

Ειδική Επιστημονική Εργασία  
**ΥΛΟΠΟΙΗΣΗ ΣΕ FPGA ΤΟΥ**  
**ΠΕΡΙΓΡΑΦΕΑ HOG**  
**ΓΙΑ ΑΝΙΧΝΕΥΣΗ ΑΝΘΡΩΠΩΝ**  
**ΣΕ ΕΙΚΟΝΕΣ ΚΑΙ BINTEO**

**ΑΝΤΩΝΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ**

A.M:144

Επιβλέπων: Ευάγγελος Ζυγούρης

Αναπλ. Καθηγητής



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ ΤΜΗΜΑ ΦΥΣΙΚΗΣ  
ΠΑΤΡΑ, ΑΥΓΟΥΣΤΟΣ 2013



Ειδική Επιστημονική Εργασία (MScThesis)

**Υλοποίηση σε FPGA του περιγραφέα HOG για  
ανίχνευση ανθρώπων σε εικόνες και βίντεο**

Γεώργιος Αντωνόπουλος  
ΑΜ: 144

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή  
την..... 2013, στα πλαίσια του ΔΠΜΣ  
Ηλεκτρονική και Επεξεργασία της Πληροφορίας

---

Ευάγγελος Ζυγούρης  
Αναπληρωτής Καθηγητής  
Τμήμα Φυσικής

Παν/μιο Πατρών

---

Γεώργιος Οικονόμου  
Καθηγητής  
Τμήμα Φυσικής

Παν/μιο Πατρών

---

Δημήτριος Μπακάλης  
Επίκουρος Καθηγητής  
Τμήμα Φυσικής

Παν/μιο Πατρών

Πάτρα, Αύγουστος 2013



# Πρόλογος

Η παρούσα ειδική ερευνητική εργασία εκπονήθηκε στα πλαίσια του Διατμηματικού Προγράμματος Μεταπτυχιακών Σπουδών στην “Ηλεκτρονική και Επεξεργασία της Πληροφορίας”, στο Τμήμα Φυσικής του Πανεπιστημίου Πατρών. Αντικείμενο της παρούσας εργασίας είναι η “Υλοποίηση σε FPGA του περιγραφέα HOG για ανίχνευση ανθρώπων σε εικόνες και βίντεο”.

Το πρώτο κεφάλαιο αποτελεί μια εισαγωγή στις βασικότερες που χρησιμοποιούνται στην παρούσα εργασία. Περιγράφεται επίσης η αναπτυξιακή πλακέτα που χρησιμοποιήθηκε καθώς και τα επί μέρους στοιχεία που τη συνθέτουν. Τέλος γίνεται μια συνοπτική αναφορά σε εργασίες με παρόμοιο αντικείμενο, οι οποίες με επηρέασαν στο σχεδιασμό και την υλοποίηση του συστήματός μου.

Στο δεύτερο κεφάλαιο αναλύεται ο περιγραφέας Ιστογραμμάτων Προσανατολισμού της Βάθμωσης ή όπως είναι ευρύτερα γνωστός Histograms of Oriented Gradient Descriptor. Παρουσιάζονται τα βήματα όπως περιγράφονται στην εργασία των Dalal&Triggs[4] και οι βέλτιστες τιμές των παραμέτρων του περιγραφέα.

Στο τρίτο κεφάλαιο ακολουθώντας τα βήματα του δευτέρου κεφαλαίου, παρουσιάζεται η διαδικασία υλοποίησης του περιγραφέα στο Matlab. Εκτός της υλοποίησης έγινε και μια προεργασία για τη μεταφορά του σε γλώσσα περιγραφής υλικού. Η προεργασία αυτή περιλαμβάνει απλοποιήσεις και τροποποιήσεις με σκοπό να μειωθεί το υπολογιστικό κόστος. Τέλος παρουσιάζονται τα αποτελέσματα δοκιμών της απόδοσης του περιγραφέα για τις διάφορες απλοποιήσεις.

Στο τέταρτο κεφάλαιο γίνεται μια μικρή αναφορά στους ταξινομητές. Περιγράφονται οι ταξινομητές που δοκιμάστηκαν στην παρούσα εργασία ως προς συγκεκριμένα χαρακτηριστικά τους καθώς και την υπολογιστική τους πολυπλοκότητα για την συγκεκριμένη εφαρμογή.

Το πέμπτο και τελευταίο κεφάλαιο περιλαμβάνει την περιγραφή της υλοποίησης σε VHDL. Αναλύονται τα επί μέρους κυκλώματα και όπου κρίθηκε αναγκαίο χρησιμοποιήθηκαν σχήματα ή πίνακες. Σε κάποιες περιπτώσεις δίνονται και οι κυματομορφές των κυκλωμάτων.

Θέλω να ευχαριστήσω το Βαγγέλη Γιαννακόπουλο για όσα μου έδωσε όσο ήταν “εδώ”, αλλά και για όσα συνεχίζει να μου δίνει μέσα από αυτά που με δίδαξε. Επίσης θέλω να

ευχαριστήσω τον επιβλέποντα μου κ. Ε. Ζυγούρη (και να του ζητήσω συγνώμη για τις “ψυχολογικές” δοκιμασίες που τον υπέβαλα) και τους Δ. Καστανιώτη και Δρ. Δ. Μπεσύρη όπου χωρίς τη συνεισφορά τους δεν θα ολοκληρωνόταν αυτή η εργασία.

Γιώργος Αντωνόπουλος

Πάτρα 2013

# Introduction

This thesis took place within the frame work of the Interdepartemental Master's Program in "Electronics and Information Processing", at the Department of Physics of University of Patras. The objective of this work is the implementation in FPGA of the HOG descriptor for the detection of people, images and videos.

The first chapter is an introduction about the basic concepts, which are used across the manuscript. (Additional descriptions concern the development board which was used as well as the individual parts that compose it.) In the end, there is a brief reference to past projects focusing on similar objectives, which influenced the design and the implementation of my system.

The second chapter concerns the presentation and discussion of the Histograms of Oriented Gradient descriptor. The steps of the procedure and the best parameter values of the descriptor are presented in a similar way as they are described in the paper of Dalal and Triggs.

In the third chapter, following the steps of the previous one, the focus shifts to the descriptor's implementation procedure in Matlab. Besides the implementation, there is a preparation for the transference of the descriptor in a Hardware Description Language. This preparation includes simplifications and modifications aiming at the reduction of the computational cost. Finally, we see the tests' results of the descriptor's performance concerning the various simplifications.

The fourth chapter is a partial reference to the classifiers. The description is about the classifiers that were used in the present work with respect to their features and their computational complexity of this particular application.

The fifth and final chapter refers to the description of the implementation in VHDL. There is an analysis of the partial circuits and, when necessary, shapes and tables were used. In some cases, the waveforms of the circuits are being presented.

Georgios Antonopoulos  
Patra 2013



# Περιεχόμενα

Πρόλογος.....	v
Introduction.....	vii
Περιεχόμενα.....	ix
Κεφάλαιο 1.....	1
Συστήματα πραγματικού χρόνου και FPGAs .....	1
1.1 Εισαγωγή .....	1
1.2 Field Programmable Gate Arrays .....	3
1.3 Είσοδος/Εξοδος βίντεο.....	6
1.4 Το ολοκληρωμένο CycloneII.....	7
1.4.1 Κανονική Λειτουργία.....	8
1.4.2 Αριθμητική Λειτουργία .....	9
1.4.3 Μνήμη .....	9
1.4.4 Προσπέλαση της μνήμης.....	11
1.4.5 Πολλαπλασιαστές .....	12
1.5 Παράλληλη επεξεργασία και Pipelining.....	13
1.6 Histograms of Oriented Gradient σε FPGA .....	15
Κεφάλαιο 2.....	17
Ο περιγραφέας Ιστογραμμάτων Προσανατολισμού της Βάθμωσης (Histograms of Oriented Gradients - HOG descriptor).....	17
2.1 Εισαγωγή .....	17
2.2 Ανίχνευση Πεζών.....	18
2.2.1 Αλγόριθμοι ανίχνευσης αντικειμένων .....	18
2.2.2 Κατηγορίες αλγορίθμων ανίχνευσης Πεζών .....	19
2.3 Ιστογράμματα Προσανατολισμού της Βάθμωσης (Histograms of Oriented Gradients) .....	21
2.3.1 Γάμμα εξισορρόπηση και κανονικοποίηση χρώματος .....	22
2.3.2 Υπολογισμός Βάθμωσης και Μέτρου .....	23
2.3.3 Δημιουργία Ιστογραμμάτων Προσανατολισμού της Βάθμωσης.....	24
2.3.4 Κανονικοποίηση σε επικαλυπτόμενα block.....	26
Κεφάλαιο 3.....	31
Υλοποίηση του περιγραφέα Ιστογραμμάτων Προσανατολισμού της Βάθμωσης στο Matlab .....	31
3.1 Εισαγωγή .....	31
3.2 To Matlab .....	32

