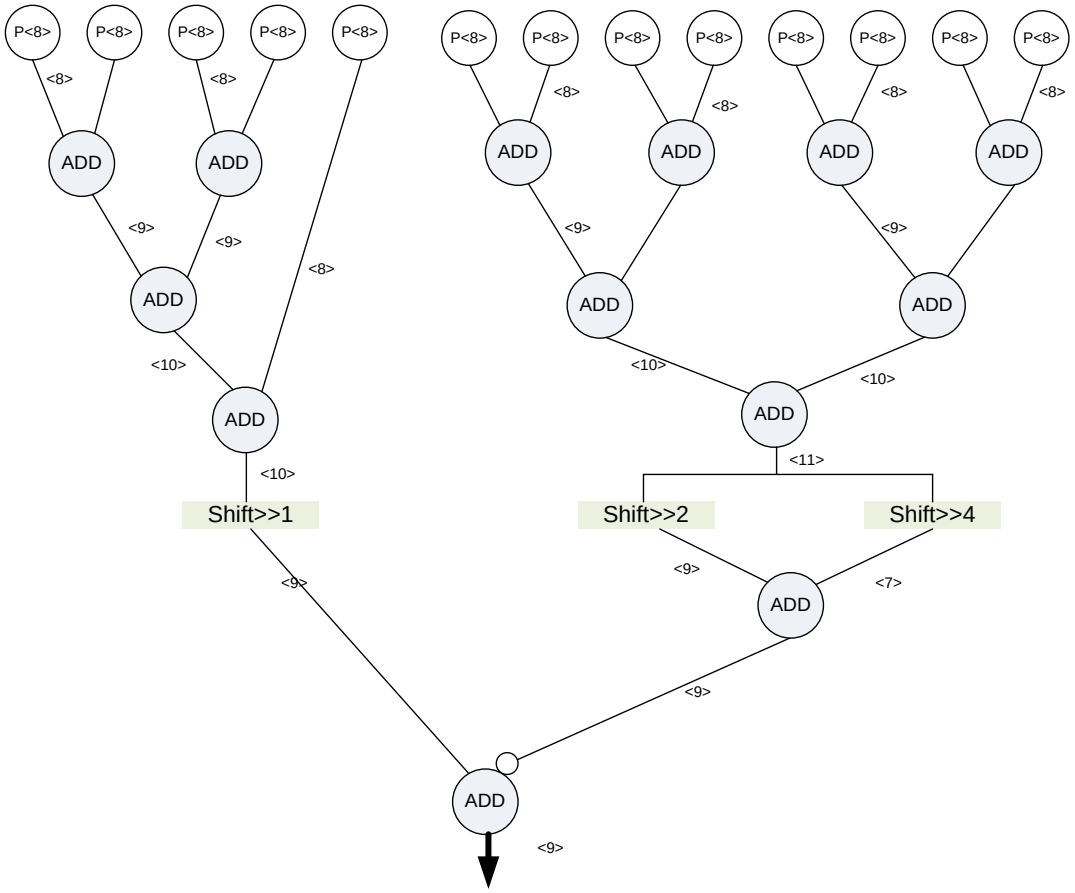
Όπως είδαμε οι δύο αλγόριθμοι που υλοποιήσαμε εισάγουν μεγάλη πολυπλοκότητα καθώς για τον υπολογισμό ενός εικονοστοιχείου και μόνο απαιτούν πολυάριθμες προσθέσεις και πολλαπλασιασμούς. Ο χρόνος υπολογισμού που μετρήσαμε παραπάνω μπορεί να μην φαίνεται απαγορευτικός αλλά αν λάβουμε υπ' όψη μας το μικρό μέγεθος την εικόνας που επεξεργαζόμαστε (μόλις 176x144 pixels) καθώς και την επιτάχυνση λόγω των ενσωματωμένων πολλαπλασιαστών καταλαβαίνουμε ότι σε μια εφαρμογή με μεγαλύτερο κάδρο, ή και ακόμα περισσότερο όταν η ανίχνευση ακμών θα αποτελεί ένα μόνο στάδιο επεξεργασίας, υπάρχει ανάγκη για επιτάχυνση ενός τέτοιου αλγόριθμου. Η ανίχνευση ακμών αποτελεί την «αρχή» για πολλές ακόμα εφαρμογές στην επεξεργασία εικόνας και την υπολογιστική όραση.

Υλοποίηση του φίλτρου BLOG σε hardware

Είδαμε παραπάνω ότι η προσέγγιση του laplacian of gaussian φίλτρου από το Bilevel LoG μας δίνει την δυνατότητα να μειώσουμε μόλις σε δύο τους πολλαπλασιαστές που χρειαζόμαστε για οποιοδήποτε μέγεθος μητρώου συνέλιξης (που καθορίζεται από την τυπική απόκλιση του LoG που θέλουμε να προσεγγίσουμε). Αν πάλι επιλέξουμε προσεκτικά την τιμή της τυπικής απόκλισης μπορούμε να δημιουργήσουμε συντελεστές που γράφονται σαν δυνάμεις του 2. Αυτό θα μας απαλλάξει από την χρήση πολλαπλασιαστών και το μόνο που θα χρειαστεί είναι ένα δένδρο αθροιστών και κάποιες ολισθήσεις. Αυτό θα μειώσει δραστικά την μέγιστη καθυστέρηση του datapath καθώς οι πολλαπλασιαστές εισάγουν πολύ μεγάλη καθυστέρηση, ειδικά για μεγάλα μήκη λέξεων.

Υλοποίηση του μονοπατιού δεδομένων

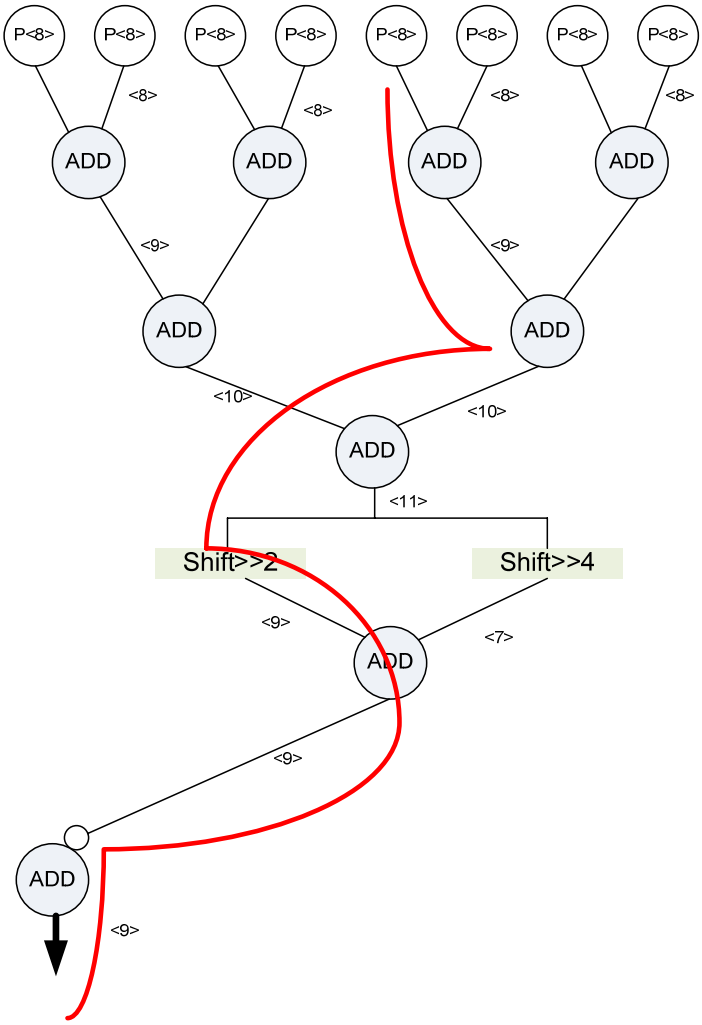
Πρώτο στάδιο για την ανάπτυξη του hardware που θα υπολογίζει τα pixel εξόδου του αλγόριθμου, είναι η υλοποίηση μιας αριθμητικής μονάδας που με δεδομένα τα pixel μιας 5x5 περιοχής της εικόνας θα δίνει το αποτέλεσμα για το κεντρικό pixel που αντιστοιχεί στην εικόνα εξόδου όπου αναπαριστούμε την ύπαρξη ακμών με δυαδικές τιμές. Για το μητρώο συνέλιξης που έχουμε επιλέξει έχουμε δυο ομάδες εικονοστοιχείων. Μία θα πολλαπλασιαστεί με F_1=0.5 και αποτελείται από 5 pixel και μία που θα πολλαπλασιαστεί με F_2=-0.3125 και αποτελείται από 8 pixel. Βλέπουμε ότι από τα συνολικά 25 εικονοστοιχεία, 13 έχουν μη μηδενικές τιμές πράγμα που μειώνει την πολυπλοκότητα ακόμα περισσότερο καθώς θα χρειαστούμε ένα μικρότερο δένδρο αθροιστών. Αυτό σε συνάρτηση και με την απουσία πολλαπλασιαστών οδηγεί σε μια πολύ οικονομική υλοποίηση από πλευράς υλικού, αλλά και μικρότερο κρίσιμο μονοπάτι καθώς δεν χρησιμοποιούμε πολλαπλασιαστές που εισάγουν μεγάλη καθυστέρηση. Το datapath εικονίζεται στο σχήμα που ακολουθεί.



Εικόνα 37. Το μονοπάτι δεδομένων για το BloG φίλτρο.

Όπως είπαμε και προηγούμενα από το μονοπάτι απουσιάζουν τελείως οι πολλαπλασιαστές και έχουν αντικατασταθεί από ολισθήσεις. Στις ακμές που συμβολίζουν τα σήματα του συστήματος υποδεικνύουμε μέσα σε αγκύλες το μέγεθος της λέξης που απαιτείται ώστε να αποφύγουμε την υπερχείλιση. Το μεγαλύτερο μήκος λέξης φτάνει τα 11 bit και αυτό θα είναι το μήκος λέξης για όλο το μονοπάτι δεδομένων. Η έξοδος θα δίνεται σε 9 ψηφία που είναι αρκετά για την αναπαράσταση της εξόδου καθώς από την φύση του ο αλγόριθμος δεν παράγει τιμές έξω από τα όρια -255< X < 255 .

Βλέπουμε ότι το κρίσιμο μονοπάτι περιλαμβάνει 5 επίπεδα αθροιστών και έναν αντιστροφέα, καθώς οι ολισθήσεις δεν εισάγουν καθυστέρηση. Ένα από τα κρίσιμα μονοπάτια συμβολίζεται στην εικόνα που ακολουθεί με κόκκινο χρώμα.



Εικόνα 38. Επισήμανση του μονοπατιού μέγιστη καθυστέρησης.

Στην περίπτωση που θα κάναμε χρήση και ενός πολλαπλασιαστή, τότε στο κρίσιμο μονοπάτι θα προσθέταμε και αυτόν αντί της ολίσθησης. Λόγω του ότι οι συντελεστές του φίλτρου παίρνουν πολύ μικρές δεκαδικές τιμές θα χρειαζόμασταν μεγάλο μήκος λέξης για να τις αναπαραστήσουμε σε αριθμητική σταθερής υποδιαστολής. Ίσως και 15 bits για μεγάλου μεγέθους πυρήνες συνέλιξης. Αυτό θα εκτόξευε την καθυστέρηση καθώς για τον πολλαπλασιασμό της 9bit εξόδου με ένα συντελεστή 15 bit θα προέκυπτε έξοδος 24 bit, και ο αθροιστής θα εισήγαγε καθυστέρηση περίπου ίση με 14 αθροιστές αν αναλογιστούμε ότι ένας πολλαπλασιαστής κατασκευάζεται με διαδοχικούς αθροιστές. Ακόμα και με την χρήση πολλαπλασιαστών carry save η καθυστέρηση θα ήταν μεγάλη για το μονοπάτι δεδομένων, καθώς επίσης θα καταλάμβανε και μεγαλύτερο χώρο πάνω στο FPGA. Αυτό με την σειρά του θα αυξήσει και την κατανάλωση ενέργειας καθώς μπορούμε να ισχυριστούμε ότι η κατανάλωση ενέργειας αυξάνει με την αύξηση της επιφάνειας που καταλαμβάνει μια υλοποίηση. Στο επόμενο κεφάλαιο όπου θα εξετάσουμε την υλοποίηση του laplacian of

gaussian φίλτρου θα εξετάσουμε λεπτομερώς τις καθυστερήσεις που εισάγουν οι δύο υλοποιήσεις.

Αποθήκευση και μεταφορά δεδομένων στον συνεπεξεργαστή.

Ο νέος συνεπεξεργαστής και συγκεκριμένα το μονοπάτι δεδομένων που σχεδιάζουμε πρέπει να τροφοδοτηθεί με δεδομένα. Αυτό είναι ένα πολύ κρίσιμο σημείο της σχεδίασης λόγω της φύσης του αλγόριθμου και του γεγονότος ότι η μεταφορά δεδομένων από την μνήμη σε κάποιο περιφερειακό μπορεί να αποτελέσει την μεγαλύτερη καθυστέρηση στην εκτέλεση των υπολογισμών καθιστώντας το σημείο αυτό σε λαιμό του μπουκαλιού.

Για την μεταφορά των δεδομένων έχουμε επιλέξει την σύνδεση FSL (fast simplex link). Με αυτό τον τρόπο εξασφαλίζουμε ανεξαρτησία στην επικοινωνία επεξεργαστή-συνεπεξεργαστή από τα άλλα περιφερειακά καθώς αποτελεί ξεχωριστό δίαυλο επικοινωνίας από τον κύριο δίαυλο του συστήματος και έτσι δεν θα έχουμε καθυστερήσεις λόγω διαιτησίας μεταξύ των υπολοίπων περιφερειακών.

Η δεύτερη σημαντική σχεδιαστική απόφαση που πρέπει να πάρουμε αφορά τον χρόνο ζωής των δεδομένων στον συνεπεξεργαστή προτού αντικατασταθεί από νέα. Αυτό συμβαίνει γιατί το κάθε pixel της εικόνας εισόδου χρησιμοποιείται παραπάνω από μία φορές για τον υπολογισμό της συνέλιξης με το μητρώο μας. Στην περίπτωσή μας για πυρήνα συνέλιξης μεγέθους 5, κάθε pixel θα χρησιμοποιηθεί 25 φορές, όσες και τα στοιχεία του πυρήνα.

a1	a2	a3	a4	a5	a6	a7	a8	a9
b1	b2	b3	b4	b5	b6	b7	b8	b9
c1	c2	c3	c4	c5	c6	c7	c8	c9
d1	d2	d3	d4	d5	d6	d7	d8	d9
e1	e2	e3	e4	e5	e6	e7	e8	e9
f1	f2	f3	f4	f5	f6	f7	f8	f9
g1	g2	g3	g4	g5	g6	g7	g8	g9
h1	h2	h3	h4	h5	h6	h7	h8	h9
i1	i2	i3	i4	i5	i6	i7	i8	i9

Στην παραπάνω εικόνα μπορούμε να διακρίνουμε τις οριακές θέσεις που μπορεί να τοποθετηθεί το μητρώο συνέλιξης και το pixel *e5* να έχει συμμετοχή. Βλέπουμε ότι υπάρχουν 5 δυνατές μετατοπίσεις τόσο κατά τον οριζόντιο όσο και κατά τον κάθετο άξονα, συνολικά 25 διαφορετικές θέσεις.

Λόγω του ότι για την μεταφορά δεδομένων από την μνήμη του συστήματος προς τον συνεπεξεργαστή απαιτούνται 4 παλμοί ρολογιού χρησιμοποιώντας την σύνδεση FSL η αποθήκευση του κάθε pixel για όλο τον χρόνο ζωής του στον συνεπεξεργαστή θα οδηγήσει σε σημαντική οικονομία χρόνου, δηλαδή μεγαλύτερη επιτάχυνση του αλγόριθμου. Για αυτό τον

λόγο θα χρησιμοποιήσουμε τις block ram που διαθέτει η πλακέτα Xilinx Spartan 3E στο ολοκληρωμένο του fpga και είναι ταχύτατες με μόλις ένα παλμό ρολογιού για ανάκτηση δεδομένων μετά την τοποθέτηση της διεύθυνσης στην πόρτα διεύθυνσης. Κάθε γραμμή της εικόνας έχει 176 εικονοστοιχεία τα οποία θα υποθηκεύσουμε σε μια block ram.