3.2.1 Γάμμα εξισορρόπηση και κανονικοποίηση χρώματος .....	33
3.2.2 Εξαγωγή Βαθμώσεων και Μέτρων .....	33
3.2.3 Υπολογισμός του Bin .....	35
3.2.4 Δημιουργία Ιστογραμμάτων .....	38
3.2.5 Κανονικοποίηση σε επικαλυπτόμενα blocks .....	39
3.3 Απλοποιήσεις στην υλοποίηση του περιγραφέα .....	42
3.3.1 Αντικατάσταση Μέτρων .....	43
3.3.2 Αντικατάσταση L2 Νόρμας με Μέσο όρο τιμών .....	44
3.3.3 Αύξηση του μεγέθους του Cell και υπο-δειγματοληψία .....	45
3.3.4 Κβάντιση εικονοστοιχείων .....	46
Κεφάλαιο 4 .....	49
Ταξινόμηση .....	49
4.1 Εισαγωγή .....	49
4.2 Ταξινομητές .....	50
4.3 Support Vector Machines .....	51
4.3.1 Γραμμικός SVM .....	52
4.4 Λογιστική Παλινδρόμηση – Logistic Regression .....	54
4.5 Σύγκριση Ταξινομητών .....	55
4.6 Υπολογισμός Κατωφλίου .....	58
4.6.1 Καμπύλες ROC .....	59
Κεφάλαιο 5 .....	63
Η υλοποίηση σε VHDL .....	63
5.1 Εισαγωγή .....	63
5.2 Έλεγχος Ορθής Λειτουργίας .....	64
5.3 Λήψη εικονοστοιχείων .....	65
5.4 Κύκλωμα υπολογισμού bin .....	67
5.4.1 Υπολογισμός του bin .....	67
5.4.2 Συγχρονισμός του αποτελέσματος .....	68
5.5 Δημιουργία Ιστογραμμάτων .....	70
5.5.1 Γενική λειτουργία υποκυκλώματος .....	70
5.5.2 Δεδομένα και μνήμη .....	71
5.5.3 Συστοιχίες Καταχωρητών .....	72
5.5.4 Άθροιση Μέτρων .....	77
5.6 Ανάγνωση μνήμης .....	78

5.7 Κύκλωμα Ταξινόμησης.....	81
5.8 Συνολικό κύκλωμα .....	84
Αποτέλεσμα.....	85
Συμπεράσματα – Προτάσεις .....	86
Βιβλιογραφία .....	89
Παράρτημα Προγραμμάτων Matlab.....	91
Κώδικας Υπολογισμού HOG στην πλήρη του εκδοχή .....	91
Εντοπισμός πεζών σε VGA Εικόνα.....	92
Bin selection – Κώδικας ελέγχου.....	94
Οπτικοποίηση των χαρακτηριστικών διανυσμάτων με VL-Feat (απαιτείται VL_feat βιβλιοθήκη). 94	94
Εκπαίδευση SVM ταξινομητή.....	95
Κώδικας Ελέγχου Ορθής Λειτουργία του κυκλώματος υπολογισμού Ιστογραμμάτων .....	96
Δημιουργία αρχείων αρχικοποίησης ROM για ταξινόμηση .....	96
ROC Curves .....	97
Παράρτημα Προγραμμάτων VHDL.....	99
Line Register .....	99
BIT BUFFER .....	102
N_BITS_SUB.....	103
Bin Selection .....	103
Histogram Generator.....	108
RAM read.....	112
Classifier .....	114
ROM.....	123

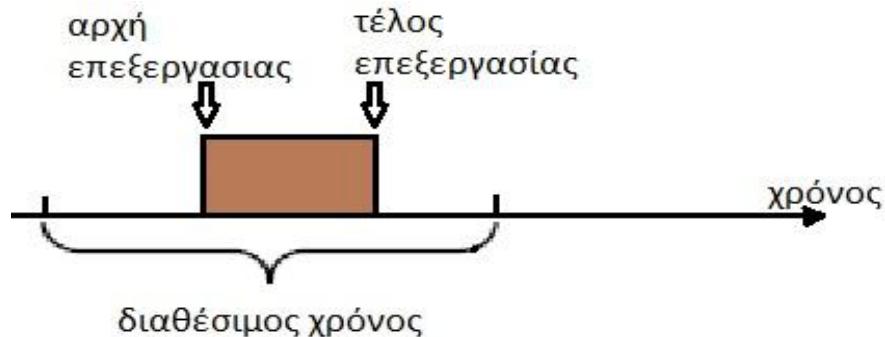


# Κεφάλαιο 1

## Συστήματα πραγματικού χρόνου και FPGAs

### 1.1 Εισαγωγή

Μια παράμετρος που καθορίζει σε μεγάλο βαθμό το σχεδιασμό ενός συστήματος είναι τα χρονικά περιθώρια που υπάρχουν για τις αποκρίσεις του συστήματος στα διάφορα εξωτερικά συμβάντα. Ένα σύστημα που χαρακτηρίζεται από αυστηρά καθορισμένη χρονική συνέπεια, ή αλλιώς που η σωστή λειτουργία του προϋποθέτει την τήρηση χρονικών προθεσμιών για κάθε λειτουργία του, λέγεται **σύστημα πραγματικού χρόνου**. Ένα σύστημα, λοιπόν, για να χαρακτηριστεί ως πραγματικού χρόνου δεν αρκεί να εξάγει σωστά αποτελέσματα. Πρέπει υποχρεωτικά να τα παρέχει στους χρόνους που έχουν καθοριστεί εξ αρχής.



**Σχήμα 1.1 Διεργασία σε σχέση με το χρόνο**

Η μη τήρηση των χρονικών ορίων σε ένα σύστημα πραγματικού χρόνου οδηγεί σε διαφορετικές καταστάσεις ανάλογα με το είδος του συστήματος. Όταν η “αθέτηση” των χρονικών περιορισμών απλά μειώνει την απόδοση του συστήματος, τότε πρόκειται για *Soft Real-Time* σύστημα. Όταν όμως η αθέτηση των χρονικών περιορισμών οδηγεί το σύστημα σε τελείως εσφαλμένη λειτουργία, τότε το σύστημα χαρακτηρίζεται ως *Hard Real-time*.

Μερικές πολύ διαδεδομένες εφαρμογές πραγματικού χρόνου είναι τα συστήματα επεξεργασίας σημάτων και εικόνας. Για παράδειγμα τα κινητά είναι συσκευές πραγματικού χρόνου, αφού το σήμα που λαμβάνεται από τη κεραία πρέπει να μετατραπεί σε ήχο σε συγκεκριμένο χρόνο έτσι ώστε να μπορεί να υπάρξει διάλογος.

Στην παρούσα εργασία σκοπός ήταν η δημιουργία ενός συστήματος πραγματικού χρόνου επεξεργασίας βίντεο. Η επεξεργασία αποσκοπεί στον εντοπισμό πεζών σε καρέ βίντεο. Η επεξεργασία βίντεο μπορεί να αναφέρεται στην επεξεργασία του κάθε frame χωριστά ή να πρόκειται για επεξεργασία πληροφορίας που προκύπτει από δύο διαδοχικά frame.

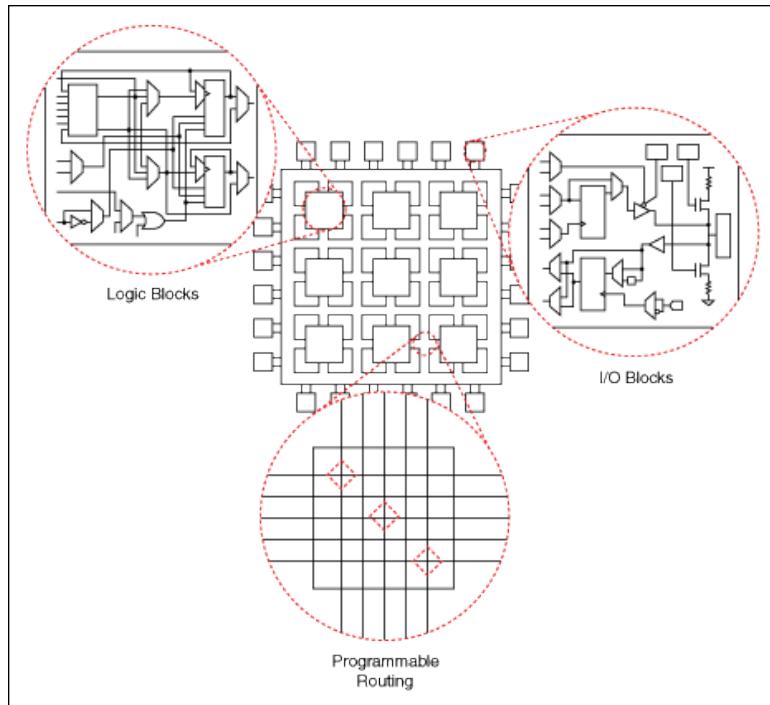
Στην επεξεργασία βίντεο σε πραγματικό χρόνο, τα χρονικά περιθώρια που έχει το σύστημα είναι πολύ συγκεκριμένα. Ο χρόνος που διατίθεται για την επεξεργασία του κάθε Frame είναι ίσος με το χρόνο που μεσολαβεί μέχρι τη λήψη του επομένου. Στο σύστημα που χρησιμοποιήθηκε στην παρούσα εργασία η κάμερα τροφοδοτεί το σύστημά με 30 frame το δευτερόλεπτο. Αυτό σημαίνει ότι η επεξεργασία του κάθε frame έπρεπε να γίνει μέσα σε 1/30 του δευτερολέπτου, δηλαδή σε περίπου 33,3 ms. Δεδομένης της μικρής ταχύτητας των πεζών η επεξεργασία μπορεί να γίνει για τα μισά Frame χωρίς να επηρεάζει το οπτικό αποτέλεσμα. Αυτό σημαίνει ότι διπλασιάζεται ο χρόνος που διατίθεται για την επεξεργασία της εικόνας.

Τέτοιου είδους επεξεργασίες είναι σχεδόν αδύνατον να υλοποιηθούν με ένα κοινό υπολογιστή μέσης επεξεργαστικής ισχύς. Ακόμα και να επιτευχθεί η επεξεργασία σε επιθυμητό χρόνο οι πόροι και κατά συνέπεια η κατανάλωση του συστήματος το καθιστούν ασύμφορο οικονομικά. Για το λόγο αυτό η επεξεργασία βίντεο γίνεται κυρίως με Επεξεργαστές Ψηφιακών Σημάτων, γνωστοί ως Digital Signal Processors, και με Field Programmable Gate Arrays (FPGAs). Στην παρούσα εργασία χρησιμοποιήθηκαν FPGAs για την υλοποίησή του συστήματος.

## 1.2 Field Programmable Gate Arrays

Τα Field Programmable Gate Arrays (FPGAs) είναι πίνακες λογικών στοιχείων με τους οποίους μπορούμε και φτιάχνουμε ολοκληρωμένα κυκλώματα συγκεκριμένου σκοπού. Η “μορφή” που θα πάρουν τα κυκλώματα περιγράφεται από γλώσσες προγραμματισμού περιγραφής υλικού (Hardware Description Languages). Μέσω κατάλληλου υλικού και λογισμικού ο κώδικας που έχει γραφτεί σε μια γλώσσα προγραμματισμού περιγραφής υλικού διαμορφώνει τους πίνακες με τα λογικά στοιχεία. Ένα από τα μεγάλα πλεονεκτήματα των FPGAs, είναι η δυνατότητάς τους να επαναπρογραμματίζονται.

Τα FPGAs έχουν διαφορετική αρχιτεκτονική από τις Σύνθετες Προγραμματιζόμενες Λογικές Διατάξεις (Complex Programmable Logic Devices). Η διασύνδεση μεταξύ των λογικών συστοιχιών του FPGA, τους δίνει μεγαλύτερη ευελιξία. Αυτή η ευελιξία δεν είναι άνευ κόστους. Έχει σαν αποτέλεσμα να κάνει τον προγραμματισμό του FPGA πιο περίπλοκο σε σχέση με τον προγραμματισμό ενός CPLD ενώ σε κάποιες περιπτώσεις προκαλεί και απρόβλεπτες καθυστερήσεις.



**Σχήμα 1.2.** Γενική περιγραφή της Εσωτερικής δομή ενός FPGA (origin-ars.els-cdn.com)

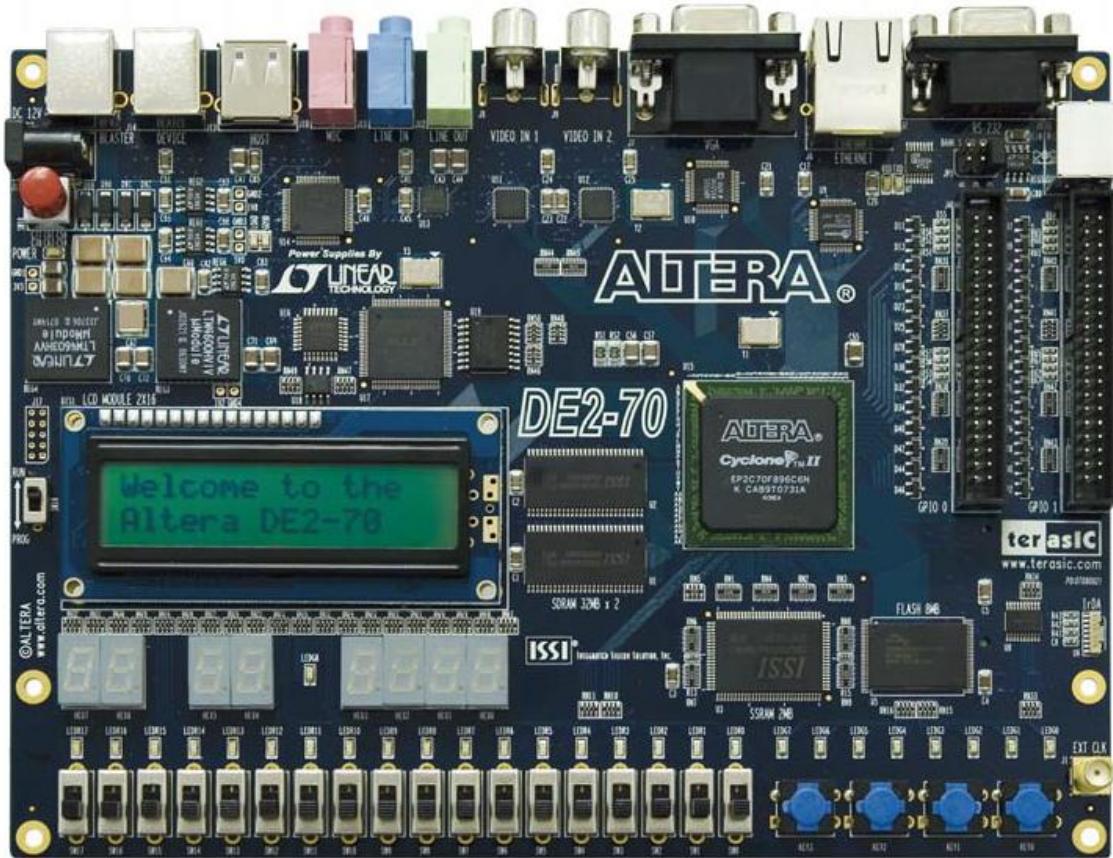
Ένα FPGA αποτελείται από προγραμματιζόμενα λογικά στοιχεία, οργανωμένα σε μπλοκ, τα οποία επικοινωνούν μεταξύ τους με προγραμματιζόμενες διασυνδέσεις, όπως φαίνεται στο παραπάνω σχήμα (Σχήμα 1.2). Εκτός των λογικών στοιχείων και των διασυνδέσεων, στα FPGAs περιέχονται και στοιχεία ψηφιακών επεξεργαστών έτοιμα προς χρήση. Τέτοια στοιχεία είναι μνήμες, πολλαπλασιαστές, βαθμίδες εισόδου/εξόδου και άλλα. Το πλήθος των λογικών στοιχείων, των διασυνδέσεων και των άλλων στοιχείων ποικίλει ανάλογα με τον κατασκευαστή και το μοντέλο. Για την εργασία αυτή χρησιμοποιήθηκε η πλατφόρμα της Altera DE2-70 και ο προγραμματισμός έγινε σε VHDL χρησιμοποιώντας το Quartus II 7.2. Τα χαρακτηριστικά της πλατφόρμας DE2-70 αναγράφονται παρακάτω και διατίθεται στην ιστοσελίδα της κατασκευάστριας εταιρίας:

**Πλατφόρμα DE2-70**

**Χαρακτηριστικά**

**Περιγραφή**

<b>FPGA</b>	Cyclone II EP2C70F896C6 with EPCS64 64-Mbit serial configuration device
<b>I/O Interfaces</b>	Ενσωματωμένο USB-Blaster για την παραμετροποίηση του FPGA Line In/Out, Microphone In (24-bit CODEC ήχου) Video Out (VGA 10-bit ψηφιοαναλογικό μετατροπέα) Video In (NTSC/PAL/Multi-format) (2 ports) RS232
	Θύρα υπερύθρων Θύρα PS/2 για πληκτρολόγιο ή ποντίκι 10/100 Ethernet USB 2.0 (type A and type B) Expansion headers (two 40-pin headers)
<b>Μνήμη</b>	64 MBSDRAM, 2 MBSSRAM, 8 MB Flash Υποδοχέα για κάρτα μνήμης SD
<b>Οθόνες</b>	Οχτώ οθόνες 7-segment Μία οθόνη 16 x 2 LCD
<b>Διακόπτες και LEDs</b>	18 toggle switches 18 κόκκινα LEDs 9 πράσινα LEDs Τέσσερα διακόπτες επαναφοράς
<b>Ρολόγια</b>	Ρολόι 50 MHz Ρολόι 27 MHz Είσοδο για εξωτερικό ρολόι SMA