Για να ξεκινήσουμε να υπολογίζουμε εικονοστοιχεία εξόδου θα χρειαστούμε δεδομένα από 5 γραμμές της εικόνας. Είμαστε δηλαδή αναγκασμένοι να χρησιμοποιήσουμε 5 block ram, μία για κάθε γραμμή της εικόνας. Αφού θα υπολογίσουμε όλες τις δυνατές συνελίξεις με τα διαθέσιμα εικονοστοιχεία, τα στοιχεία της πρώτης γραμμής πλέον δεν θα χρειαστούν ξανά, οπότε μπορούμε να τα αντικαταστήσουμε με επόμενη γραμμή και να συνεχίσουμε με τον ίδιο τρόπο. Με αυτόν τον τρόπο αποφεύγουμε τις άσκοπες μεταφορές δεδομένων. Έχουμε ομαδοποιήσει τα εικονοστοιχεία ανά 4 στις διευθύνσεις της μνήμης αλλά δεν αποτελεί πρόβλημα καθώς μπορούμε να ανακαλέσουμε 2 από την κάθε μία οπότε έχουμε την δυνατότητα να υπολογίσουμε 4 εικονοστοιχεία εξόδου ταυτόχρονα, δηλαδή επιταχύνουμε και την διαδικασία υπολογισμού. Αυτό παρουσιάζεται στην εικόνα που ακολουθεί, όπου μπορούμε να δούμε τις θέσεις του μητρώου συνέλιξης ώστε να υπολογίσουμε τα 4 αντίστοιχα εικονοστοιχεία εξόδου.

a1	a2	a3	a4	a5	a6	a7	a8
b1	b2	b3	b4	b5	b6	b7	b8
c1	c2	c3	c4	c5	c6	c7	c8
d1	d2	d3	d4	d5	d6	d7	d8
e1	e2	e3	e4	e5	e6	e7	e8

a1	a2	a3	a4	a5	a6	a7	a8
b1	b2	b3	b4	b5	b6	b7	b8
c1	c2	c3	c4	c5	c6	c7	c8
d1	d2	d3	d4	d5	d6	d7	d8
e1	e2	e3	e4	e5	e6	e7	e8

a1	a2	a3	a4	a5	a6	a7	a8
b1	b2	b3	b4	b5	b6	b7	b8
c1	c2	c3	c4	c5	c6	c7	c8
d1	d2	d3	d4	d5	d6	d7	d8
e1	e2	e3	e4	e5	e6	e7	e8

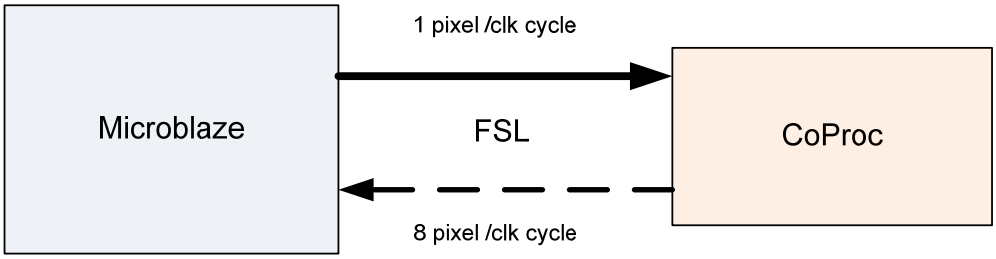
a1	a2	a3	a4	a5	a6	a7	a8
b1	b2	b3	b4	b5	b6	b7	b8
c1	c2	c3	c4	c5	c6	c7	c8
d1	d2	d3	d4	d5	d6	d7	d8
e1	e2	e3	e4	e5	e6	e7	e8

Εικόνα 39. Ταυτόχρονος υπολογισμός 4 pixel εξόδου.

Το μέγεθος του διαύλου FSL είναι 32bit, καθώς και το μήκος της λέξης στις block ram. Αυτό, σε συνάρτηση με το μέγεθος των 8 bit για τα εικονοστοιχεία grayscale εικόνας που χρησιμοποιούμε μας δίνουν την δυνατότητα να τα ομαδοποιήσουμε ανά 4 σε κάθε διεύθυνση της block ram, καθώς επίσης και να τα μεταφέρουμε ανά τέσσερα από την μνήμη του κύριου

επεξεργαστή. Ουσιαστικά δηλαδή μπορούμε να μεταφέρουμε 4 pixel / 4 παλμούς = 1 pixel / παλμό.

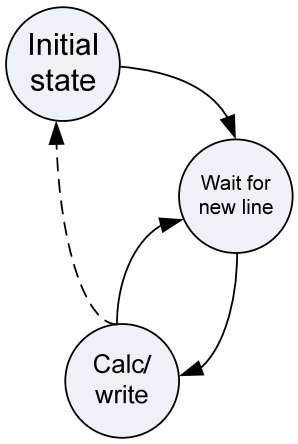
Όσον αφορά την έξοδο του συνεπεξεργαστή αυτή δεν χρειάζεται καν να είναι 8 bit. Καθώς σκοπός μας είναι να ανιχνεύσουμε την ύπαρξη ή όχι ακμών, η απάντηση μπορεί να δοθεί δυαδικά. Το '1' θα συμβολίζει την ύπαρξη ακμής ενώ το '0' όχι. Με αυτό τον τρόπο μπορούμε να μεταφέρουμε 32 pixel από τον συνεπεξεργαστή στον κύριο επεξεργαστή, δηλαδή 8 pixel / παλμό ρολογιού. Η επικοινωνία μεταξύ κύριου και συνεπεξεργαστή φαίνεται στο σχήμα που ακολουθεί



Εικόνα 40. Σύνδεση επεξεργαστή – συνεπεξεργαστή.

Finite state machine για τον έλεγχο του datapath και των μνημών

Το σύστημα που σχεδιάσαμε για να λειτουργήσει πρέπει αρχικά να τροφοδοτηθεί με δεδομένα από τον επεξεργαστή, έπειτα πρέπει να ακολουθήσει ο υπολογισμός των pixel εξόδου και τέλος η επιστροφή τους στον επεξεργαστή για περαιτέρω χρήση. Οι διαφορετικές αυτές καταστάσεις ελέγχονται από μια αλγοριθμική μηχανή πεπερασμένων καταστάσεων. Με αυτήν καθορίζουμε τα σήματα ελέγχου των μνημών, τις διευθύνσεις εγγραφής και ανάγνωσης, και την λειτουργία του datapath.



Εικόνα 41. State machine για έλεγχο του datapath.

Στο παραπάνω σχήμα παρατηρούμε τις καταστάσεις του FSM που χρησιμοποιούμε.

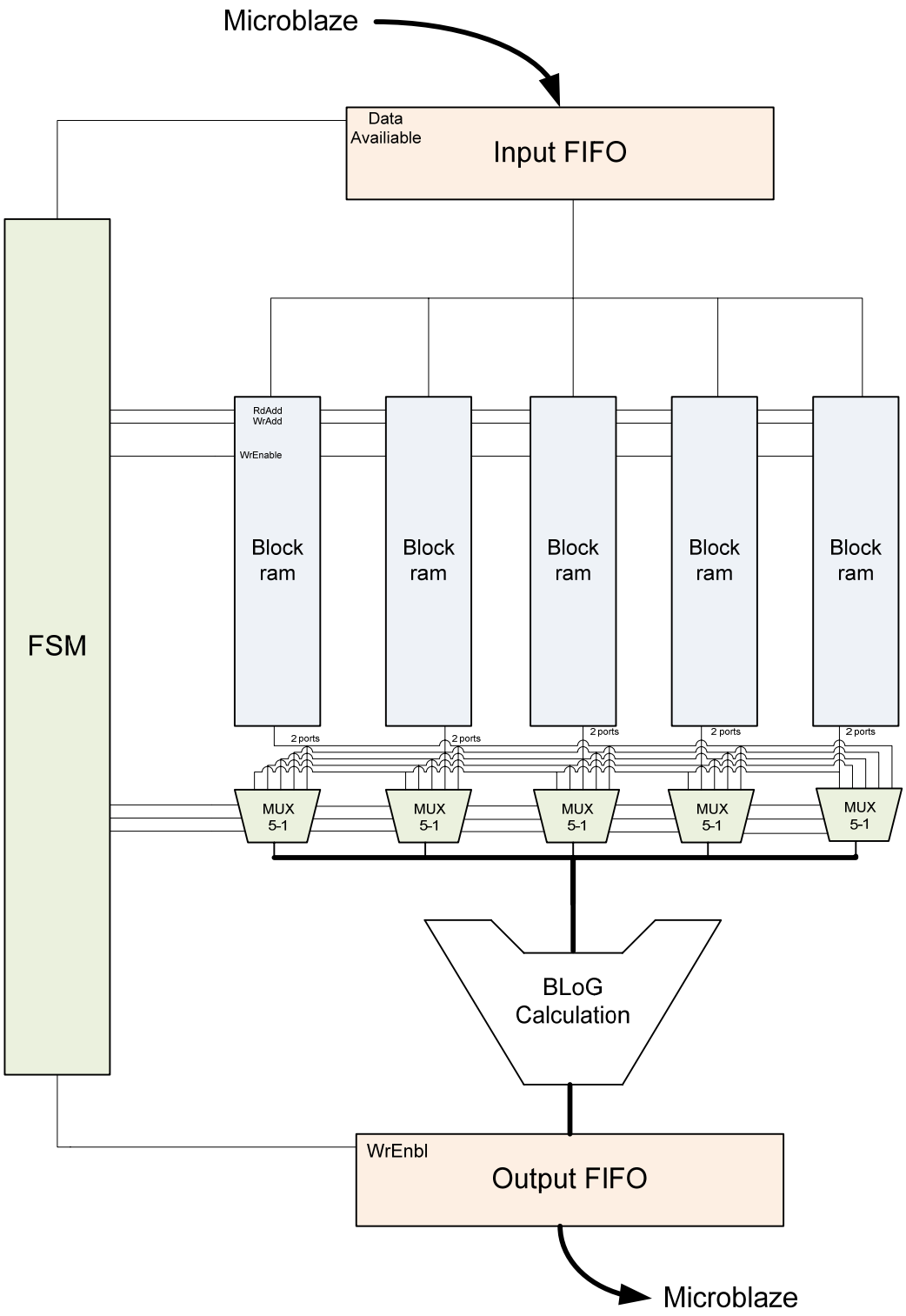
1. *Αρχική κατάσταση (Initial State)*. Όλες οι μνήμες είναι άδειες, ή πρέπει να αντικατασταθούν για να ξεκινήσει η επεξεργασία μιας νέας εικόνας.
2. *Μια γραμμή πρέπει να αντικατασταθεί (wait for new line)*. Είτε πρόκειται για την τελευταία γραμμή της αρχικοποίησης, είτε για την πιο παλαιά γραμμή μετά από έναν υπολογισμό.
3. *Υπολογισμός* των pixel εξόδου και εγγραφή στην ουρά εξόδου προς τον κύριο επεξεργαστή *(Calc-Write)*.

Αρχικά όλες οι μνήμες είναι άδειες από δεδομένα, όταν αυτά είναι διαθέσιμα τροφοδοτούν τις εισόδους των block ram. Όταν συμπληρωθούν οι 4 πρώτες μνήμες και ακόμα μια απομένει, μεταφερόμαστε στην επόμενη κατάσταση, όπου περιμένουμε την τελευταία μνήμη να συμπληρωθεί. Το ποια μνήμη τροφοδοτείται κάθε στιγμή καθορίζεται από πολυπλέκτες οι οποίοι ελέγχονται με κατάλληλα σήματα από την μηχανή καταστάσεων. Όταν πλέον και οι πέντε μνήμες διαθέτουν ανανεωμένα δεδομένα προχωράμε στον υπολογισμό των pixel εισόδου. Ό υπολογισμός γίνεται ταυτόχρονα για 4 pixel εξόδου και σε κάθε παλμό απασχολούμε συνολικά 10 διευθύνσεις μνήμης, 2 από κάθε block ram. Κατά αυτό τον τρόπο σαρώνουμε και τις (176+4)/4=45 θέσεις όπου βρίσκονται αποθηκευμένα τα pixel της υπό επεξεργασία εικόνας. Τα 4 επί πλέον εικονοστοιχεία οφείλονται στο padding της εικόνας ώστε η έξοδος να διατηρήσει τις διαστάσεις της εισόδου. Τελικά τα δεδομένα εξόδου μεταφέρονται μέσω της σύνδεσης FSL και πάλι στον κύριο επεξεργαστή και το σύστημα περιμένει δεδομένα για να αντικαταστήσει τα δεδομένα της παλαιότερης block ram. Όταν θα έχουμε επεξεργαστεί και τις 144 γραμμές της εικόνας εισόδου το σύστημα θα επιστρέψει στην αρχική κατάσταση με όλες τις μνήμες κενές προς εγγραφή περιμένοντας τα δεδομένα από μια νέα προς επεξεργασία εικόνα.

Σύνθεση του συστήματος – περιγραφή λειτουργίας.

Με τα επιμέρους τμήματα του συστήματος που περιγράψαμε μπορούμε πλέον να συνθέσουμε τον ολοκληρωμένο συνεπεξεργαστή. Στην εικόνα 42 περιλαμβάνονται όλες οι τμηματικές μονάδες που συνθέτουν το σύστημά μας.

Τα δεδομένα από τον microblaze αποθηκεύονται στην ουρά του FSL. Η μηχανή καταστάσεων ενημερώνεται από την ουρά για την ύπαρξη νέων δεδομένων και το σύστημα μπαίνει στην κατάσταση φόρτωσης των δεδομένων στις block ram. Οι διευθύνσεις για τις μνήμες και τα σήματα ενεργοποίησης εγγραφής ελέγχονται από την μηχανή καταστάσεων. Η μηχανή καταστάσεων επιλέγει σε ποια μνήμη θα εγγράψει τα νέα δεδομένα με βάση την παλαιότητα των δεδομένων σε αυτές. Τα παλαιότερα δεδομένα θα αντικατασταθούν, καθώς αυτά δεν θα χρειαστούν ξανά σε κάποιον υπολογισμό. Η επιλογή κάποιας μνήμης γίνεται με χρήση του αντίστοιχου σήματος επίτρεψη εγγραφής.



Εικόνα 42. Μονοπάτι δεδομένων για το BloG φίλτρο.

Οι έξοδοι των block ram είναι αυτές που θα τροφοδοτήσουν με δεδομένα την αριθμητική μονάδα του μονοπατιού δεδομένων. Καθώς όμως αντικαθιστούμε κάθε φορά την παλαιότερη γραμμή τα δεδομένα που περιέχονται στις μνήμες δεν είναι ταξινομημένα με βάση την γραμμή τους στην αρχική εικόνα. Δηλαδή η $10^{η}$ γραμμή μπορεί να βρίσκεται στην μνήμη 1 και η $6^{η}$ στην μνήμη 5. Για να ξεπεράσουμε αυτό το πρόβλημα η μηχανή καταστάσεων χρησιμοποιεί έναν μετρητή για να ξέρουμε ποια μνήμη περιέχει δεδομένα από την μικρότερη (σε δείκτη πίνακα) γραμμή. Με ένα δίκτυο από πολυπλέκτες θα ταξινομήσουμε τις εξόδους των μνημών στις εισόδους της αριθμητικής μονάδας. Το σήμα ελέγχου των πολυπλεκτών είναι ο μετρητής που χρησιμοποιεί η μηχανή καταστάσεων.

Το επόμενο στάδιο είναι ο υπολογισμός των δεδομένων εξόδου. Η μηχανή καταστάσεων επιλέγει κατάλληλες διευθύνσεις για τις πόρτες των μνημών και στον ίδιο παλμό υπολογίζονται τα δεδομένα εξόδου από την αριθμητική μονάδα. Άμεσα γίνεται η απόφαση για την ύπαρξη ακμής και η δυαδικές έξοδοι ομαδοποιούνται σε λέξεις των 32 bit για να οδηγηθούν στην ουρά εξόδου προς τον Microblaze.

Η σύνθεση του συστήματος στο περιβάλλον Xilinx EDK φαίνεται στην εικόνα 44.

Μπορούμε να διακρίνουμε τον κύριο δίαυλο του συστήματος όπου βρίσκονται συνδεδεμένα ο ελεγκτής της σειριακής θύρας, τα ενδεικτικά led, ο χρονομέτρης και η μονάδα εκσφαλμάτωσης. Επίσης ο συνεπεξεργαστής που δημιουργήσαμε συνδέεται με τον δίαυλο τύπου Fast Simplex Link με τον κύριο επεξεργαστή με δύο ανεξάρτητα μονόδρομα κανάλια, ένα για αποστολή δεδομένων και ένα για λήψη δεδομένων.

Για τη σύνθεση του συνεπεξεργαστή που σχεδιάσαμε το Xilinx EDK μας παρέχει αναφορά για το hardware που καταναλώσαμε από το FPGA.

General			
IP Core	edgeDetectionHard		
Version	1.00.a		
Post Synthesis Device Utilization			
Resource Type	Used	Available	Percent
Slices	1131	4656	24
Slice Flip Flops	451	9312	4
4 input LUTs	2007	9312	21
BRAMs	5	20	25
GCLKs	1	24	4

Όπως είδαμε χρησιμοποιούμε 5 από τις 20 block ram του FPGA για την προσωρινή αποθήκευση των pixel της υπό επεξεργασίας εικόνας, καθώς και 1131 slices και 2007 look up tables. Τέλος χρειάστηκαν μόλις 451 flip-flop. Ειδικά ο αριθμός των flip flop είναι πολύ μικρός κυρίως λόγω της χρήσης των block rams, που μας απαλλάσσουν από την χρήση περισσοτέρων flip flop.

Χρονισμός του συστήματος

Το Xilinx EDK μας παρέχει επίσης τα δεδομένα χρονισμού του συστήματος μας. Δηλαδή την μέγιστη καθυστέρηση που παρουσιάζεται στο κύκλωμά μας. Από την ανάλυση αυτή παίρνουμε

```
Delay:              22.133ns (data path - clock path skew + uncertainty)
  Source:              EdgeDetect/Mram_ram_line_41.A (RAM)
  Destination:         EdgeDetect/d1/pixel_0 (FF)
  Data Path Delay:     22.133ns (Levels of Logic = 18)
  Clock Path Skew:     0.000ns
  Source Clock:        FSL_Clk_BUFGP rising
  Destination Clock:   FSL_Clk_BUFGP rising
  Clock Uncertainty:   0.000ns

  Data Path: EdgeDetect/Mram_ram_line_41.A to EdgeDetect/d1/pixel_0
    Delay type         Delay(ns)  Logical Resource(s)
    ----------------------------------------------------------------------------------------------
    Tbcko              2.812   EdgeDetect/Mram_ram_line_41.A    net (fanout=3)    e  0.100
                ………………………………………………………………………………………
                ………………………………………………………………………………………
net (fanout=1)     e  0.100   EdgeDetect/data_out_temp<35>    Tfck
    ----------------------------------------------------------------------------------------------
    Total              22.133ns (20.333ns logic, 1.800ns route)
                         (91.9% logic, 8.1% route)
```

Η τιμή αυτή για την μέγιστη καθυστέρηση που παίρνουμε για το μονοπάτι της εικόνας 37 δεν είναι ικανοποιητική καθώς είναι μεγαλύτερη από την περίοδο του ρολογιού του microblaze. Για να το μειώσουμε ακόμα περισσότερο προβαίνουμε σε μια μετατροπή του κυκλώματος. Αντί πρώτα να προσθέσουμε όλα τα εικονοστοιχεία που θα πολλαπλασιαστούν με τον ίδιο συντελεστή, θα εκτελέσουμε τις απαραίτητες ολισθήσεις και μετά θα προσθέσουμε για να λάβουμε το τελικό αποτέλεσμα. Με τον τρόπο αυτό μειώνεται το αρχικό μήκος λέξης των δεδομένων, δημιουργούνται πολλά εσφαλμένα μονοπάτια κρατουμένου (false paths) τα οποία θα συμβάλουν στην μείωση της καθυστέρησης. Μειονεκτήματα αυτής της υλοποίησης είναι η απώλεια ακρίβειας λόγω της αποκοπής που γίνεται κατά την ολίσθηση πριν την πρόσθεση όλων των εικονοστοιχείων. Ευτυχώς η απώλεια ακρίβειας είναι ανεκτή για την εφαρμογή μας. Με την μετατροπή αυτή του μονοπατιού δεδομένων πετυχαίνουμε τα παρακάτω αποτελέσματα.

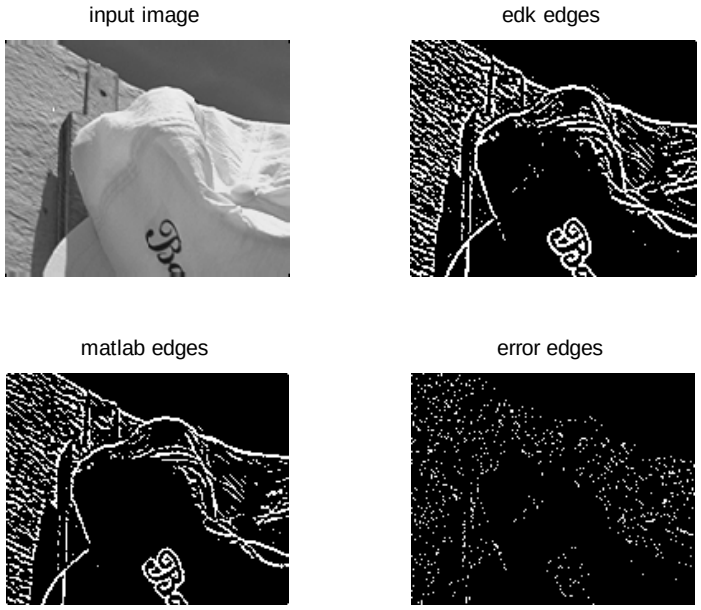
```
Delay:            10.059ns (Levels of Logic = 11)
  Source:            edgedetectionhard_0/EdgeDetect/d1/counter_23 (FF)
  Destination:       edgedetectionhard_0/EdgeDetect/d1/counter_0 (FF)
  Source Clock:      FSL_Clk rising
  Destination Clock: FSL_Clk rising

  Data Path: edgedetectionhard_0/EdgeDetect/d1/counter_23 to edgedetectionhard_0/EdgeDetect/d1/counter_0
                     Gate     Net
    Cell:in->out     fanout   Delay   Delay  Logical Name (Net Name)
    ------------------------------------------------------------------------------------------------------------------
     FDRE:C->Q            2 0.591 1.052 edgedetectionhard_0/EdgeDetect/d1/counter_23
                          (edgedetectionhard_0/EdgeDetect/d1/counter_23)
     LUT4_L:I0->LO        1   0.704   0.000  edgedetectionhard_0/EdgeDetect/d1/Ker1_wg_sel (N23)
     LUT4_D:I2->LO        2   0.704   0.179  edgedetectionhard_0/EdgeDetect/d1/Ker2 (N7898)
                ………………………………………………………………………………………………
                ………………………………………………………………………………………………
```

```
LUT2:I1->O            32   0.704   1.853  edgedetectionhard_0/EdgeDetect/d1/_n00211
                           (edgedetectionhard_0/EdgeDetect/d1/_n0021)
FDRE:R                0.711        edgedetectionhard_0/EdgeDetect/d1/counter_0
----------------------------------------------------------------------------------------------------
Total                 10.059ns (5.152ns logic, 4.907ns route)
                           (51.2% logic, 48.8% route)
```

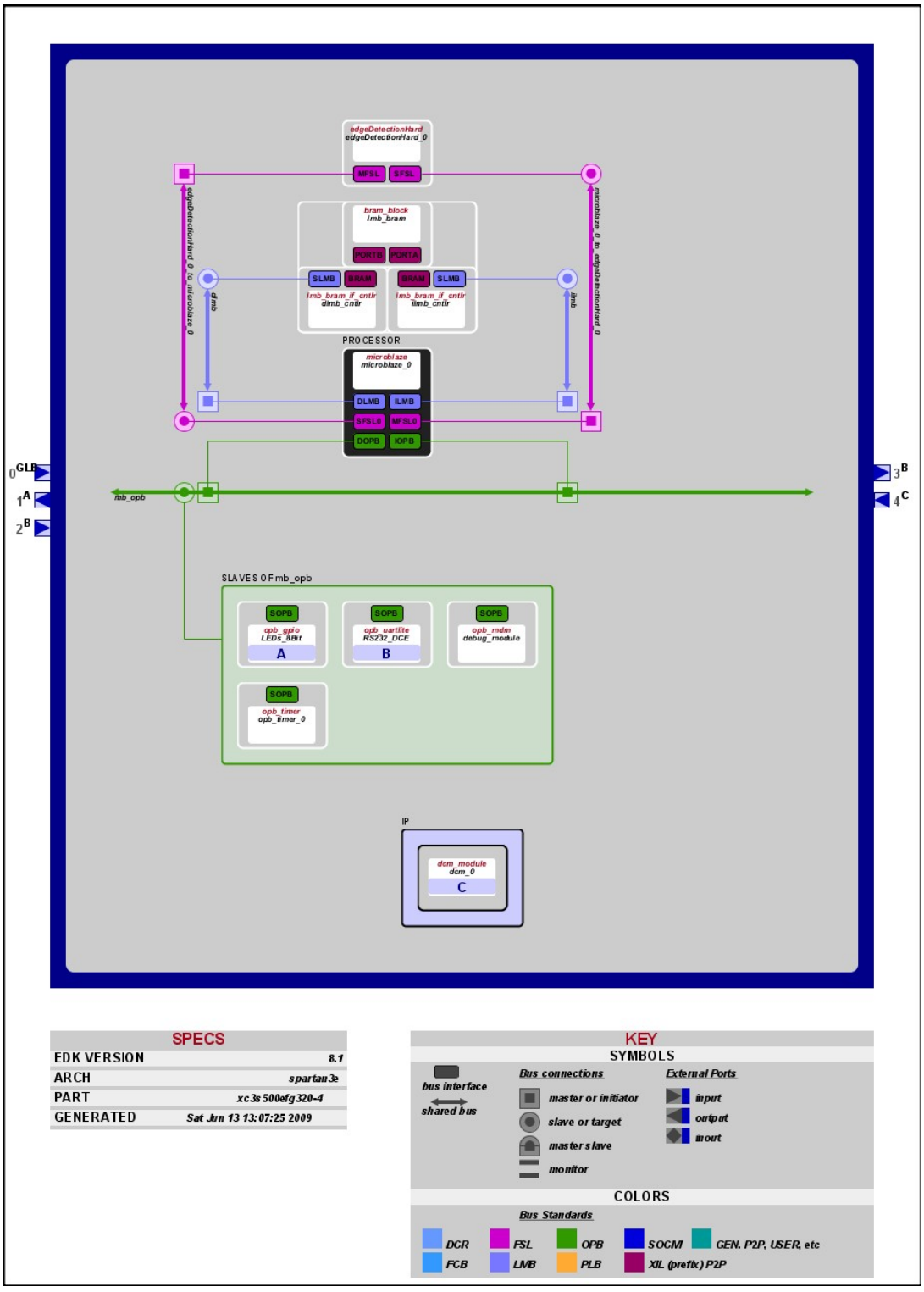
Η μέγιστη τιμή συχνότητας λειτουργίας που μας δίνει για την μονάδα ανίχνευσης ακμών είναι ***99.412MHz***, τιμή μεγαλύτερη ακόμα και από την μέγιστη συχνότητα λειτουργίας του Microblaze. Αυτή η συχνότητα λειτουργίας προκύπτει από την μέγιστη καθυστέρηση που εισαγάγει το μονοπάτι δεδομένων. Αν είχαμε την δυνατότητα να χρησιμοποιήσουμε την μονάδα που σχεδιάσαμε με κάποιον ταχύτερο επεξεργαστή θα μπορούσαμε να επιτύχουμε ακόμα μικρότερους χρόνους εκτέλεσης, κάτι που θα μας δώσει την δυνατότητα για επεξεργασία καρέ μεγάλου μεγέθους και υψηλής συχνότητας σε πραγματικό χρόνο, η ακόμα και την επεξεργασία εικόνων από πολλαπλές πηγές.

Εκτέλεση του αλγόριθμου

Εκτελέσαμε τον αλγόριθμο με χρήση του συνεπεξεργαστή και έπειτα συγκρίναμε τα αποτελέσματα με αυτά από το περιβάλλον matlab. Παρατηρούμε ότι σε κάποια οριακά εικονοστοιχεία ανιχνεύονται λανθασμένα ακμές, αλλά αυτό έχει να κάνει με το σφάλμα αποκοπής λόγω της ολίσθησης. Ο υπολογισμός της εξόδου ολοκληρώνεται σε 39.316 παλμούς!



Εικόνα 43. Αποτελέσματα της εκτέλεσης του αλγόριθμου σε hardware και σύγκριση με το matlab.



Εικόνα 44. Ολοκληρωμένο σύστημα στο EDK με τον microblaze και τον συνεπεξεργαστή.

Υλοποίηση σε hardware του φίλτρου LoG

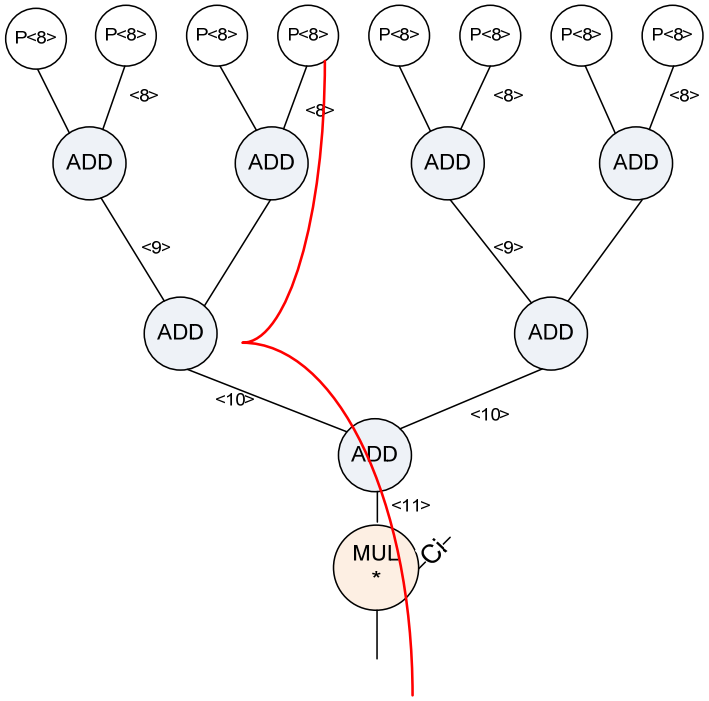
Σε αυτό το κεφάλαιο θα εξετάσουμε τι συμβαίνει στην περίπτωση που θα επιλέξουμε το φίλτρο laplacian of gaussian για την ανίχνευση ακμών και εξετάζουμε το κόστος που θα προσθέσει αυτή σε υλικό και καθυστέρηση.

Το μονοπάτι δεδομένων (datapath)

Στην περίπτωση του laplacian of gaussian φίλτρου όπως έχουμε ξαναδεί για την περίπτωση μιας μήτρας 5x5 όπως η ακόλουθη,

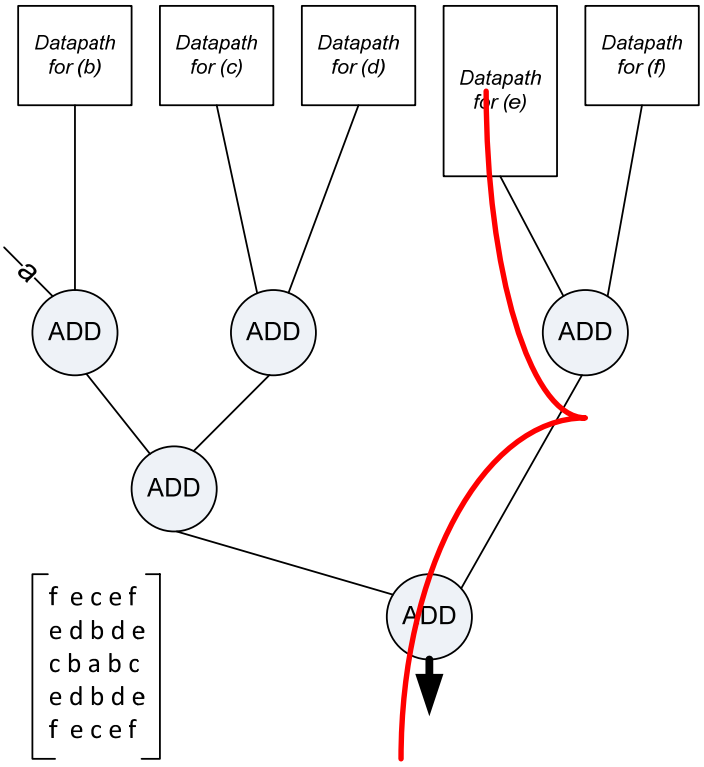
0.0059	*0.0417*	*0.0744*	*0.0417*	*0.0059*
0.0417	*0.1323*	*-0.0460*	*0.1323*	*0.0417*
0.0744	*-0.0460*	*-1.0005*	*-0.0460*	*0.0744*
0.0417	*0.1323*	*-0.0460*	*0.1323*	*0.0417*
0.0059	*0.0417*	*0.0744*	*0.0417*	*0.0059*

παρατηρούμε πως έχουμε 25 μη μηδενικούς συντελεστές, που λαμβάνουν 6 διαφορετικές τιμές. Ένας από τους συντελεστές έχει πολλαπλότητα εμφάνισης 1, ένας έχει πολλαπλότητα 8, και οι υπόλοιποι 4. Συνολικά δηλαδή μας αρκούν 6 πολλαπλασιαστές και 24 αθροιστές. Με αυτή την επιλογή το κρίσιμο μονοπάτι με την μεγαλύτερη καθυστέρηση εισάγεται από τον συντελεστή 0.0417 που έχει πολλαπλότητα εμφάνισης 8 και θα εισάγει ένα επιπλέον επίπεδο αθροιστών.



Εικόνα 45. Πρώτο μέρος του μονοπατιού δεδομένων για το LoG φίλτρο

Φυσικά η προσθήκη ενός επιπλέον επιπέδου αθροιστών είναι αμελητέα σε σχέση με την καθυστέρηση που θα εισάγει ο πολλαπλασιαστής, καθώς ο συντελεστής με τον οποίο πολλαπλασιάζουμε πρέπει να έχει μεγάλο μήκος λέξης ώστε να ικανοποιεί τα κριτήρια ακρίβειας του αλγόριθμου. Επίσης παρατηρούμε ότι ο κεντρικός συντελεστής της μήτρας είναι μονάδα κι έτσι δεν θα χρειαστούμε πολλαπλασιαστή γι αυτόν. Συνολικά το μονοπάτι δεδομένων δίνεται στην εικόνα που ακολουθεί.