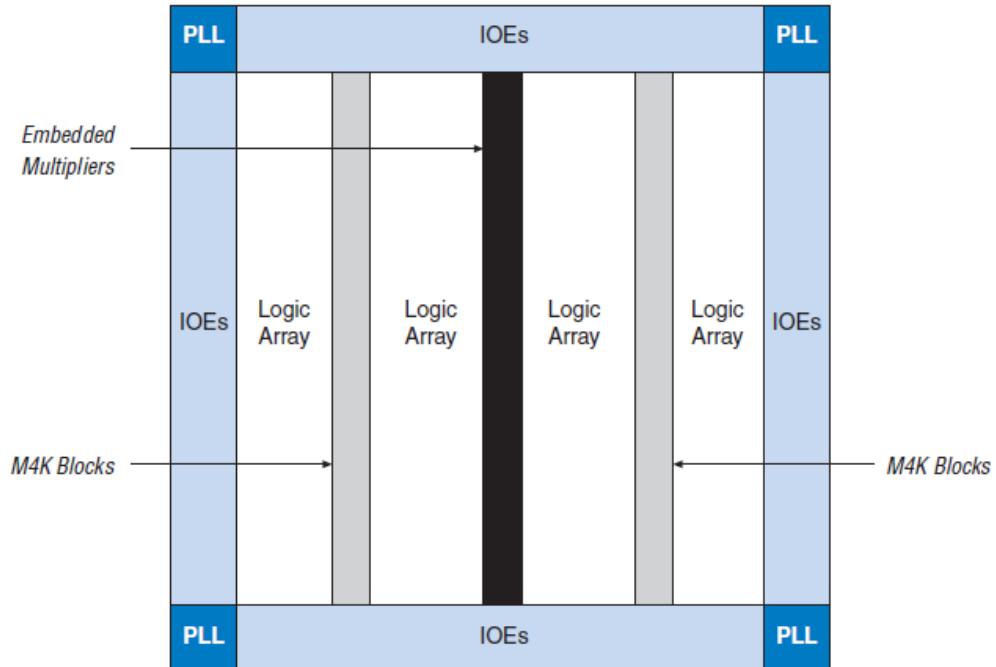
**Σχήμα 1.3** Η πάνω όψη της πλατφόρμας DE2 – 70

### 1.3 Είσοδος/Έξοδος βίντεο

Για την είσοδο και την έξοδο της εικόνας χρησιμοποιήθηκαν οι πόρτες που υπάρχουν ενσωματωμένες στο DE2 – 70, Video In και VGA OUT. Η είσοδος βίντεο στην πλατφόρμα γίνονται σύμφωνα με το πρότυπο ITU – 656. Το ITU – 656 περιγράφει ένα πρωτόκολλο για τη μετάδοση ασυμπίεστου PAL ή NTSC Σήματος Βασικής Ευκρίνειας. Τα δεδομένα αποτελούνται από λέξεις των 8 ή 10 bitsμε ρυθμό μετάδοσης 27 Mbytes/s και μπορούν να μεταδοθούν σειριακά αλλά και παράλληλα. Οι οριζόντιες γραμμές δεδομένων οριοθετούνται από σήματα μεγέθους τεσσάρων byte, που καλούνται SAV (Start of Active Video) και EAV (End of Active Video). Στο SAV εμπεριέχετε και πληροφορία για τη θέση της γραμμής στο Frame. Το ITU – 656 χρησιμοποιεί το χρωματικό χώρο YCbCr και παραμέτρους υποδειγματοληψίας 4:2:2.

## 1.4 Το ολοκληρωμένο Cyclone II

Εκτός των χαρακτηριστικών της πλακέτας είναι χρήσιμο να σταθούμε στα χαρακτηριστικά του FPGA (Cyclone II EP2C70) που αποτελεί τον “εγκέφαλο” της πλακέτας. Το πλήθος των στοιχείων που περιέχει είναι αυτό που θέτει τους περιορισμούς στην υλοποίησή μας.



**Σχήμα 1.4** Διάταξη στο εσωτερικό του Cyclone II ([www.altera.com](http://www.altera.com))

Το Cyclone II EP2C70 περιέχει 68.416 λογικά στοιχεία, 250 συστοιχίες μνήμης RAM μεγέθους 4Kbits η κάθε μία, 1.125 Kbits ενσωματωμένη μνήμη, 150 πολλαπλασιαστές των 18 x 18 bits τοποθετημένοι σε μια στήλη με συχνότητα λειτουργίας έως και 250 MHz, και 4 PLL. Τα λογικά στοιχεία είναι η κυριότερη δομική μονάδα του FPGA. Τα λογικά στοιχεία είναι τοποθετημένα σε συστοιχίες των 16 και αποτελούν ένα block λογικών στοιχείων. Τα blocks στοιχίζονται σε στήλες και μαζί με τις υπόλοιπες μονάδες διατάσσονται όπως στο Σχήμα 1.4, που παρατίθεται στο εγχειρίδιο από τον κατασκευαστή.

Το κάθε λογικό στοιχείο διαθέτει μικρο-μονάδες που προσφέρουν πολύ αποδοτικά χαρακτηριστικά ψηφιακής λογικής. Πιο αναλυτικά, το κάθε λογικό στοιχείο περιλαμβάνει:

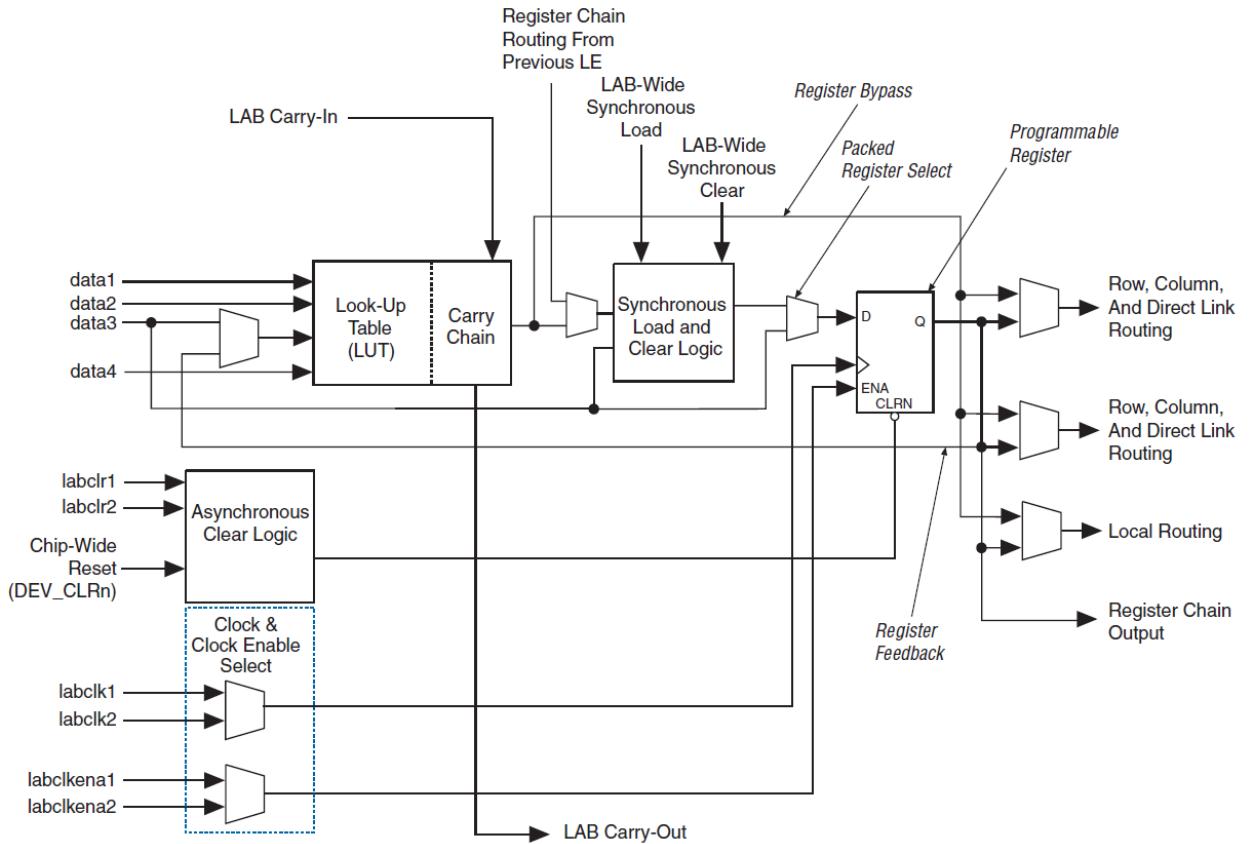
- Τρείς εξόδους για τη επικοινωνία μεταξύ των λογικών στοιχείων ή για την επικοινωνία τους με άλλα στοιχεία του FPGA. Οι έξοδοι αυτοί επιτρέπουν τη διασύνδεση όλων των ειδών: οριζόντια, κάθετη, μέσα στο block, αλυσίδας καταχωρητών κλπ.
- Τέσσερις Πίνακες Αντιστοίχισης Τιμών (Look – Up Table), που πραγματοποιούν οποιαδήποτε λογική πράξη για μέχρι και τέσσερις μεταβλητές. Οι πίνακες μπορούν να επικοινωνούν απευθείας με τις εξόδους του λογικού στοιχείου .
- Έναν προγραμματιζόμενο καταχωρητή που μπορεί να λειτουργήσει ως D, T, JK ή SR flip – flop. Ο κάθε καταχωρητής μπορεί να έχει εισόδους data, clock, clock enable, clear inputs.
- Άμεση έξοδο κρατουμένου.
- Άμεση έξοδο του καταχωρητή για διασύνδεσή του με καταχωρητές του ίδιου block.
- Ανατροφοδότηση της εισόδου από τον καταχωρητή.