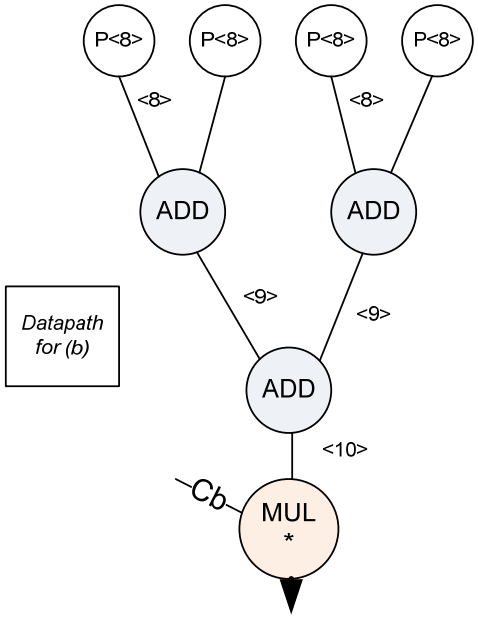
Εικόνα 46. Ολοκληρωμένο μονοπάτι δεδομένων για το LoG φίλτρο

Όπου τα επί μέρους τμήματα του μονοπατιού για τις ομάδες εικονοστοιχείων b,c,d,f εικονίζονται στην εικόνα 47.

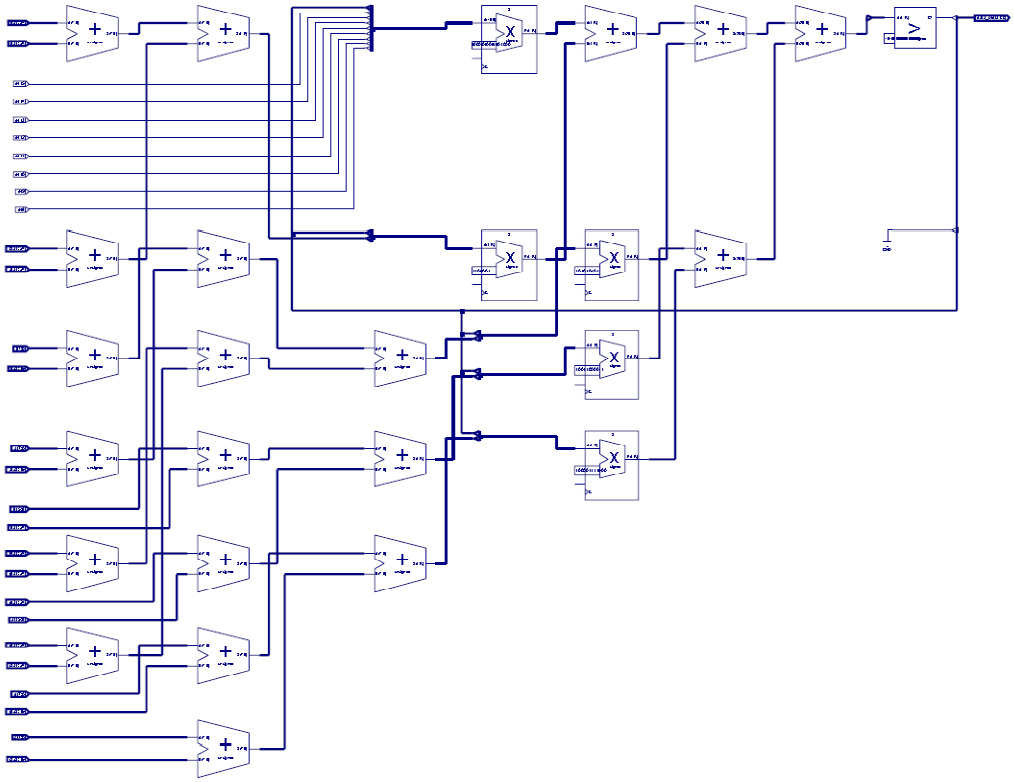
Βλέπουμε ότι συνολικά θα χρειαστούν 5 πολλαπλασιαστές και 24 πολλαπλασιαστές, και το κρίσιμο μονοπάτι περιλαμβάνει 5 επίπεδα αθροιστών και έναν πολλαπλασιαστή.

Υλοποίηση του φίλτρου σε vhdl

Υλοποιήσαμε το φίλτρο αυτό σε vhdl κώδικα για να εξετάσουμε εκτενέστερα τα χαρακτηριστικά του μέσα στο περιβάλλον της Xilinx. Το μονοπάτι υλοποιήθηκε στο Xilinx ISE 8.1 και έδωσε το παρακάτω αποτέλεσμα μετά την διαδικασία κατασκευής του RTL σχηματικού όπως φαίνονται στο σχήμα 48.



Εικόνα 47. Δεύτερο μέρος του μονοπατιού δεδομένων για το LoG φίλτρο



Εικόνα 48. RTL σχηματικό διάγραμμα του μονοπατιού δεδομένων για το LoG φίλτρο.

Παρατηρούμε ότι οι αλγόριθμοι του Xilinx ISE υλοποίησαν ακόμα πιο αποδοτικά απ' ότι σχεδιάζαμε το μονοπάτι δεδομένων χρησιμοποιώντας τέσσερις λιγότερους αθροιστές. Στο τέλος του μονοπατιού διακρίνουμε και έναν συγκριτή. Αυτός χρησιμοποιείται για την απόφαση ύπαρξης ακμής ή όχι συγκρίνοντας με το κατώφλι που έχουμε ορίσει το αποτέλεσμα της συνελικτικής πράξης.

Χρονισμός του συστήματος

Το xilinx ISE μας παρέχει επίσης πληροφορίες για την μέγιστη καθυστέρηση που εισάγουμε με την σχεδίαση αυτή. Δυστυχώς η καθυστέρηση που υπολογίζεται είναι τεράστια σε σύγκριση με την προηγούμενη υλοποίηση που παρουσιάσαμε κάτι που δικαιώνει του πρότερους ισχυρισμούς μας για τα πλεονεκτήματα της προσέγγισης του laplacian of gaussian με το BLOG φίλτρο.

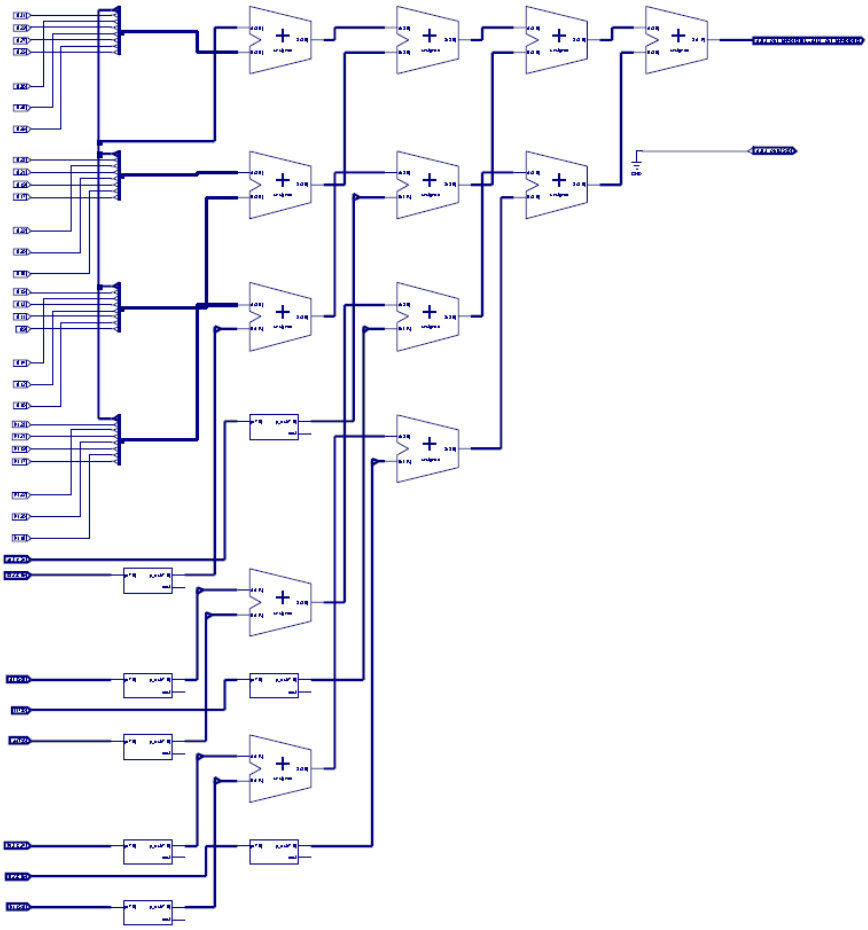
```
Offset:          28.027ns (Levels of Logic = 39)
 Source:         edgedetectionhard_0/EdgeDetect/p1/e_out_0 (LATCH)
 Destination:    FSL_M_Data<24> (PAD)
 Source Clock:   edgedetectionhard_0/EdgeDetect/p1/_n00231 falling

 Data Path: edgedetectionhard_0/EdgeDetect/p1/e_out_0 to FSL_M_Data<24>
                    Gate    Net
  Cell:in->out      fanout  Delay   Delay  Logical Name (Net Name)
 ----------------------------------------------------------------------------------------
  LD:G->Q    12   0.676   1.197  edgedetectionhard_0/EdgeDetect/p1/e_out_0
                                  ........

  XORCY:CI->O         0   0.780   0.000  edgedetectionhard_0/EdgeDetect/...
 ----------------------------------------------------------------------------------------
 Total                28.027ns (18.916ns logic, 9.111ns route)
                      (67.5% logic, 32.5% route)
```

Από την αναφορά που λαμβάνουμε από το Xilinx EDK βλέπουμε πως η ακριβή τιμή της μέγιστης καθυστέρησης είναι 28.027ns. Η τιμή αυτή υπολείπεται κατά πολύ της τιμής που υπολογίστηκε παραπάνω για το προσεγγιστικό bilevel φίλτρο και μάλιστα ξεπερνά την περίοδο λειτουργίας του μικροεπεξεργαστή Microblaze.

Ανάλυση καθυστέρησης για το φίλτρο BloG

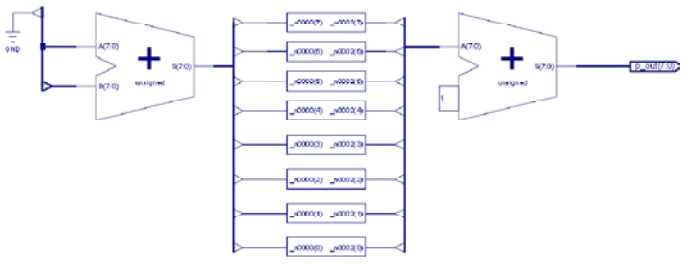
Στο προηγούμενο κεφάλαιο είδαμε ότι το μονοπάτι δεδομένων για το φίλτρο BloG στην περίπτωση της μέγιστης καθυστέρησης περιλαμβάνει 5 επίπεδα αθροιστών, τον υπολογισμό του συμπληρώματος ως προς δυο που υλοποιείται από έναν αντιστροφέα και έναν αθροιστή προσθέτοντας 1 από το εργαλείο σύνθεσης, και ολίσθηση η οποία όμως δεν εισάγει καθυστέρηση λογικού επιπέδου. Επίσης ενας ακόμα αθροιστής χρησιμοποιείται μετά τις ολισθήσεις για να υλοποιήσουμε τον πολλαπλασιασμό με τον συντελεστή F2 που αναπαριστάται ως άθροισμα δυνάμεων του δύο. Οι αθροιστές που χρησιμοποιήθηκαν έχουν μέγεθος εννέα ψηφίων κάτι που είναι αρκετό για την εφαρμογή, και η έξοδος διατηρεί το ίδιο μέγεθος. Στο σχήμα που ακολουθεί φαίνεται το RTL σχηματικό που υλοποιήθηκε στο Xilinx

ISE. Μπορούμε άμεσα να διακρίνουμε τους 4 από τους συνολικά 6 αθροιστές που απαρτίζουν το μονοπάτι μέγιστης καθυστέρησης.



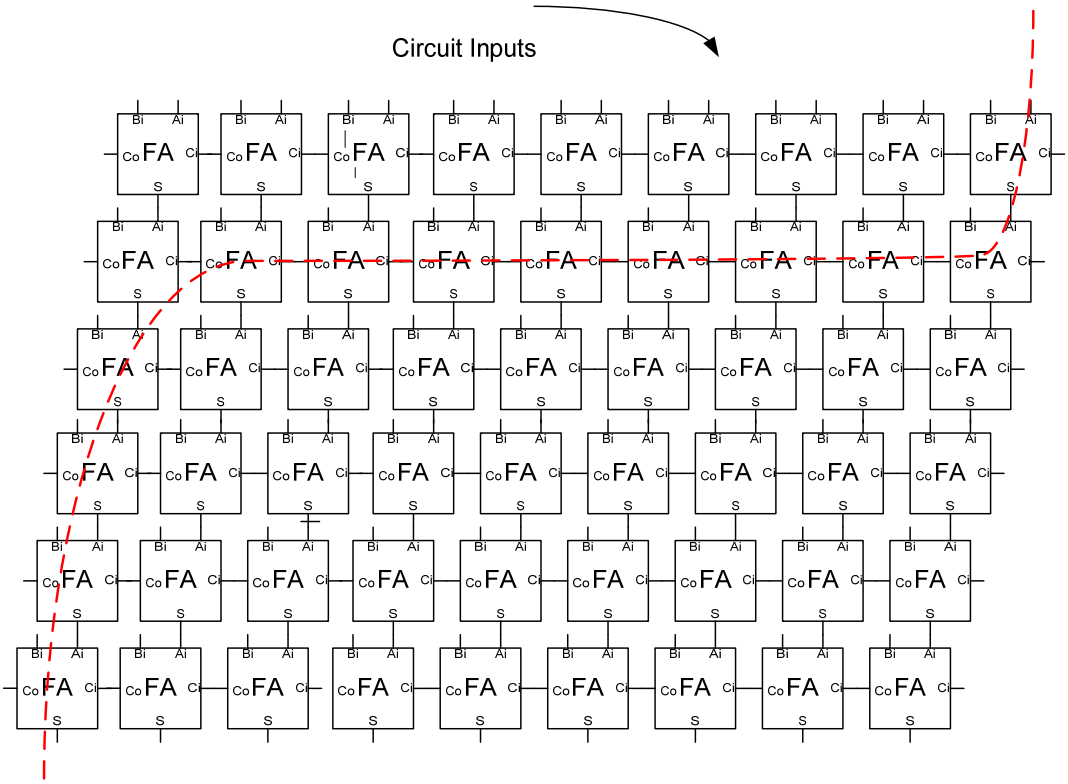
Εικόνα 49. RTL σχηματικό για το μονοπάτι δεδομένων του BLOG φίλτρου

Δύο ακόμα βρίσκονται μέσα στο επιμέρους τμήμα του κυκλώματος που υπολογίζει το γινόμενο των εικονοστοιχείων με τον συντελεστή F2=0.3125 του φίλτρου με δύο ολισθήσεις και μια πρόσθεση καθώς και την συμπλήρωσή του αποτελέσματος ως προς 2 με αντιστροφή των bit και πρόσθεση της σταθεράς ‘1’.



Εικόνα 50. RTL σχηματικό για τον υπολογισμό γινομένου και του συντελεστή F2.

Ουσιαστικά το μονοπάτι μέγιστης καθυστέρησης αποτελείται όπως βλέπουμε από τους 6 αθροιστές και έναν αντιστροφέα. Θα εξετάσουμε σε επίπεδο πυλών πως ερμηνεύεται αυτή η καθυστέρηση. Θα θεωρήσουμε αθροιστές διάδοσης κρατούμενου 9 bit, καθώς αυτό είναι το μέγεθος του μονοπατιού που χρησιμοποιούμε για να αποφύγουμε υπερχείλιση κατά την εκτέλεση των πράξεων. Επίσης θα θεωρήσουμε σαν δομική μονάδα των αθροιστών έναν πλήρη αθροιστή. Για τους 6 αθροιστές του μονοπατιού μας λοιπόν μπορούμε να παρατηρήσουμε το μονοπάτι της μέγιστης καθυστέρησης στο ακόλουθο σχήμα.



Εικόνα 51. Λεπτομερές μονοπάτι δεδομένων για το BloG φίλτρο.

Βλέπουμε ότι η καθυστέρηση συγκεντρώνεται σε 14 επίπεδα πλήρων αθροιστών, και στην χειρότερη περίπτωση θα έχουμε 6 καθυστερήσεις αθροίσματος (s = a xor b xor c_{in}) και 8 καθυστερήσεις διάδοσης κρατούμενου (C_{out} = [{a xor b} and C_{in}] or [a and b]) . Εξετάζουμε μόνο την μία είσοδο του αθροιστή για απλότητα καθώς το κύκλωμα είναι συμμετρικό και για την άλλη είσοδο. Τελικά πρέπει να προσθέσουμε στην καθυστέρηση ένος αντιστροφέα που χρησιμοποιείται όπως είπαμε για την συμπλήρωση ως προς 2 των εικονοστοιχείων που αφαιρούνται.

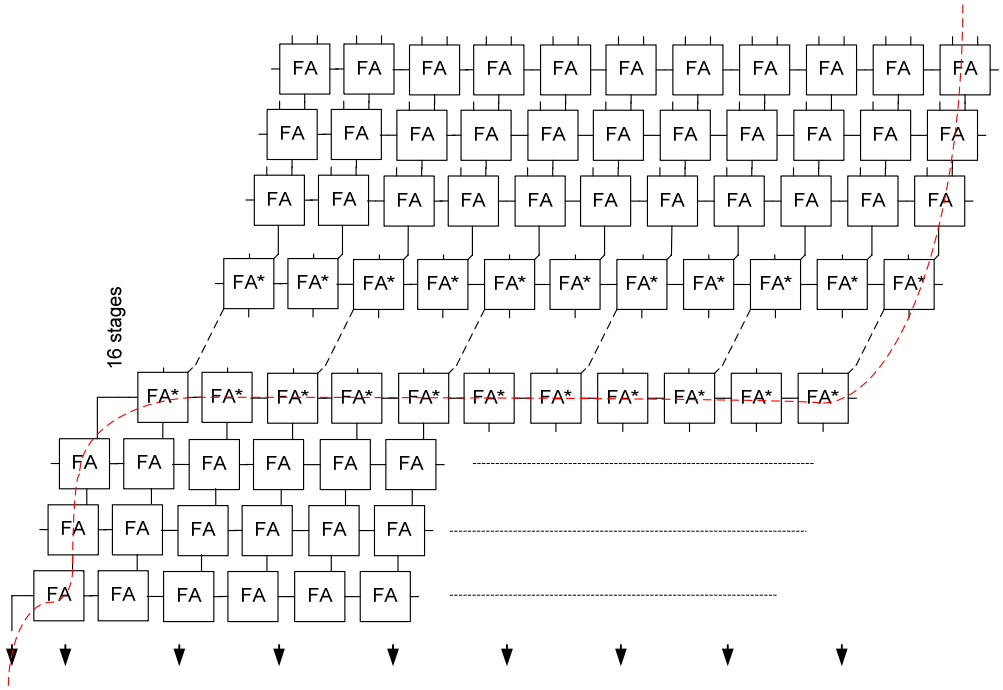
Ένα σημαντικό σημείο αυτού του μονοπατιού δεδομένων είναι πως στον πρώτο αθροιστή τα δεδομένα έχουν υποστεί ολίσθηση κατά 2 και κατά 4 ψηφία. Αυτό θα δημιουργήσει εσφαλμένα μονοπάτια για τα κρατούμενα των πιο σημαντικών ψηφίων και μειώνει την μέγιστη καθυστέρηση του κυκλώματος. Αυτό συνεχίζεται και σε επόμενα επίπεδα αθροιστών

αλλά για μικρότερο αριθμό ψηφίων κάθε φορά καθώς αυξάνεται το άθροισμα προχωρώντας βαθύτερα στο μονοπάτι.

Αναλύοντας το συνολικό μονοπάτι με το Xilinx ISE και το Timing Analyzer υπολογίζουμε ότι η μέγιστη καθυστέρηση που παρατηρείται για το μονοπάτι δεδομένων είναι *10.059ns*.

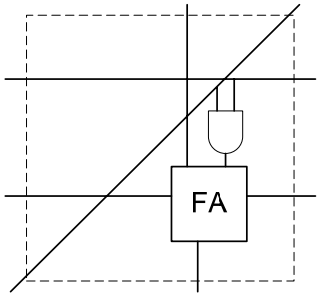
Ανάλυση καθυστέρησης για το φίλτρο LoG

Οι συντελεστές του φίλτρου laplacian of gaussian είναι δεκαδικοί και συνήθως πολύ μικροί αριθμοί. Είμαστε αναγκασμένοι λοιπόν να χρησιμοποιήσουμε αριθμητική σταθερής υποδιαστολής με πολύ μεγάλο μήκος λέξης για την αναπαράσταση των συντελεστών ώστε να επιτύχουμε ικανοποιητική ακρίβεια με την εκτέλεση του αλγόριθμού μας. Στην περίπτωσή μας θα χρησιμοποιήσουμε λέξη 16 ψηφίων, όπου το πρώτο χρησιμοποιείται για πρόσημο και τα υπόλοιπα 15 αποτελούν το δεκαδικό μέρος του συντελεστή. Στο μονοπάτι δεδομένων που εξετάσαμε και παραπάνω για το laplacian of gaussian φίλτρο είδαμε ότι η μέγιστη καθυστέρηση αποτελείται από 5 επίπεδα αθροιστών και έναν πολλαπλασιαστή. Τα τρία πρώτα επίπεδα αθροιστών έχουν μήκος λέξης 11 ψηφίων, ο πολλαπλασιαστής είναι 11x16 και οι υπόλοιποι αθροιστές έχουν μήκος 27 ψηφίων, καθώς δεν κινδυνεύουμε από υπερχείλιση από την φύση του αλγόριθμου, προσθαφαιρώντας θετικούς και αρνητικούς αριθμούς. Φυσικά η έξοδος δεν χρειάζεται να έχει αυτό το μήκος και θα προχωρήσουμε σε αποκοπή κρατώντας 9 ψηφία. Αυτό εισάγει φυσικά σφάλμα αποκοπής στην έξοδο αλλά το σφάλμα είναι αποδεκτό για τον αλγόριθμό μας και μάλιστα πολύ μικρό αφού χάνουμε μόνο το δεκαδικό μέρος της εξόδου. Στην ακόλουθη εικόνα μπορούμε να δούμε αναλυτικά το μονοπάτι που εισάγει την μέγιστη καθυστέρηση.



Εικόνα 52. Λεπτομερές μονοπάτι δεδομένων για το LoG φίλτρο.

Βλέπουμε ότι το μονοπάτι περιλαμβάνει 6 επίπεδα πλήρους αθροιστή και 27 επίπεδα του στοιχειώδους δομικού στοιχείου του πολλαπλασιαστή, FA* που είναι ένας πλήρης αθροιστής με την προσθήκη μιας πύλης AND. Έντεκα από τα επίπεδα αυτά είναι επίπεδα διάδοσης κρατούμενου, κάτι που μπορεί να αποφευχθεί με χρήση carry save μορφής για τον πολλαπλασιαστή.



Εικόνα 53. Στοιχειώδης δομική μονάδα πολλαπλασιαστή

Βλέπουμε ότι με την εισαγωγή και μόνο ενός πολλαπλασιαστή αυξάνει δραματικά το μέγεθος του μονοπατιού και φυσικά την μέγιστη καθυστέρηση που εισάγει. Στο εικονιζόμενο μονοπάτι πρέπει να προσθέσουμε και ένα επίπεδο συγκριτή προκειμένου να αποφασίσουμε για την ύπαρξη ακμών.

Από την ανάλυση που έγινε με το Xilinx ISE και το Xilinx Timing analyzer η μέγιστη καθυστέρηση μετά την τοποθέτηση του κυκλώματος στο FPGA υπολογίστηκε σε *28.166ns*. Η αύξηση της τιμής αυτής οφείλεται τόσο στην αύξηση της λογικής αλλά και στο κόστος της δρομολόγησης των σημάτων στο FPGA.

Δέσμευση πόρων του FPGA

Άλλο ένα μεγάλο μειονέκτημα του συγκεκριμένου μονοπατιού δεδομένων είναι η μεγάλη επιφάνεια που καταλαμβάνει το κύκλωμα λόγω των μεγάλων λέξεων που αναγκαζόμαστε να χρησιμοποιήσουμε για τους συντελεστές του φίλτρου προκειμένου να επιτύχουμε ικανοποιητική ακρίβεια. Από την αναφορά που λαμβάνουμε από το Xilinx EDK για την κάλυψη των πόρων του FPGA επιβεβαιώνεται ακριβώς αυτός ο ισχυρισμός.

General			
IP Core	edgeDetectionHard		
Version	1.00.a		
Driver	API		
Post Synthesis Device Utilization			
Resource Type	Used	Available	Percent
Slices	2106	4656	45
Slice Flip Flops	518	9312	5
4 input LUTs	3916	9312	42
BRAMs	5	20	25
GCLKs	1	24	4

Σύγκριση δέσμευσης πόρων συστήματος από τις δύο υλοποιήσεις

Στον παρακάτω πίνακα θα συγκρίνουμε τις δύο υλοποιήσεις αναφορικά με τους πόρους που δεσμεύουν στο FPGA. Θα συγκρίνουμε τους δύο συνεπεξεργαστές που υλοποιήσαμε και όχι το ολικό σύστημα. Επίσης να διευκρινίσουμε ότι σε κάθε περιφερειακό έχουν υλοποιηθεί τέσσερα μονοπάτια δεδομένων για ταχύτερους υπολογισμούς όπως έχουμε προαναφέρει.

Resource Type	BloG	coverage %	LoG	coverage %	Available	Reduction
Slices	1131	24%	2106	45%	4656	**46.3%**
Slice Flip Flops	451	5%	518	6%	9312	**12.9%**
4 input LUTs	2007	22%	3916	42%	9312	**48.7%**
BRAMs	5	25%	5	25%	20	**0.0%**
GCLKs	1	4%	1	4%	24	**0.0%**

Διαπιστώνουμε ότι η οικονομία σε slices και look up tables είναι εξαιρετικά σημαντική στα όρια του 50%. Αυτό θα οδηγήσει και σε οικονομία ενέργειας καθιστώντας το κύκλωμα κατάλληλο και για εφαρμογές χαμηλών ενεργειακών εφαρμογών, καθώς η κατανάλωση ενέργειας είναι ανάλογη με την επιφάνεια που καλύπτει το κύκλωμα.

Αξιολόγηση του επιταχυντή

Αξιολόγηση επίδοσης

Σκοπός για την υλοποίηση του φίλτρου ανίχνευσης ακμών σε γλώσσα περιγραφής υλικού είναι η επιτάχυνση του αλγόριθμου σε σχέση με την υλοποίηση που έχει γίνει σε software. Όπως είδαμε και παραπάνω έχουν γίνει μετρήσεις και για την επίδοση του αλγόριθμου όταν είναι υλοποιημένος σε software. Τις ίδιες μετρήσεις πραγματοποιήσαμε και στην περίπτωση που περιλαμβάνουμε στο σύστημά μας τον συνεπεξεργαστή για το φίλτρο ανίχνευσης ακμών και συγκρίνουμε τα αποτελέσματα στον παρακάτω πίνακα.

Implementation	Clk cycles	reduction
software LoG	128668	
software BLoG	128668	0
Hardware BLoG	39316	69%

Βλέπουμε ότι πετυχαίνουμε μια μείωση των απαιτούμενων παλμών ρολογιού για την εκτέλεση του αλγόριθμου που αγγίζει το 70%. Αυτό ισοδυναμεί με ***speedup rate ίσο με 3.27***, δηλαδή ο αλγόριθμος που περιλαμβάνει τον συνεπεξεργαστή, αφοσιωμένο στον υπολογισμό της εξόδου του φίλτρου ανίχνευσης ακμών, είναι ταχύτερος περίπου 3,3 φορές σε σχέση με την συμβατική υλοποίηση σε software. Επίσης αξίζει να σημειώσουμε ότι ακόμα και στην υλοποίηση σε software η πλακέτα Xilinx Spartan 3E που χρησιμοποιούμε χρησιμοποιεί τους ενσωματωμένους πολλαπλασιαστές που διαθέτει καθώς και εξειδικευμένη μονάδα κινητής υποδιαστολής που διαθέτει. Με αυτό τον τρόπο επιταχύνει το software καθώς πρόκειται για πράξεις που κοστίζουν ιδιαίτερα υπολογιστικά όταν υλοποιούνται εξ' ολοκλήρου σε λογισμικό. Αυτό σημαίνει ότι αν η υλοποίηση γινόταν καθαρά σε λογισμικό η διαφορά στην επίδοση θα ήταν ακόμα μεγαλύτερη.

Επεκτάσεις

Με τον τρόπο που είναι γραμμένος ο συνεπεξεργαστής, απλά αλλάζοντας μερικές σταθερές στον κώδικα που είναι γραμμένος σε VHDL μπορούμε να διαχειριστούμε ότι μέγεθος εικόνας θέλουμε. Φυσικά υπάρχει ο περιορισμός από πλήθος των στοιχείων που μπορούν να αποθηκευτούν σε μία block ram (βέβαια είναι ικανοποιητικά μεγάλο ακόμα και για υψηλή ανάλυση), αλλά και ο περιορισμός της συνολικής μνήμης που είναι διαθέσιμη για την αποθήκευση ενός κάδρου.

Σε περίπτωση που θα θελήσουμε την υλοποίηση ενός φίλτρου με μεγαλύτερη διάσταση τότε για κάθε επιπλέον συντελεστή στο μητρώο συνέλιξης θα προσθέτουμε και μια ακόμα block ram. Αξίζει όμως να σημειωθεί πως αυτό δεν θα επηρεάσει την επίδοση του συστήματος όσο μεγάλη μήτρα και να χρησιμοποιήσουμε! Το μόνο που θα επηρεαστεί είναι το latency μέχρι την πρώτη έξοδο καθώς αρχικά θα χρειαστεί να γεμίσουν παραπάνω block rams με στοιχεία από τις πρώτες γραμμές της εικόνας.

Υλοποίηση σε ASIC

Τα μονοπάτια δεδομένων που αναπτύχθηκαν παραπάνω για τα φίλτρα laplacian-of-gaussian και το bilevel προσεγγιστικό φίλτρο υλοποιήθηκαν και με την ASIC μεθοδολογία σε τεχνολογία 90nm στο περιβάλλον Synopsys και τα αποτελέσματα χρονισμού για τις διάφορες υλοποιήσεις, αναλογικά, είναι ταυτόσημα με την περίπτωση των FPGA. Φυσικά, όπως ήταν αναμενόμενο, η επίδοση της υλοποίησης σε ASIC είναι σαφώς ανώτερη από αυτή στο FPGA. Παρακάτω δίνονται σε πίνακα τα αποτελέσματα για την μέγιστη καθυστέρηση στις διαφορετικές υλοποιήσεις.

ASIC technology 0.90nm	
LoG	2.91ns
BloG 1	1.31ns
BloG 2	1.33ns

Η υλοποίηση BloG 1 αναφέρεται στην περίπτωση όπου πρώτα προσθέτουμε τα εικονοστοιχεία και έπειτα ολισθαίνουμε, ενώ στην περίπτωση της υλοποίησης BloG 2 πρώτα ολισθαίνουμε και έπειτα προσθέτουμε τα εικονοστοιχεία. Παρατηρούμε ότι ενώ στην περίπτωση των FPGA οι δύο αυτές υλοποιήσεις παρουσίασαν σημαντική διαφορά αναφορικά με την μέγιστη καθυστέρηση του κυκλώματος, στην περίπτωση της υλοποίησης σε ASIC εμφανίζει σχεδόν την ίδια επίδοση. Οι διαφορές αυτές έχουν να κάνουν με τις διαφορετικές τεχνολογίες και τα διαφορετικά εργαλεία σύνθεσης που χρησιμοποιούμε στις δύο περιπτώσεις. Η υλοποίηση του φίλτρου laplacian of gaussian και πάλι όμως παρουσιάζει εμφανώς μεγαλύτερη καθυστέρηση.

Το δεύτερο μεγάλο πλεονέκτημα της υλοποίησης του προσεγγιστικού bilevel φίλτρου είναι η μείωση του hardware και της κατανάλωση ισχύος. Η έξοδος του synopsys συγκεντρώνεται στους πίνακες που ακολουθούν.

	LoG	BloG 1	BloG 2
Cell internal power	1.3916 mW	183 μW	523.95 μW
Net switching power	430 μW	16.4 μW	98.73 μW
Total dynamic power	1.8221 mW	200 μW	622.69 μW
Cell leakage power	174 μW	11.59 μW	54.55 μW

	LoG	BloG 1	BloG 2
Combinational area	51859	2061	13070
Non-combinational area	4825	1864	3564
Total area	56683	3925	16634

Βλέπουμε ότι τα πλεονεκτήματα που προαναφέραμε για την υλοποίηση του bilevel προσεγγιστικού φίλτρου επιβεβαιώνονται και από την έξοδο του synopsys αναφορικά με την επιφάνεια που καταλαμβάνουν τα κυκλώματα και την κατανάλωση ισχύος που

παρουσιάζουν. Η διαφορά που παρατηρούμε είναι ότι στην περίπτωση των ASIC υλοποιήσεων το μονοπάτι δεδομένων της εικόνας 42 όπου πρώτα υλοποιούμε το δένδρο αθροιστών και έπειτα ολισθαίνουμε παρουσιάζει το μικρότερο κρίσιμο μονοπάτι αλλά και κατά πολύ μικρότερη κάλυψη επιφάνειας στην ψηφίδα καθώς και μικρότερη κατανάλωση ενέργειας. Κατανοούμε ότι είναι η πιο συμφέρουσα υλοποίηση αν επιλέξουμε την ASIC τεχνολογία.