Το σχηματικό διάγραμμα για κάθε λογικό στοιχείο φαίνεται παρακάτω (Σχήμα 1.5). Το διάγραμμα του κάθε λογικού στοιχείου μπορεί να διαμορφωθεί ανάλογα με τον τρόπο λειτουργίας του. Υπάρχουν δύο διαφορετικοί τρόποι που μπορεί να λειτουργήσει κάθε λογικό στοιχείο:

- Κανονική Λειτουργία (Normal Mode)
- Αριθμητική Λειτουργία (Arithmetic Mode)

#### **1.4.1 Κανονική Λειτουργία**

Η Κανονική Λειτουργία είναι κατάλληλη για εφαρμογές με λογικές πράξεις και συνδυαστικές εφαρμογές. Στην Κανονική Λειτουργία χρησιμοποιούνται τέσσερις είσοδοι και μια επιπλέον είσοδος, που συνδέεται στην έξοδο κρατουμένου άλλου στοιχείου με σκοπό τη συνεχόμενη διασύνδεση των δύο στοιχείων.



Σχήμα 1.5 Σχηματικό διάγραμμα Λογικού Στοιχείου ([www.altera.com](http://www.altera.com))

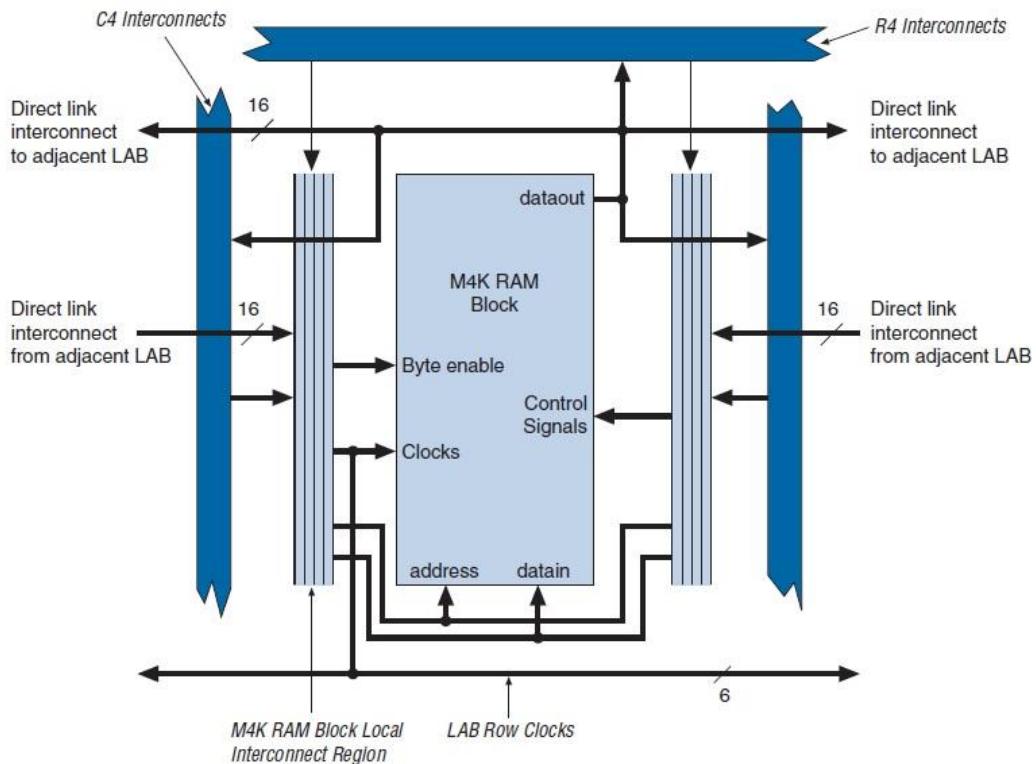
## 1.4.2 Αριθμητική Λειτουργία

Η Αριθμητική Λειτουργία είναι κατάλληλη για τη δημιουργία αθροιστών, μετρητών, συσσωρευτών και συγκριτών. Ένα λογικό στοιχείο στην αριθμητική λειτουργία δημιουργεί έναν πλήρη αθροιστή με έξοδο κρατουμένου. Τα δεδομένα μπορούν να διέρχονται από τον καταχωρητή ή να τον παρακάμπτουν. Τέλος σε αυτόν τον τρόπο λειτουργίας η έξοδος του καταχωρητή μπορεί να ανατροφοδοτεί την είσοδο.

## 1.4.3 Μνήμη

Όπως φαίνεται στο σχήμα 1.6 η μνήμη είναι τοποθετημένη σε στήλες. Οι στήλες της μνήμης απαρτίζονται από συστοιχίες μνήμης M4K. Σε κάθε συστοιχία M4K περιέχονται καταχωρητές οι οποίοι συγχρονίζουν τα δεδομένα που γράφονται ή διαβάζονται από αυτή. Οι καταχωρητές εξόδου μπορούν να αγνοηθούν από τον προγραμματιστή ενώ οι

καταχωρητές εισόδου όχι. Οι τελευταίοι χρησιμοποιούνται υποχρεωτικά για όλα τα δεδομένα που εγγράφονται.



**Σχήμα 1.6** Εσωτερική δομή συστοιχίας μνήμης M4K ([www.altera.com](http://www.altera.com))

Η διασύνδεση μιας συστοιχίας μνήμης M4K με τις παρακείμενες συστοιχίες λογικών στοιχείων φαίνεται στο Σχήμα 1.6. Η επικοινωνία μεταξύ τους επιτυγχάνεται μέσω των κάθετων και παράλληλων πόρων. Κάθε συστοιχία M4K συνδέεται με δεκαέξι εισόδους και δεκαέξι εξόδους σε κάθε πλευρά, με τις γειτονικές συστοιχίες λογικών στοιχείων.

Κάθε συστοιχία M4K, μπορεί να υλοποιήσει πολλά διαφορετικά ήδη μνήμης όπως διπλής θύρας (true & simple dual Port) RAM, μονής θύρας (single Port) ROM και FIFO (First In – First Out) Buffers. Πιο αναλυτικά, τα χαρακτηριστικά των M4K συστοιχιών έχουν ως εξής:

- 4608 bits
- 250 MHz συχνότητα λειτουργίας
- Διάφορους τρόπους προσπέλασης
- Ενεργοποίηση με Byte

- Ψηφίο Ισοτιμίας
- Καταχωρητή ολίσθησης
- FIFO Buffer
- ROM
- Διάφορες λειτουργίες χρονισμού
- Ενεργοποίηση μνήμης με χρονισμό.

#### **1.4.4 Προσπέλαση της μνήμης**

Η προσπέλαση της μνήμης μπορεί να γίνει με πολλούς τρόπους, όπως αναφέρθηκε επιγραμματικά προηγουμένων. Παρακάτω περιγράφονται οι διαφορετικοί τρόποι με τους οποίους μπορεί να προσπελαστεί η μνήμη μας. Ο τρόπος που προσπελαύνεται μια μνήμη είναι βαρύνουσας σημασίας στα συστήματα πραγματικού χρόνου. Δίνει τη δυνατότητα να χρησιμοποιείται η μνήμη από δύο διατάξεις την ίδια χρονική στιγμή, γεγονός που προσδίδει ευελιξία και ενισχύει την παράλληλη επεξεργασία του συστήματος.

Λειτουργία Μονής Θύρας: Γίνεται μία και μόνο λειτουργία κάθε χρονική στιγμή. Δεν μπορεί να γίνεται ταυτόχρονη εγγραφή και ανάγνωση.

Απλή λειτουργία Διπλής Θύρας: Υπάρχει δυνατότητα ταυτόχρονης εγγραφής και ανάγνωσης από τη μνήμη στην ίδια χρονική στιγμή.