Αναφορές – Βιβλιογραφία.

1. Ψηφιακά συστήματα VLSI, Κ.Ζ. Πεκμεστζή, ΕΜΠ 2003.
2. DSP Integrated Circuits, Lars Wanhammar, Academic Press.
3. Architectures for Digital Signal Processing, Peter Pirsch John Wiley & Sons.
4. DSP design with HDL, Lecture Notes, Johny Öberg, KTH, 2008-2009.
5. Getting Started with EDK, Xilinx documentation.
6. Microblaze reference guide, Xilinx Documentation
7. Xilinx Processor IP Library,
    a. Fast Simplex Link (FSL) Bus
    b. OPB UART (Lite)
    c. Block RAM (BRAM) Block
    d. MicroBlaze
    e. OPB Timer/Counter
8. Σχεδίαση και Υλοποίηση ενσωματωμένων συστημάτων σε προγραμματιζόμενες ψηφίδες Xilinx. Ιορδάνης Ασλανίδης ΕΜΠ 2005.
9. Fixed-Point Arithmetic: An Introduction, Randy Yates, August 23, 2007.
10. Synopsys Documentation, 2004

Παράρτημα

Εδώ θα παραθέσουμε τον κώδικα που γράφτηκε για τον συνεπεξεργαστή σε VHDL, καθώς και το πρόγραμμα ελέγχου που γράφτηκε σε C και τρέχει στον Microblaze. Τέλος δίνεται ο κώδικας σε matlab που προσομοιώνει την κάμερα μέσω της σειριακής θύρας και στέλνει τα δεδομένα της εικόνας στην πλακέτα Xilinx Spartan 3E. Τέλος τα αποτελέσματα στέλνονται πίσω στο matlab και συγκρίνουμε τα αποτελέσματα με τα θεωρητικά αναμενόμενα.

Κώδικας VHDL

Instantiation του περιφερειακού πάνω στο FSL

```
------------------------------------------------------------------------------
-- edgeDetectionHard - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                       **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          edgeDetectionHard
-- Version:           1.00.a
-- Description:       Example FSL core (VHDL).
-- Date:                    Mon May 12 15:45:28 2008 (by Create and Import
Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
```

```
--   combinatorial signals:                "*_com"
--   pipelined or register delay signals:  "*_d#"
--   counter signals:                      "*cnt*"
--   clock enable signals:                 "*_ce"
--   internal version of output port:      "*_i"
--   device pins:                          "*_pin"
--   ports:                                "- Names begin with Uppercase"
--   processes:                            "*_PROCESS"
--   component instantiations:             "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

------------------------------------------------------------------------------
-------
--
--
-- Definition of Ports
-- FSL_Clk             : Synchronous clock
-- FSL_Rst           : System reset, should always come from FSL bus
-- FSL_S_Clk       : Slave asynchronous clock
-- FSL_S_Read      : Read signal, requiring next available input to be read
-- FSL_S_Data      : Input data
-- FSL_S_CONTROL   : Control Bit, indicating the input data are control word
-- FSL_S_Exists     : Data Exist Bit, indicating data exist in the input FSL
bus
-- FSL_M_Clk       : Master asynchronous clock
-- FSL_M_Write     : Write signal, enabling writing to output FSL bus
-- FSL_M_Data      : Output data
-- FSL_M_Control   : Control Bit, indicating the output data are contol word
-- FSL_M_Full      : Full Bit, indicating output FSL bus is full
--
------------------------------------------------------------------------------
-

------------------------------------------------------------------------------
-- Entity Section
------------------------------------------------------------------------------

entity edgeDetectionHard is
       port
       (
              -- DO NOT EDIT BELOW THIS LINE ----------------------
              -- Bus protocol ports, do not add or delete.
              FSL_Clk      : in   std_logic;
              FSL_Rst      : in   std_logic;
              FSL_S_Clk    : out  std_logic;
              FSL_S_Read   : out  std_logic;
              FSL_S_Data   : in   std_logic_vector(0 to 31);
              FSL_S_Control: in   std_logic;
              FSL_S_Exists : in   std_logic;
              FSL_M_Clk    : out  std_logic;
              FSL_M_Write  : out  std_logic;
              FSL_M_Data   : out  std_logic_vector(0 to 31);
              FSL_M_Control: out  std_logic;
```

```
            FSL_M_Full    : in   std_logic
            -- DO NOT EDIT ABOVE THIS LINE ---------------------
      );

attribute SIGIS : string;
attribute SIGIS of FSL_Clk : signal is "Clk";
attribute SIGIS of FSL_S_Clk : signal is "Clk";
attribute SIGIS of FSL_M_Clk : signal is "Clk";

end edgeDetectionHard;

library edgeDetectionHard_v1_00_a;
use edgeDetectionHard_v1_00_a.all;

architecture behavioural of edgeDetectionHard is

      component Edge_detector_BRAM is
      generic(line_length: integer:=92;
            row_length: integer:=76);
      port(
            data_availiable : in STD_LOGIC;
            write_ack : in STD_LOGIC;
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            data_in : in STD_LOGIC_VECTOR(31 downto 0);
            data_out : out STD_LOGIC_VECTOR(31 downto 0);
            data_ready : out std_logic;
            read_req : out std_logic
            );
      end component Edge_detector_BRAM;


begin
-- instantiating Edge detector

EdgeDetect: Edge_detector_BRAM
      generic map(line_length=>92,
                              row_length=>76)
      port map(
            data_availiable => FSL_S_Exists,
            write_ack => FSL_M_Full,
            clk => FSL_Clk,
            reset => FSL_Rst,
            data_in => FSL_S_Data,
            data_out => FSL_M_Data,
            data_ready => FSL_M_Write,
            read_req => FSL_S_Read
            );



end architecture behavioural;
```

Μηχανή καταστάσεων για έλεγχο του συνεπεξεργαστή

```
--------------------------------------------------------------------------------
--
-- Title       : Edge_detector_BRAM
-- Design      : Edge_detector_BLoG
-- Author      : Nikolaos Anastasiadis
-- Company     : Personal Computer System
--
--------------------------------------------------------------------------------
--
-- File        : Edge_detector_BRAM.vhd
-- Generated   : Wed Apr 16 19:23:33 2008
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.20
--
--------------------------------------------------------------------------------
--
-- Description :
--
--------------------------------------------------------------------------------

--{{ Section below this comment is automatically maintained
--   and may be overwritten
--{entity {Edge_detector_BRAM} architecture {Edge_detector_BRAM}}

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Edge_detector_BRAM is
        generic(line_length: integer:=92;
                row_length: integer:=76);
        port(
                data_availiable : in STD_LOGIC;
                write_ack : in STD_LOGIC;
                clk : in STD_LOGIC;
                reset : in STD_LOGIC;
                data_in : in STD_LOGIC_VECTOR(31 downto 0);
                data_out : out STD_LOGIC_VECTOR(31 downto 0);
                data_ready : out std_logic:='0';
                read_req : out std_logic:='0'
                );
end Edge_detector_BRAM;

--}} End of automatically maintained section

architecture Edge_detector_BRAM of Edge_detector_BRAM is

component Line_Distributer is
         port(
                 a : in STD_LOGIC_VECTOR(31 downto 0);
                 b : in STD_LOGIC_VECTOR(31 downto 0);
                 c : in STD_LOGIC_VECTOR(31 downto 0);
                 d : in STD_LOGIC_VECTOR(31 downto 0);
                 e : in STD_LOGIC_VECTOR(31 downto 0);
                 f : in STD_LOGIC_VECTOR(31 downto 0);
                 g : in STD_LOGIC_VECTOR(31 downto 0);
                 h : in STD_LOGIC_VECTOR(31 downto 0);
                 i : in STD_LOGIC_VECTOR(31 downto 0);
                 j : in STD_LOGIC_VECTOR(31 downto 0);
                 row_counter : in STD_LOGIC_VECTOR(3 downto 0);
```

```
                a_out : out STD_LOGIC_VECTOR(31 downto 0);
                b_out : out STD_LOGIC_VECTOR(31 downto 0);
                c_out : out STD_LOGIC_VECTOR(31 downto 0);
                d_out : out STD_LOGIC_VECTOR(31 downto 0);
                e_out : out STD_LOGIC_VECTOR(31 downto 0);
                f_out : out STD_LOGIC_VECTOR(31 downto 0);
                g_out : out STD_LOGIC_VECTOR(31 downto 0);
                h_out : out STD_LOGIC_VECTOR(31 downto 0);
                i_out : out STD_LOGIC_VECTOR(31 downto 0);
                j_out : out STD_LOGIC_VECTOR(31 downto 0)
            );
end component Line_Distributer;

Component BloG_calc is
        port(
                a : in STD_LOGIC_VECTOR(31 downto 0);
                b : in STD_LOGIC_VECTOR(31 downto 0);
                c : in STD_LOGIC_VECTOR(31 downto 0);
                d : in STD_LOGIC_VECTOR(31 downto 0);
                e : in STD_LOGIC_VECTOR(31 downto 0);
                f : in STD_LOGIC_VECTOR(31 downto 0);
                g : in STD_LOGIC_VECTOR(31 downto 0);
                h : in STD_LOGIC_VECTOR(31 downto 0);
                i : in STD_LOGIC_VECTOR(31 downto 0);
                j : in STD_LOGIC_VECTOR(31 downto 0);
               data_out : out STD_LOGIC_VECTOR(0 to 35)
            );
end component BloG_calc;

component edgeDecision is
       port (
               data_in: in STD_LOGIC_VECTOR (35 downto 0);
               clk: in STD_LOGIC;
               enable_in: in STD_LOGIC;
               reset: in STD_LOGIC;
               enable_write: out STD_LOGIC:='0';
               done : out std_logic:='0';
               data_out: out STD_LOGIC_VECTOR (31 downto 0)
       );
end component edgeDecision;

       type word_array is array(natural range <>) of std_logic_vector(31 downto 0);
       signal  ram_line_1,ram_line_2,ram_line_3,ram_line_4,ram_line_5  :  word_array(0  to
22);
       signal ram_we_1,ram_we_2,ram_we_3,ram_we_4,ram_we_5: std_logic:='0' ;
       signal
ram_out_11,ram_out_12,ram_out_21,ram_out_22,ram_out_31,ram_out_32,ram_out_41,ram_out_42,
ram_out_51,ram_out_52 :std_logic_vector(31 downto 0);
       signal ram_addr_rd :std_logic_vector(5 downto 0):="000000";
       signal ram_addr_rd_wr :std_logic_vector(5 downto 0):="000000";
       signal ram_1_en,ram_2_en,ram_3_en,ram_4_en,ram_5_en : std_logic:='0';
       signal distr_counter,line_counter,row_counter,total_rows: integer:=0;
       signal data_in_counter: integer:=0;
       signal replace_flag:integer:=5;
       SUBTYPE reg_width IS STD_LOGIC_VECTOR(31 DOWNTO 0);
       TYPE reg10x32 IS ARRAY (0 to 9) OF reg_width;
       signal row_counter_vec:std_logic_vector(3 downto 0);
       signal temp_line,blog_in : reg10x32;
       type STATE_TYPE is (Idle, First_Write, replace_line, CalcWrite, stop);
       signal state : STATE_TYPE:=Idle;
       signal write_permit: std_logic:='0';
       signal read_permit: std_logic:='0';
       signal read_req_d,data_ready_t,data_ready_t2,done,edgeDesReset: std_logic:='0';
       signal data_in_t1 : std_logic_vector(31 downto 0);
```

```
	signal data_out_temp : std_logic_vector(35 downto 0);
	signal pieces : integer :=0;

begin

-- Block Ram Cells --
process (clk)
begin
	if (clk'event and clk = '1') then
		if (ram_1_en = '1') then
			if ram_we_1 = '1' then
				ram_line_1(conv_integer(ram_addr_rd_wr)) <= data_in_t1;
			end if;
			ram_out_11 <= ram_line_1(conv_integer(ram_addr_rd));
			ram_out_12 <= ram_line_1(conv_integer(ram_addr_rd_wr));
		end if;
	end if;
end process;

process (clk)
begin
	if (clk'event and clk = '1') then
		if (ram_2_en = '1') then
			if ram_we_2 = '1' then
				ram_line_2(conv_integer(ram_addr_rd_wr)) <= data_in_t1;
			end if;
			ram_out_21 <= ram_line_2(conv_integer(ram_addr_rd));
			ram_out_22 <= ram_line_2(conv_integer(ram_addr_rd_wr));
		end if;
	end if;
end process;

process (clk)
begin
	if (clk'event and clk = '1') then
		if (ram_3_en = '1') then
			if ram_we_3 = '1' then
				ram_line_3(conv_integer(ram_addr_rd_wr)) <= data_in_t1;
			end if;
			ram_out_31 <= ram_line_3(conv_integer(ram_addr_rd));
			ram_out_32 <= ram_line_3(conv_integer(ram_addr_rd_wr));
		end if;
	end if;
end process;

process (clk)
begin
	if (clk'event and clk = '1') then
		if (ram_4_en = '1') then
			if ram_we_4 = '1' then
				ram_line_4(conv_integer(ram_addr_rd_wr)) <= data_in_t1;
			end if;
			ram_out_41 <= ram_line_4(conv_integer(ram_addr_rd));
			ram_out_42 <= ram_line_4(conv_integer(ram_addr_rd_wr));
		end if;
	end if;
end process;

process (clk)
begin
	if (clk'event and clk = '1') then
		if (ram_5_en = '1') then
			if ram_we_5 = '1' then
				ram_line_5(conv_integer(ram_addr_rd_wr)) <= data_in_t1;
```

```
                    end if;
                    ram_out_51 <= ram_line_5(conv_integer(ram_addr_rd));
                    ram_out_52 <= ram_line_5(conv_integer(ram_addr_rd_wr));
             end if;
      end if;
end process;

process (clk)
begin
      if clk'event and clk='1' then
             data_in_t1<=data_in;
             data_ready_t2<=data_ready_t;
      --     data_in_t2<=data_in_t1;
      end if;
end process;

process(clk)
begin
      if clk'event and clk='1' then
             if reset='1' then
                    state<=Idle;
             else
             case state is

                    when stop => null;

                    when idle =>
                    if data_availiable='1' then
                           state<=First_write;
                           ram_1_en<='1';
                           ram_we_1<='1';
                           line_counter<=0;
                           row_counter<=0;
                           total_rows<=0;
                           distr_counter<=0;
                           read_permit <= '0';
                           write_permit<='0';
                    end if;


                    when First_write=>
                           if row_counter=5 and line_counter=23 then
                                  read_permit<='0';
                                  ram_5_en<='0';
                                  ram_we_5<='0';
                                  ram_addr_rd<="000000";
                                  ram_addr_rd_wr<="000001";
                                  state<=CalcWrite;

      ram_1_en<='0';ram_2_en<='0';ram_3_en<='0';ram_4_en<='0';ram_5_en<='0';

      ram_we_1<='0';ram_we_2<='0';ram_we_3<='0';ram_we_4<='0';ram_we_5<='0';
                                  line_counter<=0;
                                  read_permit <= '0';
                           else
                                  read_permit <= '1';
                           end if;

                           if line_counter=line_length/4 then
                                        line_counter<=0;

      ram_1_en<='0';ram_2_en<='0';ram_3_en<='0';ram_4_en<='0';ram_5_en<='0';

      ram_we_1<='0';ram_we_3<='0';ram_we_3<='0';ram_we_4<='0';ram_we_5<='0';
```

```
                        end if;


                        ram_addr_rd_wr<=conv_std_logic_vector(line_counter,6);
                        if data_availiable='1' and read_permit='1' then
                                line_counter<=line_counter+1;
                                data_in_counter<=data_in_counter+1;
                                if line_counter=line_length/4-1 then
                                        line_counter<=line_counter+1;
                                        row_counter<=row_counter+1;
                                        total_rows<=total_rows+1;

ram_1_en<='1';ram_2_en<='1';ram_3_en<='1';ram_4_en<='1';ram_5_en<='1';

ram_we_1<='0';ram_we_3<='0';ram_we_3<='0';ram_we_4<='0';ram_we_5<='0';
                                end if;

                                -- next state
                                if row_counter=5 then
                                        ram_5_en<='0';
                                        ram_we_5<='0';
                                        ram_addr_rd<="000000";
                                        ram_addr_rd_wr<="000001";
                                        state<=CalcWrite;

ram_1_en<='1';ram_2_en<='1';ram_3_en<='1';ram_4_en<='1';ram_5_en<='1';
                                        line_counter<=0;
                                        read_permit <= '0';
                                end if;
                                case row_counter is
                                        when 0 =>
                                                -- gemizw tin ram_line_1
                                                ram_1_en<='1';
                                                ram_we_1<='1';
                                        when 1 =>
                                                -- gemizw tin ram_line_2
                                                ram_1_en<='0';
                                                ram_we_1<='0';
                                                ram_2_en<='1';
                                                ram_we_2<='1';
                                        when 2 =>
                                                -- gemizw tin ram_line_3
                                                ram_2_en<='0';
                                                ram_we_2<='0';
                                                ram_3_en<='1';
                                                ram_we_3<='1';
                                        when 3 =>
                                                -- gemizw tin ram_line_4
                                                ram_3_en<='0';
                                                ram_we_3<='0';
                                                ram_4_en<='1';
                                                ram_we_4<='1';
                                        when 4 =>
                                                -- gemizw tin ram_line_5
                                                ram_4_en<='0';
                                                ram_we_4<='0';
                                                ram_5_en<='1';
                                                ram_we_5<='1';
                                        when others => null;
                                end case;

                        end if;

                when CalcWrite =>
```

```
                write_permit<= '1';
                read_permit<='0';
                if write_ack='0' then

ram_1_en<='1';ram_2_en<='1';ram_3_en<='1';ram_4_en<='1';ram_5_en<='1';
                                if line_counter=line_length/4-1 then
                                        if done='1' then
                                                if total_rows=row_length then
                                                        pieces<=pieces+1;
                                                        if pieces=3 then
                                                                state<=stop;
                                                        else
                                                                state<=idle;
                                                        end if;
                                                else
                                                        state<=replace_line;
                                                end if;
                                                line_counter<=0;
                                        end if;

ram_5_en<='0';ram_4_en<='0';ram_3_en<='0';ram_2_en<='0';ram_1_en<='0';

ram_we_5<='0';ram_we_4<='0';ram_we_3<='0';ram_we_2<='0';ram_we_1<='0';
                                        ram_addr_rd_wr<="000000";
                                        ram_addr_rd<="000000";
                                        read_permit<='0';
                                else
                                        line_counter<=line_counter+1;

ram_addr_rd<=conv_std_logic_vector(line_counter,6);

ram_addr_rd_wr<=conv_std_logic_vector(line_counter+1,6);
                                end if;
                        end if;

                when replace_line =>
                        write_permit<= '0';

                        if data_availiable='1' then
                                read_permit<='1';
                        end if;

                        if line_counter=line_length/4 then
                                read_permit <= '0';

                        end if;

                        if line_counter=23 then
                                null;
                        else

ram_addr_rd_wr<=conv_std_logic_vector(line_counter,6);
                        end if;


                        if line_counter=line_length/4 then
                                state<=CalcWrite;
                                line_counter<=0;
                                        row_counter<=row_counter+1;
                                        total_rows<=total_rows+1;

ram_1_en<='1';ram_2_en<='1';ram_3_en<='1';ram_4_en<='1';ram_5_en<='1';

ram_we_5<='0';ram_we_4<='0';ram_we_3<='0';ram_we_2<='0';ram_we_1<='0';
```

```
                                        ram_addr_rd_wr<="000001";
                                        ram_addr_rd<="000000";
                                        read_permit <= '0';
                                        if row_counter=9 then
                                                row_counter<=5;
                                        end if;

                                end if;


                                if data_availiable='1' and read_permit='1' then
                                        line_counter<=line_counter+1;
                                        data_in_counter<=data_in_counter+1;

                                        case row_counter is
                                                when 5 =>
                                                        -- gemizw tin ram_line_1
                                                        ram_1_en<='1';
                                                        ram_we_1<='1';
                                                when 6 =>
                                                        -- gemizw tin ram_line_2
                                                        ram_2_en<='1';
                                                        ram_we_2<='1';
                                                when 7 =>
                                                        -- gemizw tin ram_line_3
                                                        ram_3_en<='1';
                                                        ram_we_3<='1';
                                                when 8 =>
                                                        -- gemizw tin ram_line_4
                                                        ram_4_en<='1';
                                                        ram_we_4<='1';
                                                when 9 =>
                                                        -- gemizw tin ram_line_5
                                                        ram_5_en<='1';
                                                        ram_we_5<='1';
                                                when others => null;
                                        end case;
                                end if;
                        when others => null;
                end case;
                end if;
        end if;
end process;

temp_line(0)<=ram_out_11;
temp_line(1)<=ram_out_12;
temp_line(2)<=ram_out_21;
temp_line(3)<=ram_out_22;
temp_line(4)<=ram_out_31;
temp_line(5)<=ram_out_32;
temp_line(6)<=ram_out_41;
temp_line(7)<=ram_out_42;
temp_line(8)<=ram_out_51;
temp_line(9)<=ram_out_52;

read_req    <=(data_availiable    and    read_permit)    when    state=First_write    or
state=replace_line
                else '0';
read_req_d <=(data_availiable and read_permit) when state=First_write or
                state=replace_line else '0';
data_ready_t <=((not write_ack) and write_permit) when state=CalcWrite else '0';
edgeDesReset <='0' when state=CalcWrite else '1';
row_counter_vec <=conv_std_logic_vector(row_counter,4);
```

```
p1: Line_distributer port map
       (temp_line(0),temp_line(1),temp_line(2),temp_line(3),temp_line(4),

       temp_line(5),temp_line(6),temp_line(7),temp_line(8),temp_line(9),
       row_counter_vec,blog_in(0),blog_in(1),blog_in(2),blog_in(3),
       blog_in(4),blog_in(5),blog_in(6),blog_in(7),blog_in(8),blog_in(9));

p2:BloG_calc port map(blog_in(0),blog_in(1),blog_in(2),blog_in(3),blog_in(4),
                blog_in(5),blog_in(6),blog_in(7),blog_in(8),blog_in(9),
                                                        data_out_temp);

d1:edgeDecision port map
              (data_out_temp,clk,data_ready_t2,edgeDesReset,data_ready,done,data_out);

end Edge_detector_BRAM;
```

Ταξινόμηση των block ram

```
--------------------------------------------------------------------------------
--
-- Title       : Line_Distributer
-- Design      : Edge_detector_BLoG
-- Author      : Nikolaos Anastasiadis
-- Company     : Personal Computer System
--
--------------------------------------------------------------------------------
--
-- File        : Line_Distributer.vhd
-- Generated   : Thu Apr 17 17:50:49 2008
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.20
--
--------------------------------------------------------------------------------
--
-- Description :
--
--------------------------------------------------------------------------------

--{{ Section below this comment is automatically maintained
--   and may be overwritten
--{entity {Line_Distributer} architecture {Line_Distributer}}

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Line_Distributer is
        port(
               a : in STD_LOGIC_VECTOR(31 downto 0);
               b : in STD_LOGIC_VECTOR(31 downto 0);
               c : in STD_LOGIC_VECTOR(31 downto 0);
               d : in STD_LOGIC_VECTOR(31 downto 0);
               e : in STD_LOGIC_VECTOR(31 downto 0);
               f : in STD_LOGIC_VECTOR(31 downto 0);
               g : in STD_LOGIC_VECTOR(31 downto 0);
               h : in STD_LOGIC_VECTOR(31 downto 0);
               i : in STD_LOGIC_VECTOR(31 downto 0);
               j : in STD_LOGIC_VECTOR(31 downto 0);
               row_counter : in STD_LOGIC_VECTOR(3 downto 0);
               a_out : out STD_LOGIC_VECTOR(31 downto 0);
               b_out : out STD_LOGIC_VECTOR(31 downto 0);
               c_out : out STD_LOGIC_VECTOR(31 downto 0);
```

```
               d_out : out STD_LOGIC_VECTOR(31 downto 0);
               e_out : out STD_LOGIC_VECTOR(31 downto 0);
               f_out : out STD_LOGIC_VECTOR(31 downto 0);
               g_out : out STD_LOGIC_VECTOR(31 downto 0);
               h_out : out STD_LOGIC_VECTOR(31 downto 0);
               i_out : out STD_LOGIC_VECTOR(31 downto 0);
               j_out : out STD_LOGIC_VECTOR(31 downto 0)
            );
end Line_Distributer;

--}} End of automatically maintained section

architecture Line_Distributer of Line_Distributer is
begin

        -- enter your statements here --
        process(a,b,c,d,e,f,g,h,i,j)
        begin
        case row_counter is
                when "0101"=>
                a_out<= a;
                b_out<= b;
                c_out<= c;
                d_out<= d;
                e_out<= e;
                f_out<= f;
                g_out<= g;
                h_out<= h;
                i_out<= i;
                j_out<= j;
                when "0110"=>
                a_out<=c;
                b_out<=d;
                c_out<=e;
                d_out<=f;
                e_out<=g;
                f_out<=h;
                g_out<=i;
                h_out<=j;
                i_out<=a;
                j_out<=b;
                when "0111"=>
                a_out<=e;
                b_out<=f;
                c_out<=g;
                d_out<=h;
                e_out<=i;
                f_out<=j;
                g_out<=a;
                h_out<=b;
                i_out<=c;
                j_out<=d;
                when "1000"=>
                a_out<=g;
                b_out<=h;
                c_out<=i;
                d_out<=j;
                e_out<=a;
                f_out<=b;
                g_out<=c;
                h_out<=d;
                i_out<=e;
                j_out<=f;
                when "1001"=>
```

```
                a_out<=i;
                b_out<=j;
                c_out<=a;
                d_out<=b;
                e_out<=c;
                f_out<=d;
                g_out<=e;
                h_out<=f;
                i_out<=g;
                j_out<=h;
                when others => null;
        end case;
        end process;


end Line_Distributer;
```

Υλοποίηση του γινομένου με F2=-0.3125

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_unsigned.all;


entity f2_calc is
                port(
                        p : in std_logic_vector(7 downto 0);
                        p_out : out std_logic_vector(7 downto 0);
                        cout: out std_logic);
end f2_calc;

architecture f2_calc_rtl of f2_calc is


signal a_shift,b_shift,p_out_temp : std_logic_vector(7 downto 0):="00000000";

begin
        -- pixel*2^-2 calculation
        a_shift <= "00" & p(7 downto 2);
        -- pixel*2^-4 calculation
        b_shift <= ('0','0','0','0',p(7),p(6),p(5),p(4));

        p_out<=not (a_shift+b_shift) +1;

end f2_calc_rtl;
```

Μονοπάτι δεδομένων

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_signed.all;
use ieee.std_logic_arith.all;



entity BloG_calc is
         port(
                 a : in STD_LOGIC_VECTOR(31 downto 0);
```

```
               b : in STD_LOGIC_VECTOR(31 downto 0);
               c : in STD_LOGIC_VECTOR(31 downto 0);
               d : in STD_LOGIC_VECTOR(31 downto 0);
               e : in STD_LOGIC_VECTOR(31 downto 0);
               f : in STD_LOGIC_VECTOR(31 downto 0);
               g : in STD_LOGIC_VECTOR(31 downto 0);
               h : in STD_LOGIC_VECTOR(31 downto 0);
               i : in STD_LOGIC_VECTOR(31 downto 0);
               j : in STD_LOGIC_VECTOR(31 downto 0);
               data_out : out STD_LOGIC_VECTOR(35 downto 0)
            );
end BloG_calc;

---                   input tip                     ---- ATTENTION ----

--      input registers
--      a       b
--      c       d
--      e       f
--      h       i
--      j       k                     all registers are [31:0]

--}} End of automatically maintained section

architecture BloG_calc_rtl of BloG_calc is

component f2_calc is
               port(
                      p : in std_logic_vector(7 downto 0);
                      p_out : out std_logic_vector(7 downto 0);
                      cout: out std_logic);
end component f2_calc;


SUBTYPE reg_width IS STD_LOGIC_VECTOR(63 DOWNTO 0);
TYPE reg5x64 IS ARRAY (4 DOWNTO 0) OF reg_width;
Subtype reg_width_8 is std_logic_vector(7 downto 0);
type reg13x8 is array (12 downto 0) of reg_width_8;
Subtype reg_width_9 is std_logic_vector(8 downto 0);
type reg4x9 is array (3 downto 0) of reg_width_9;
type reg13x9 is array (12 downto 0) of reg_width_9;
Signal blog_reg : reg5x64;
Signal reg_1,reg_2,reg_3,reg_4 : reg13X8;
Signal reg_1_ext,reg_2_ext,reg_3_ext,reg_4_ext : reg13X9;
signal overflow,overflow2,overflow3,overflow4: std_logic_vector(7 downto 0);
signal data_out_temp : reg4x9;
signal data_out_temp2 : std_logic_vector(31 downto 0);

begin

       -- enter your statements here --
       blog_reg(4)(31 downto 0)<=b;
       blog_reg(4)(63 downto 32)<=a;
       blog_reg(3)(31 downto 0)<=d;
       blog_reg(3)(63 downto 32)<=c;
       blog_reg(2)(31 downto 0)<=f;
       blog_reg(2)(63 downto 32)<=e;
       blog_reg(1)(31 downto 0)<=h;
       blog_reg(1)(63 downto 32)<=g;
       blog_reg(0)(31 downto 0)<=j;
       blog_reg(0)(63 downto 32)<=i;

       -- calculating f1*a[i,j] for the first convolution
       reg_1(12)<= '0' & blog_reg(3)(23 downto 17);  --d3
```

```
reg_1(11)<= '0' & blog_reg(2)(31 downto 25);  --c4
reg_1(10)<= '0' & blog_reg(2)(23 downto 17);  --c3
reg_1(9)<= '0' & blog_reg(2)(15 downto 9);  --c2
reg_1(8)<= '0' & blog_reg(1)(23 downto 17);  --b3
-- calculating f2*a[i,j] for the first convolution
f1: f2_calc port map (blog_reg(4)(23 downto 16),reg_1(7),overflow(0)); --e3
f2: f2_calc port map (blog_reg(3)(31 downto 24),reg_1(6),overflow(1)); --d4
f3: f2_calc port map (blog_reg(3)(15 downto 8),reg_1(5),overflow(2)); --d2
f4: f2_calc port map (blog_reg(2)(39 downto 32),reg_1(4),overflow(3)); --c5
f5: f2_calc port map (blog_reg(2)(7 downto 0),reg_1(3),overflow(4)); --c1
f6: f2_calc port map (blog_reg(1)(31 downto 24),reg_1(2),overflow(5)); --b4
f7: f2_calc port map (blog_reg(1)(15 downto 8),reg_1(1),overflow(6)); --b2
f8:  f2_calc  port  map  (blog_reg(0)(23  downto  16),reg_1(0),overflow(7));  --a3


-- calculating f1*a[i,j] for the second convolution
reg_2(12)<= '0' & blog_reg(3)(31 downto 25);  --d4
reg_2(11)<= '0' & blog_reg(2)(39 downto 33);  --c5
reg_2(10)<= '0' & blog_reg(2)(31 downto 25);  --c4
reg_2(9)<= '0' & blog_reg(2)(23 downto 17);  --c3
reg_2(8)<= '0' & blog_reg(1)(31 downto 25);  --b4
-- calculating f2*a[i,j] for the second convolution
g1: f2_calc port map (blog_reg(4)(31 downto 24),reg_2(7),overflow2(0)); --e4
g2: f2_calc port map (blog_reg(3)(39 downto 32),reg_2(6),overflow2(1)); --d5
g3: f2_calc port map (blog_reg(3)(23 downto 16),reg_2(5),overflow2(2)); --d3
g4: f2_calc port map (blog_reg(2)(47 downto 40),reg_2(4),overflow2(3)); --c6
g5: f2_calc port map (blog_reg(2)(15 downto 8),reg_2(3),overflow2(4)); --c2
g6: f2_calc port map (blog_reg(1)(39 downto 32),reg_2(2),overflow2(5)); --b5
g7: f2_calc port map (blog_reg(1)(23 downto 16),reg_2(1),overflow2(6)); --b1
g8: f2_calc port map (blog_reg(0)(31 downto 24),reg_2(0),overflow2(7)); --a2

-- calculating f1*a[i,j] for the 3rd convolution
reg_3(12)<= '0' & blog_reg(3)(39 downto 33);  --d5
reg_3(11)<= '0' & blog_reg(2)(47 downto 41);  --c6
reg_3(10)<= '0' & blog_reg(2)(39 downto 33);  --c5
reg_3(9)<= '0' & blog_reg(2)(31 downto 25);  --c4
reg_3(8)<= '0' & blog_reg(1)(39 downto 33);  --b5
-- calculating f2*a[i,j] for the 3rd convolution
h1: f2_calc port map (blog_reg(4)(39 downto 32),reg_3(7),overflow3(0)); --e5
h2: f2_calc port map (blog_reg(3)(47 downto 40),reg_3(6),overflow3(1)); --d6
h3: f2_calc port map (blog_reg(3)(31 downto 24),reg_3(5),overflow3(2)); --d4
h4: f2_calc port map (blog_reg(2)(55 downto 48),reg_3(4),overflow3(3)); --c7
h5: f2_calc port map (blog_reg(2)(23 downto 16),reg_3(3),overflow3(4)); --c3
h6: f2_calc port map (blog_reg(1)(47 downto 40),reg_3(2),overflow3(5)); --b6
h7: f2_calc port map (blog_reg(1)(31 downto 24),reg_3(1),overflow3(6)); --b4--
h8: f2_calc port map (blog_reg(0)(39 downto 32),reg_3(0),overflow3(7)); --a5--

-- calculating f1*a[i,j] for the 4th convolution
reg_4(12)<= '0' & blog_reg(3)(47 downto 41);  --d6
reg_4(11)<= '0' & blog_reg(2)(55 downto 49);  --c7
reg_4(10)<= '0' & blog_reg(2)(47 downto 41);  --c6
reg_4(9)<= '0' & blog_reg(2)(39 downto 33);  --c5
reg_4(8)<= '0' & blog_reg(1)(47 downto 41);  --b6
-- calculating f2*a[i,j] for the 4th convolution
k1: f2_calc port map (blog_reg(4)(47 downto 40),reg_4(7),overflow4(0)); --e6
k2: f2_calc port map (blog_reg(3)(55 downto 48),reg_4(6),overflow4(1)); --d7
k3: f2_calc port map (blog_reg(3)(39 downto 32),reg_4(5),overflow4(2)); --d5
k4: f2_calc port map (blog_reg(2)(63 downto 56),reg_4(4),overflow4(3)); --c8
k5: f2_calc port map (blog_reg(2)(31 downto 24),reg_4(3),overflow4(4)); --c4
k6: f2_calc port map (blog_reg(1)(55 downto 48),reg_4(2),overflow4(5)); --b7
k7: f2_calc port map (blog_reg(1)(39 downto 32),reg_4(1),overflow4(6)); --b5--
k8: f2_calc port map (blog_reg(0)(47 downto 40),reg_4(0),overflow4(7)); --a6--
process(reg_1,reg_2,reg_3,reg_4)
begin
```

```
                for i in 0 to 12 loop
                        reg_1_ext(i)(7 downto 0)<=reg_1(i);
                        reg_1_ext(i)(8)<=reg_1(i)(7);
                        reg_2_ext(i)(7 downto 0)<=reg_2(i);
                        reg_2_ext(i)(8)<=reg_2(i)(7);
                        reg_3_ext(i)(7 downto 0)<=reg_3(i);
                        reg_3_ext(i)(8)<=reg_3(i)(7);
                        reg_4_ext(i)(7 downto 0)<=reg_4(i);
                        reg_4_ext(i)(8)<=reg_4(i)(7);
                end loop;
        end process;
        -- 1st conv result
data_out_temp(0)<=reg_1_ext(0)+reg_1_ext(1)+reg_1_ext(2)+reg_1_ext(3)+reg_1_ext(4)+
        reg_1_ext(5)+reg_1_ext(6)+reg_1_ext(7)+reg_1_ext(8)+reg_1_ext(9)+reg_1_ext(10)+
        reg_1_ext(11)+reg_1_ext(12);
        --2nd conv result
data_out_temp(1)<=reg_2_ext(0)+reg_2_ext(1)+reg_2_ext(2)+reg_2_ext(3)+reg_2_ext(4)+
        reg_2_ext(5)+reg_2_ext(6)+reg_2_ext(7)+reg_2_ext(8)+reg_2_ext(9)+reg_2_ext(10)+
        reg_2_ext(11)+reg_2_ext(12);
        --3rd conv result
data_out_temp(2)<=reg_3_ext(0)+reg_3_ext(1)+reg_3_ext(2)+reg_3_ext(3)+reg_3_ext(4)+
        reg_3_ext(5)+reg_3_ext(6)+reg_3_ext(7)+reg_3_ext(8)+reg_3_ext(9)+reg_3_ext(10)+
        reg_3_ext(11)+reg_3_ext(12);
        --4th conv result
data_out_temp(3)<=reg_4_ext(0)+reg_4_ext(1)+reg_4_ext(2)+reg_4_ext(3)+reg_4_ext(4)+
        reg_4_ext(5)+reg_4_ext(6)+reg_4_ext(7)+reg_4_ext(8)+reg_4_ext(9)+reg_4_ext(10)+
        reg_4_ext(11)+reg_4_ext(12);
data_out(8 downto 0) <=  data_out_temp(3);
data_out(17 downto 9) <=  data_out_temp(2);
data_out(26 downto 18) <=  data_out_temp(1);
data_out(35 downto 27) <=  data_out_temp(0);

end BloG_calc_rtl;
```