Απλή λειτουργία Διπλής Θύρας Μεικτού εύρους: Έχει ίδια λειτουργία με την προηγούμενη με τη διαφορά ότι, η ανάγνωση και η εγγραφή μπορούν να έχουν δεδομένα διαφορετικού μεγέθους.

Πραγματική λειτουργία Διπλής Θύρας: Συνδυάζονται όλες οι εκδοχές διπλής λειτουργίας ανά χρονική στιγμή, όπως δύο εγγραφές, δύο αναγνώσεις ή ακόμα και μια εγγραφή και μια ανάγνωση με διαφορετικές συχνότητες.

Πραγματική λειτουργία Διπλής Θύρας Μεικτού εύρους: Έχει ακριβώς ίδια λειτουργία με την προηγούμενη με τη διαφορά ότι τα δεδομένα μπορούν να έχουν διαφορετικά μεγέθη.

Λειτουργία Ενσωματωμένου Καταχωρητή ολίσθησης: Χρησιμοποιούνται για ολίσθηση.

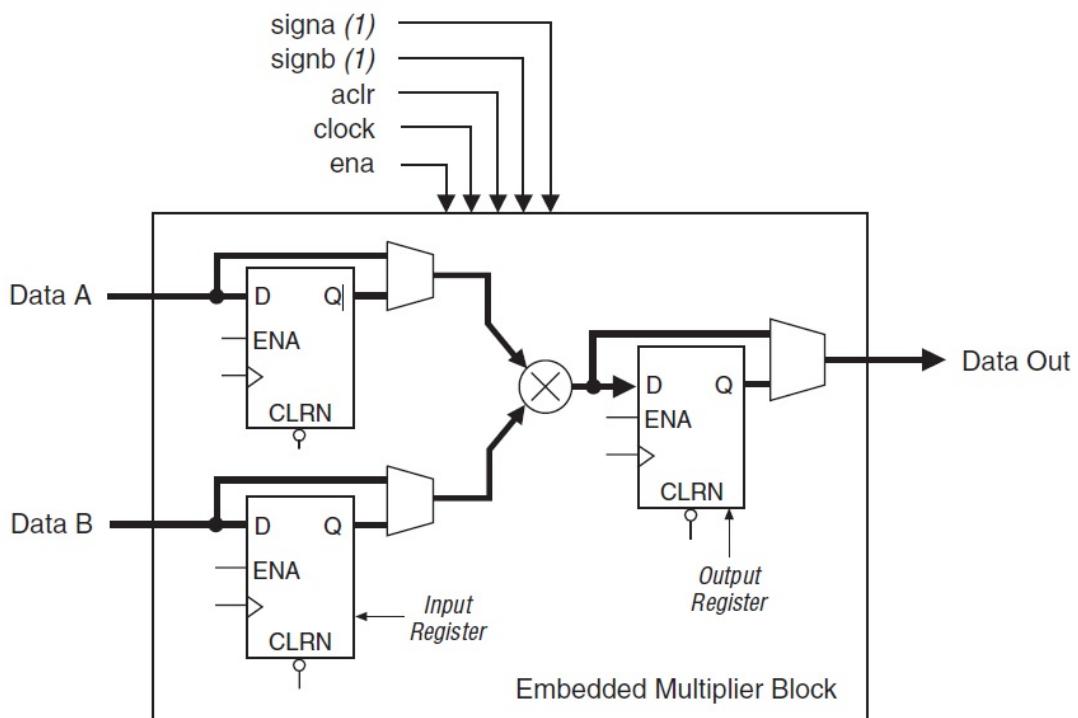
Read Only Memory (ROM): Λειτουργία μνήμης ROM. Η αρχικοποίηση της μνήμης γίνεται με αρχείο MIF.

FIFO buffers: FIFO με διπλό ή μονό χρονισμό.

### 1.4.5 Πολλαπλασιαστές

Κάθε ολοκληρωμένο Cyclone II περιέχει ενσωματωμένες τρεις στήλες με συστοιχίες που περιέχουν συνολικά 150 πολλαπλασιαστές. Είναι ειδικά σχεδιασμένες για την επεξεργασία ψηφιακών σημάτων όπως ο σχεδιασμός ψηφιακών φίλτρων πεπερασμένης κρουστικής απόκρουσης (FIR), συναρτήσεις που υλοποιούν Γρήγορο Μετασχηματισμό Φουριέ (FFT) και Διακριτό Μετασχηματισμό Συνημιτόνου.

Κάθε πολλαπλασιαστής, ανάλογα με την εφαρμογή, μπορεί να χρησιμοποιηθεί είτε σαν ένας πολλαπλασιαστής των 18 bit ή εναλλακτικά σαν δύο ανεξάρτητοι πολλαπλασιαστές των 9 bit. Οι πολλαπλασιαστές ανεξάρτητα με το εύρος των ψηφίων, μπορούν να λειτουργήσουν με ταχύτητα έως και 250 MHz. Οι πολλαπλασιαστέοι μπορούν να είναι προσημασμένοι ή μη προσημασμένοι αριθμοί και ορίζονται από τα σήματα signa και signb.



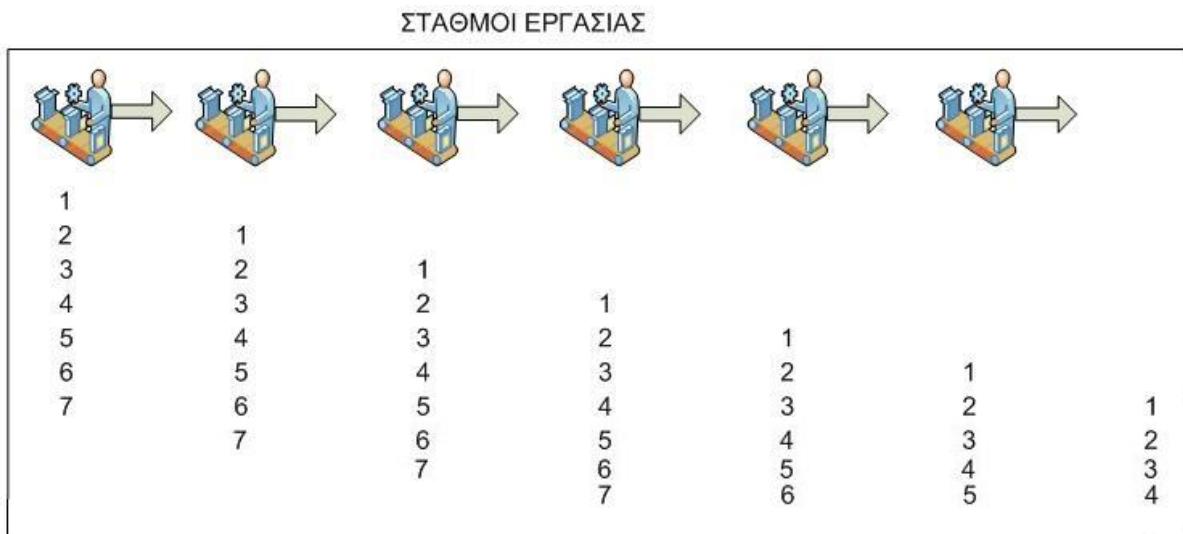
Σχήμα 1.7 Εσωτερική αρχιτεκτονική Πολλαπλασιαστή ([www.altera.com](http://www.altera.com))

Η υλοποίηση μιας εφαρμογής σε FPGA είναι η δημιουργία διατάξεων που πραγματοποιούν τους υπολογισμούς της εφαρμογής αυτής. Οι διατάξεις αυτές πρέπει να είναι συγχρονισμένες άριστα, έτσι ώστε να παρέχουν τα δεδομένα όπου χρειάζεται και στις κατάλληλες χρονικές στιγμές. Ο συγχρονισμός των δεδομένων αποτελεί μια επιπλέον δυσκολία αφού είναι ευαίσθητος σε αλλαγές του κυκλώματος.

Πριν υλοποιηθεί ένα κύκλωμα σε FPGA, μπορεί και πρέπει να γίνει μελέτη των αποτελεσμάτων του σε πρόγραμμα που υλοποιεί μαθηματικούς υπολογισμούς με προκαθορισμένη ακρίβεια. Με σωστή προσομοίωση μπορεί να προβλεφθεί το αποτέλεσμα που θα δώσει το τελικό κύκλωμα, με πολύ μεγάλη ακρίβεια. Αυτό που δεν γίνεται να μελετηθεί χρησιμοποιώντας μαθηματικά προγράμματα και μπορεί να δοκιμαστεί μόνο κατά την υλοποίηση του κυκλώματος, είναι ο συγχρονισμός των εισόδων και των εξόδων των επί μέρους διατάξεων του κυκλώματος. Εάν το κύκλωμα δεν έχει συγχρονισμένες με ακρίβεια διατάξεις, τότε αποτυγχάνει το Pipeline και κατά συνέπεια οι παράλληλοι υπολογισμοί, που είναι το κύριο πλεονέκτημα των FPGAs σε σχέση με άλλες διατάξεις υλοποίησης συστημάτων πραγματικού χρόνου.

## 1.5 Παράλληλη επεξεργασία και Pipelining

Η υπεροχή των FPGAs σε εφαρμογές πραγματικού χρόνου είναι η δυνατότητα παράλληλης επεξεργασίας. Η αρχιτεκτονική των κυκλωμάτων, δηλαδή, είναι τέτοια ώστε να λειτουργούν πολλές μονάδες του κυκλώματος ταυτόχρονα για την επεξεργασία διαφορετικών δεδομένων. Αυτό επιτυγχάνεται χρησιμοποιώντας την τεχνική του Pipeline (σωλήνωση). Το Pipeline μπορεί να παρομοιαστεί με τη γραμμή παραγωγής σε ένα εργοστάσιο όπου μια μονάδα τροφοδοτεί την επόμενη ενώ γίνονται πολλές διεργασίες ταυτόχρονα.



**Σχήμα 1.8** Γραμμή παραγωγής

Στο παραπάνω σχήμα φαίνονται έξι σταθμοί επεξεργασίας και τα δεδομένα προς επεξεργασία. Το δεδομένο -1- εισέρχεται, υπόκειται μια επεξεργασία και προωθείται στο επόμενο στάδιο. Τη θέση του -1- παίρνει το επόμενο δεδομένο, το -2-. Έτσι πλέον ο πρώτος σταθμός επεξεργάζεται το -2- και ο δεύτερος το -1-. Μόλις τελειώσει και αυτό το στάδιο τα δεδομένα προωθούνται στους επόμενους κατά αντιστοιχία σταθμούς και στον πρώτο εισέρχεται το δεδομένο -3-. Αυτή η διαδικασία συνεχίζεται όσο υπάρχουν δεδομένα. Η διάρκεια της κάθε επεξεργασίας θα είναι ίση με τη μέγιστη διάρκεια επεξεργασίας μεταξύ των σταθμών. Αν είναι λιγότερος ο χρόνος τότε ο “αργότερος” σταθμός δεν θα προλαβαίνει να επεξεργάζεται τα δεδομένα, με αποτέλεσμα να αποτυγχάνει το σύστημα.

Ας θεωρήσουμε ότι έχουμε  $\Sigma$  σε πλήθος σταθμούς και  $\Delta$  δεδομένα, το αργότερο στάδιο επεξεργασίας είναι ίσο με 5 sec και ο συνολικός χρόνος που απαιτείται για την επεξεργασία ενός δεδομένου είναι 20 sec τότε:

- Χωρίς Pipeline η διάρκεια επεξεργασίας είναι  $20 * \Delta$ .
- Με Pipeline ο χρόνος που απαιτείται είναι  $5 * \Sigma + \Delta - 1$ .

Είναι δηλαδή ο χρόνος που απαιτείται μέχρι να τελειώσει η επεξεργασία του πρώτου δεδομένου συν το πλήθος των εναπομεινάντων δεδομένων. Στην πράξη, λόγω του σχετικά μεγάλου όγκου δεδομένων και του μικρού πλήθους σταδίων επεξεργασίας, η επιτάχυνση της επεξεργασίας είναι σημαντικά μεγάλη.

Κατά συνέπεια βασική προϋπόθεση για αποδοτικό pipeline είναι η κατανομή των εργασιών σε περίπου ίσες χρονικά εργασίες ώστε να μην υπάρχουν μονάδες με μεγάλες στιγμές αδράνειας.

## 1.6 Histograms of Oriented Gradient σε FPGA

Ο περιγραφέας Ιστογραμμάτων Προσανατολισμένων Βαθμώσεων είναι ένας από τους πιο διαδεδομένους περιγραφείς αναγνώρισης αντικειμένου και έχει αποσπάσει τη προσοχή πολλών ερευνητών. Έχουν γίνει πολλές υλοποιήσεις του περιγραφέα σε γλώσσες υλικού με πολύ διαφορετικές αρχιτεκτονικές και ποικίλα αποτελέσματα. Παρακάτω αναφέρονται τα αποτελέσματα από μερικές εργασίες που έχουν δημοσιευτεί πάνω στο συγκεκριμένο αντικείμενο.

Το 2009 παρουσιάστηκε από τους Ryusuke Miyamoto και Yukihiro Nakamura [1] μια υλοποίηση σε Altera Startix II που προγραμματίστηκε σε Verilog. Στην εργασία παρουσιάστηκαν πολλές απλοποιήσεις και προσεγγίσεις, όπως για παράδειγμα:

$$\sqrt{2} = 1,375 \quad \text{και} \quad ||V_k||^2 + 1 = 2^{(n+1)/2}$$

Σκοπός των αλλαγών τους ήταν να έχουν να απλοποιήσουν την υπολογιστική πολυπλοκότητα. Στην υλοποίησή τους εφαρμόστηκε Pipeline τεχνική στην επεξεργασία των ιστογραμμάτων. Εν τέλει κατάφεραν να επεξεργαστούν 30 καρέ ανά δευτερόλεπτο χρησιμοποιώντας δέκα στιγμιότυπα της αρχιτεκτονικής που προτείνουν τα οποία δουλεύουν παράλληλα.

Στη εργασία “Implementation of real-time pedestrian detection on FPGA” των Tam Phuong Cao, Guang Deng, Mulligan David [2] παρουσιάζεται ένα σύστημα εξαιρετικά απλοποιημένο. Τα Bin είναι μειωμένα σε πέντε από εννέα που είναι το επικρατέστερο. Επίσης δεν συμπεριλαμβάνονται καθόλου τα μέτρα των Βαθμώσεων στη δημιουργία των ιστογραμμάτων. Στα συμπεράσματα αναφέρουν πως η ανακολουθία των εικόνων εκπαίδευσης σε σχέση με τις πραγματικές εικόνες έχει μεγάλη σημασία για την απόδοση του συστήματος.

Μια πολύ αξιόλογη εργασία παρουσιάστηκε το 2009 με τίτλο: "FPGA Implementation of a HOG –based Pedestrian Recognition System", γραμμένη από τους Sebastian Bauer, Ulrich Brunsmann και Stefan Schlotterbeck – Macht [3]. Το σύστημα αποτελείται από ένα δικτύωμα FPGA–CPU–GPU. Το FPGA χρησιμοποιήθηκε για την είσοδο των δεδομένων, την προεπεξεργασία της εικόνας, τον υπολογισμό των Βαθμώσεων και των μέτρων και τέλος για τη δημιουργία των ιστογραμμάτων. Στη CPU γινόταν η κανονικοποίηση του περιγραφέα και από κει τα δεδομένα μεταφέρονταν στην GPU για την ταξινόμηση και μετά πάλι πίσω στη CPU για την εξαγωγή του αποτελέσματος. Μεγάλο ενδιαφέρον παρουσιάζει η τεχνική με την οποία εξάγουν τα bin. Με συγκρίσεις και τέσσερις πολλαπλασιασμούς καταφέρνουν να αποφύγουν τις διαιρέσεις. Το αποτέλεσμα που επιτυγχάνεται είναι η επεξεργασία είκοσι καρέ ανά δευτερόλεπτο όπου σε κάθε καρέ εξετάζονται τριακόσια παράθυρα εντοπισμού. Για την περαιτέρω βελτίωση του συστήματος προτείνεται από τους συγγραφείς η προσθήκη επιπλέον μνήμης και FPGA από πλατφόρμα επέκτασης.

## Κεφάλαιο 2

# Ο περιγραφέας Ιστογραμμάτων Προσανατολισμού της Βάθμωσης (Histograms of Oriented Gradients - HOG descriptor)

### 2.1 Εισαγωγή

Ο περιγραφέας Ιστογραμμάτων Προσανατολισμού της Βάθμωσης (Histograms of Oriented Gradients) παρουσιάστηκε πρώτη φορά στην εργασία “Histograms of Oriented Gradients for Human Detection” από τον Nanveet Dalal και τον Bill Triggs το 2005 [4]. Κύριο γνώρισμα του είναι η ικανότητα να κωδικοποιεί τοπικά το σχήμα – περίγραμμα βρίσκοντας τις Βαθμώσεις των εικονοστοιχείων, λαμβάνοντας όμως υπόψη και το μέτρο του κάθε εικονοστοιχείου. Είναι ένας από τους πιο διαδεδομένους περιγραφείς για ανθρώπους, ζώα, οχήματα και γενικότερα για οτιδήποτε μπορεί να περιγραφεί με βάση το περίγραμμά του. Από την παρουσίασή του το 2005 μέχρι σήμερα, έχει προσελκύσει το ενδιαφέρον πολλών ερευνητών και έχουν προταθεί πολλές παραλλαγές και βελτιώσεις του αλγορίθμου. Οι Piotr Dollar, Christian Wojek, Bernt Schiele, Pietro Perona [5] χαρακτηρίζουν τα ιστογράμματα

Προσανατολισμού της Βάθμωσης ως το ισχυρότερο χαρακτηριστικό για εντοπισμό πεζών σε μια εικόνα.