Απόφαση για ύπαρξη ακμών και συμπίεση εξόδου.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity edgeDecision is
port (
        data_in: in STD_LOGIC_VECTOR (35 downto 0);
        clk: in STD_LOGIC;
        enable_in: in STD_LOGIC;
        reset: in STD_LOGIC;
        enable_write: out STD_LOGIC:='0';
        done : out std_logic:='0';
        data_out: out STD_LOGIC_VECTOR (31 downto 0)
);
end edgeDecision;

--}} End of automatically maintained section

architecture edgeDecisionBeh of edgeDecision is

signal out_register : std_logic_vector(31 downto 0);
signal pixel : std_logic_vector(3 downto 0);
signal counter,rep_counter : integer:=0;
signal self_reset : std_logic:='0';
```

```
begin

process(clk)
begin


	if clk'event and clk='1' then
		if rep_counter=3 then
			done<='1';
		end if;
		if reset='1' or self_reset='1' then
			out_register<=(others=>'0');
			enable_write<='0';
			self_reset<='0';
			counter<=0;
			rep_counter<=0;
			done<='0';
		else
			if counter=0 then
				enable_write<='0';
			end if;

			if enable_in='1' then
				if      conv_integer(data_in(8 downto 0))>15 then
					pixel(3)<='1';
				else
					pixel(3)<='0';
				end if;
				if      conv_integer(data_in(17 downto 9))>15 then
					pixel(2)<='1';
				else
					pixel(2)<='0';
				end if;
				if      conv_integer(data_in(26 downto 18))>15 then
					pixel(1)<='1';
				else
					pixel(1)<='0';
				end if;
				if      conv_integer(data_in(35 downto 27))>15 then
					pixel(0)<='1';
				else
					pixel(0)<='0';
				end if;
				out_register(31  downto  4)<=out_register(27  downto  0);

				out_register(3 downto 0)<=pixel;
				counter<=counter+1;
				if counter=8 and rep_counter=0 then
					rep_counter<=rep_counter+1;
					counter<=0;
					enable_write<= '1';
				end if;
				if counter=7 and rep_counter=1 then
					rep_counter<=rep_counter+1;
					counter<=0;
					enable_write<= '1';
				end if;
				if rep_counter=2 and counter=5 then
					rep_counter<=rep_counter+1;
					counter<=0;
					enable_write<= '1';
				end if;
			end if;
```

```
                end if;
        end if;
end process;

data_out<=out_register;


end edgeDecisionBeh;
```

Οδηγός του συνεπεξεργαστή σε C

```
#include "xparameters.h"
#include "stdio.h"
#include "stdlib.h"
#include "xbasic_types.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"
#include "string.h"
#include "xtmrctr_l.h"
#include "fsl.h"
#include "mb_interface.h"
#include "microblaze_interrupts_i.h"


Xuint32 data[1748];

char readOneByte(void)
{
	return (char)XUartLite_RecvByte(STDIN_BASEADDRESS);
}

int main(void)
{

	char a[3];
	unsigned char mode;
	Xuint32 t1,t2,t3,t4,k1;
	int i,j,k,p,m,h,s;
	const Xuint8 black=0x00;
	const Xuint8 white=0xff;
	const Xuint32 mask8=0x000000ff;
	const Xuint32 mask1=0x00000001;
	Xuint32 inp,timer_counter;
	int u;
	Xuint32 pixelfeed,outpixels,cc;
	Xuint8 temp[4];


inp=0x0;
while(1){
	XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0x01));
	mode=readOneByte();
	xil_printf("mode is : %c \r",mode);
	if(mode=='t')
		break;
}
//	mode='t';
XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0x04));
u=0;
for(k=0;k<4;k++){
//		transmision mode
	if (mode=='t')
	{
//		XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0x04));
		for(i=0;i<1748;i++){
//			for(i=0;i<76;i++){
//				for(j=0;j<23;j++){
					a[0]=readOneByte();
					a[1]=readOneByte();
					a[2]=readOneByte();
					temp[3]=atoi(a);
					a[0]=readOneByte();
```

```
                        a[1]=readOneByte();
                        a[2]=readOneByte();
                        temp[2]=atoi(a);
                        a[0]=readOneByte();
                        a[1]=readOneByte();
                        a[2]=readOneByte();
                        temp[1]=atoi(a);
                        a[0]=readOneByte();
                        a[1]=readOneByte();
                        a[2]=readOneByte();
                        temp[0]=atoi(a);
                        t4=(Xuint32)temp[3] ;
                        t4=(t4 << 24)& 0xff000000;
                        t3=(Xuint32)temp[2];
                        t3=(t3 << 16) & 0x00ff0000;
                        t2=(Xuint32)temp[1];
                        t2=(t2 << 8) & 0x0000ff00;
                        t1=(Xuint32)temp[0];
                        t1=t1 & 0x000000ff;
                        data[i] = t4 | t3 | t2 | t1;
                }
        }
        //send image data to IP
        i=0;
        XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0x66));
        //set load register
        XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_0_BASEADDR,0,(Xuint32)0x00);
        //set control status register
        XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_0_BASEADDR,0,
                XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_ENABLE_INT_MASK );
        //mb enable interupts
        microblaze_enable_interrupts();
        //start timer
        XTmrCtr_mEnable(XPAR_OPB_TIMER_0_BASEADDR,0);
        XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0x0f));
        for(p=0;p<76;p++){
                putfsl(data[i],0);putfsl(data[i+1],0);putfsl(data[i+2],0);
                putfsl(data[i+3],0);putfsl(data[i+4],0);putfsl(data[i+5],0);
                putfsl(data[i+6],0);putfsl(data[i+7],0);putfsl(data[i+8],0);
                putfsl(data[i+9],0);putfsl(data[i+10],0);putfsl(data[i+11],0);
                putfsl(data[i+12],0);putfsl(data[i+13],0);putfsl(data[i+14],0);
                putfsl(data[i+15],0);putfsl(data[i+16],0);putfsl(data[i+17],0);
                putfsl(data[i+18],0);putfsl(data[i+19],0);putfsl(data[i+20],0);
                        putfsl(data[i+21],0);putfsl(data[i+22],0);
                        i=i+23;
                if(p>3)
                {       //harvest values
                        m=(p-4)*3;
                        getfsl(data[m],0);
                        getfsl(data[m+1],0);
                        getfsl(data[m+2],0);
                }
        }


        //stop timer
        XTmrCtr_mDisable(XPAR_OPB_TIMER_0_BASEADDR,0);
        //get cc
        cc = XTmrCtr_mGetTimerCounterReg(XPAR_OPB_TIMER_0_BASEADDR,0);
        xil_printf("%d\r",timer_counter);
        xil_printf("%d\r",cc);
        XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,0x99);
        for(m=0;m<72;m++){
                XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)m);
```

```
                        for(s=31;s>=0;s--){
                                k1=( data[m*3] >> s ) & mask1;
                                xil_printf("%d\r",k1);
                        }
                        for(s=31;s>=0;s--){
                                k1=( data[m*3+1] >> s ) & mask1;
                                xil_printf("%d\r",k1);
                        }
                        for(s=23;s>=0;s--){
                                k1=( data[m*3+2] >> s ) & mask1;
                                xil_printf("%d\r",k1);
                        }
                }


}
XGpio_mWriteReg(XPAR_LEDS_8BIT_BASEADDR,0,(Xuint32)(0xff));
return 1;
}
```

Κώδικας σε Matlab

Προσομοίωση της κάμερας

```
clc;
close all;
clear all;

I=imread('hatsBaham.tif');
I=double(rgb2gray(I));
%I=rgb2gray(I);

%fi_table=fi(I,0,8);
%hex_table=hex(fi_table);
%I=round(255*checkerboard(24));
M_t=I(1:144,1:176);

[ylength,xlength] = size(M_t); % determines size of input image
%% side pading
M(3:ylength+2,3:xlength+2)=M_t(1:ylength,1:xlength);
%%right, left padding
M(1,3:xlength+2)=M_t(1,:);M(2,3:xlength+2)=M_t(1,:);
M(ylength+3,3:xlength+2)=M_t(ylength,:);M(ylength+4,3:xlength+2)=M_t(ylength,:);

%up, down padding
M(3:ylength+2,1)=M_t(:,1);M(3:ylength+2,2)=M_t(:,1);
M(3:ylength+2,xlength+3)=M_t(:,xlength);M(3:ylength+2,xlength+4)=M_t(:,xlength);
I=M;


%For write operation
serobjw = serial('COM7') ;

%Set connection properties
serobjw.Baudrate = 9600;
set(serobjw, 'Parity', 'none') ;
set(serobjw, 'Databits', 8) ;
set(serobjw, 'StopBits', 1) ;
set(serobjw, 'Terminator', 'CR') ;
set(serobjw, 'OutputBufferSize', 50000) ;
set(serobjw, 'InputBufferSize', 50000) ;
set(serobjw, 'ReadAsyncMode', 'Continuous');

get(serobjw) ;
fopen(serobjw) ;
get(serobjw, 'Status')
set(serobjw, 'Timeout', 5) ;

% send mode state to FPGA
%mode = fscanf(serobjw,'%c')
%msg = fscanf(serobjw,'%c')
fprintf(serobjw,'%c','t') ;
mode = fscanf(serobjw,'%c')



for(u=1:2)
    sprintf('start transmision')
    for (i=(1+(u-1)*72):(76+(u-1)*72))
        for(j=1:92)
            fprintf(serobjw,'%03d',I(i,j)) ;
```

```
        end;
    end;
%
     timer = fscanf(serobjw,'%d')
     cc = fscanf(serobjw,'%d')

    sprintf('start harvesting')
    for(m=(1+(u-1)*72):(72+(u-1)*72))
        for(l=1:88)%88
            vhdl_image(m,l) = fscanf(serobjw,'%d');
        end;
    end;


%    figure, imshow(vhdl_image);
    %%%%%%%%%%%%%%%% right part of image %%%%%%%%%%%%%%%%
    sprintf('start transmision')
    for (i=(1+(u-1)*72):(76+(u-1)*72))
        for(j=89:180)
            fprintf(serobjw,'%03d',I(i,j)) ;
        end;
    end;

    timer = fscanf(serobjw,'%d')
    cc = fscanf(serobjw,'%d')

    sprintf('start harvesting')
    for(m=(1+(u-1)*72):(72+(u-1)*72))
        for(l=89:176)%88
            vhdl_image(m,l) = fscanf(serobjw,'%d');
        end;
    end;


end;
%figure, imshow(vhdl_image);

%dlmwrite('vhdl_output.txt', vhdl_image, 'delimiter', ' ', 'precision', 3, 'newline',
'pc');

% close serial connection
fclose(serobjw) ;
delete(serobjw);
clear serobjw;

output_image=BloG(M_t(1:144,1:176));
%output_image=LoG(M_t(1:144,1:176));
[ylength,xlength] = size(output_image);

edges_matlab=(output_image>15);
edges_dif=vhdl_image-double(edges_matlab);

fig3=figure;
subplot(2,2,2);
imshow(vhdl_image);
title('edk edges');
subplot(2,2,3);
imshow(edges_matlab);
title('matlab edges');
subplot(2,2,4);
imshow(abs(edges_dif));
title('error edges');
subplot(2,2,1)
imshow(M/255);
```

```
title('input image')

[X,Y]=meshgrid(1:xlength,1:ylength);
figure, mesh(X,Y,abs(edges_dif));
title('error plot');
```

Υπολογισμός του Bilevel laplacian of Gaussian φίλτρου

```
function output_image=BloG(in_image);

%%%in_image is a grayscale image of any dimensions
%%%output_image are the image edges detected with BloG

M_t=double(in_image);
f1=0.5;
f2=-0.3125;

K=[0 0 f2 0 0;
   0 f2 f1 f2 0;
   f2 f1 f1 f1 f2;
   0 f2 f1 f2 0;
   0 0 f2 0 0];
[ylength,xlength] = size(M_t); % determines size of input image
%% side pading
M(3:ylength+2,3:xlength+2)=M_t(1:ylength,1:xlength);
%%right, left padding
M(1,3:xlength+2)=M_t(1,:);M(2,3:xlength+2)=M_t(1,:);
M(ylength+3,3:xlength+2)=M_t(ylength,:);M(ylength+4,3:xlength+2)=M_t(ylength,:);

%up, down padding
M(3:ylength+2,1)=M_t(:,1);M(3:ylength+2,2)=M_t(:,1);
M(3:ylength+2,xlength+3)=M_t(:,xlength);M(3:ylength+2,xlength+4)=M_t(:,xlength);

[ylength,xlength] = size(M);

output_image(1:ylength-4,1:xlength-4) = zeros; %inits output_image
vhdl_image(1:ylength-4,1:xlength-4) = zeros;
%output_image_8(1:ylength,1:xlength) = zeros; %inits output_image_8
% loops to simulate SE window passing over image

% ypologismos idanikis syneliksis
for y=1:(ylength-4)
    for x=1:(xlength-4)
        window = [M(y:(y+4),x:(x+4))];
        mult = window.*K;
        add = sum(sum(mult));
        output_image(y,x) = add;

    end
end
```