## 2.2 Ανίχνευση Πεζών

Η ανίχνευση πεζών εντάσσεται στο γενικότερο πρόβλημα της ανίχνευσης αντικειμένων και είναι ένα από τα πιο συνηθισμένα και πολύ-μελετημένα προβλήματα στην υπολογιστική όραση. Η διαφορά των πεζών με τα σταθερά αντικείμενα είναι ότι οι πεζοί δεν είναι “άκαμπτοι” (RIGID). Ένας πεζός μπορεί να εμφανιστεί σε μια εικόνα σε πολλές διαφορετικές καταστάσεις. Μια εικόνα μπορεί να περιλαμβάνει ένα πεζό σε οποιοδήποτε από τα στάδια του βηματισμού ή ακόμα και σε στάση, όπου πάλι υπάρχει τεράστια ποικιλία. Ένας αλγόριθμος εντοπισμού πεζών πρέπει να μπορεί εντοπίζει ένα πεζό σε όλες τις προηγούμενες καταστάσεις ακόμα και με μικρές διαφορές στη γωνία λήψης. Χρειάζεται δηλαδή να είναι λίγο πιο “ευέλικτος” από ένα απλό εντοπιστή αντικειμένων.

Η ανίχνευση αντικειμένου δεν περιορίζεται μόνο στο να αποφανθούμε αν υπάρχει αντικείμενο σε μια εικόνα αλλά και να το εντοπίσουμε μέσα σε αυτή. Για να επιτευχθεί η ανίχνευση χρειάζεται να εξάγουμε και να αξιολογήσουμε επιλεγμένα χαρακτηριστικά της εικόνας. Ποια θα είναι αυτά και με ποιο τρόπο θα αξιολογηθούν για να επιτευχθεί το καλύτερο δυνατό αποτέλεσμα, είναι τα κύρια ζητούμενα. Δεν πρέπει βέβαια να υποτιμάται η πολυπλοκότητα χρόνου ενός αλγορίθμου αφού αυτή τον καθιστά εφαρμόσιμο ή μη.

Ο τρόπος που θα αξιολογηθούν τα χαρακτηριστικά που έχουμε εξάγει, δηλαδή η επιλογή του ταξινομητή που θα χρησιμοποιηθεί, και ο τρόπος που θα τροφοδοτηθεί με χαρακτηριστικά (πλήθος, σειρά εισόδου κ.α.) είναι και αυτά μείζονος σημασίας. Αυτός είναι και ο λόγος που σε κάθε αλγόριθμο εικόνας χρησιμοποιείται συγκεκριμένος ταξινομητής.

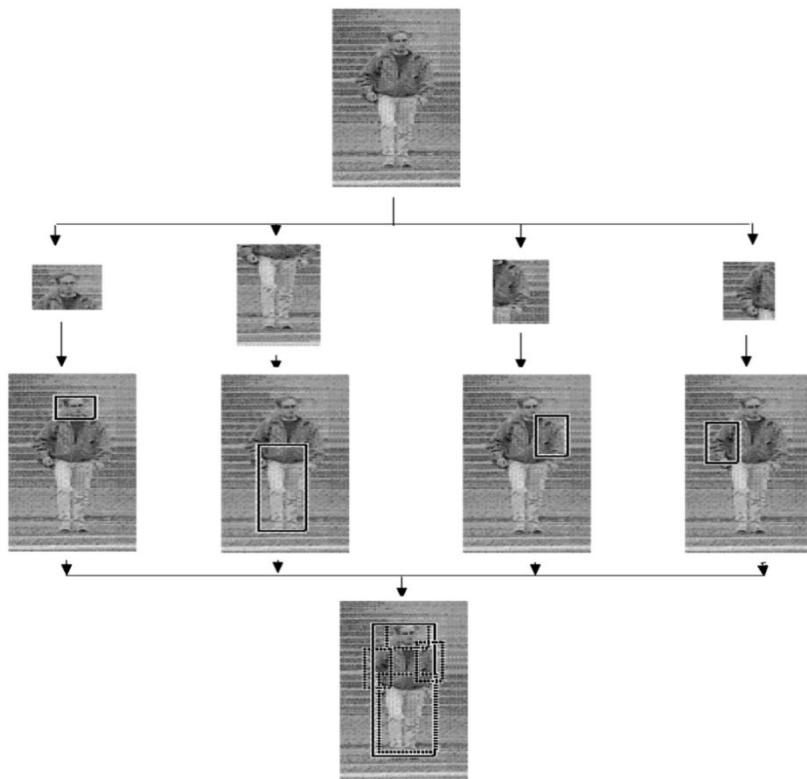
### 2.2.1 Αλγόριθμοι ανίχνευσης αντικειμένων

Έδη από το 1983 υπάρχει δημοσιευμένη εργασία του David Hogg [6], στην οποία επιχειρείται η περιγραφή πεζού με βάση ιεραρχημένα επίπεδα. Κάθε επίπεδο αντιστοιχεί σε κάποιο σημείο του σώματός του και κάνοντας υπέρθεση της εικόνας που δημιουργείται

από τον υπολογιστή στην αρχική εικόνα γίνεται η ταυτοποίηση. Το 1997 οι Michael Orenetal [7] παρουσίασαν μια τεχνική ανίχνευσης βασισμένη στην πρωτότυπη, για την εποχή, ιδέα των χαρακτηριστικών Haar. Σ' αυτή τη μέθοδο, το σχήμα ενός αντικειμένου περιγράφεται χρησιμοποιώντας κυματιδιακές συνιστώσες. Το 2001 οι Viola & Jones παρουσίασαν ένα βελτιωμένο αλγόριθμο [8] βασισμένο στα “χαρακτηριστικά Haar”. Με χρήση πινάκων πέτυχαν πολύ γρήγορο υπολογισμό των χαρακτηριστικών κάνοντας το σύστημά τους να λειτουργεί σε πραγματικό χρόνο. Οι Bastian Leibe, Ales Leonardis, και Bernt Schiele παρουσίασαν το 2004 έναν αλγόριθμο [9], ο οποίος ψάχνει συγκεκριμένες δομές σε ένα αντικείμενο τις οποίες ταυτοποιεί μέσα από ένα Codebook. Οι δομές αυτές εμφανίζονται στο αντικείμενο γύρω από κάποια σημεία ενδιαφέροντος. Συγκρίνοντας τα μοντέλα και συμψηφίζοντας κάποια βάρη, ο αλγόριθμος αποφαίνεται αν μια δομή ανήκει στο προς αναζήτηση αντικείμενο. Σημαντικό πλεονέκτημα του αλγορίθμου αυτού, είναι ότι δεν χρειάζεται να εμφανίζονται όλα τα μέρη του αντικειμένου για τον εντοπισμό του.

## 2.2.2 Κατηγορίες αλγορίθμων ανίχνευσης Πεζών

Οι αλγόριθμοι αναγνώρισης πεζών μπορούν να διαχωριστούν σε δύο κατηγορίες. Στην μια εντοπίζονται ξεχωριστά διάφορα μέρη του ανθρώπινου σώματος, όπως χέρι, πόδια και το κεφάλι και στη συνέχεια εξετάζεται η σχετική θέση τους και το μέγεθός τους από τα οποία και εξάγεται το αποτέλεσμα. Με την τμηματοποίηση της εικόνας ξεπερνιέται εν μέρει το πρόβλημα των διαφορετικών εμφανίσεων του πεζού. Έτσι μπορεί να αντιμετωπιστεί πολύ πιο εύκολα ως στατικό – άκαμπτο αντικείμενο. Κάνοντας όμως πολλές τμηματοποιήσεις και ταξινομήσεις για μια μόνο εικόνα αυξάνεται η υπολογιστική πολυπλοκότητα του αλγορίθμου. Η αύξηση της υπολογιστικής πολυπλοκότητας έρχεται σε αντίθεση με την υλοποίησή του για τον εντοπισμό πεζών σε πραγματικό χρόνο. Στο Σχήμα 2.1 φαίνεται η γενικότερη λειτουργία της ανίχνευσης ανά τμήματα σε μια εικόνα, όπως περιγράφτηκε από τους Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio στο [10].



**Σχήμα 2.1** Αναπαράσταση του τρόπου ανίχνευσης ανά τμήματα του αντικειμένου[10]

Στη δεύτερη κατηγορία, σκοπός είναι η εξαγωγή ενός χαρακτηριστικού διανύσματος με βάση ένα παράθυρο της εικόνας. Το παράθυρο αυτό είναι συγκεκριμένων διαστάσεων και περιγράφει, με κάποιο τρόπο που έχουμε επιλέξει, το εσωτερικό του. Εξετάζοντας την “ομοιότητα” αυτού του διανύσματος με μια ομάδα πρότυπων διανυσμάτων, γίνεται και η αξιολόγησή του. Στη δεύτερη κατηγορία μεθόδων εντάσσεται και η μέθοδος των Ιστογραμμάτων Προσανατολισμού της Βάθμωσης.



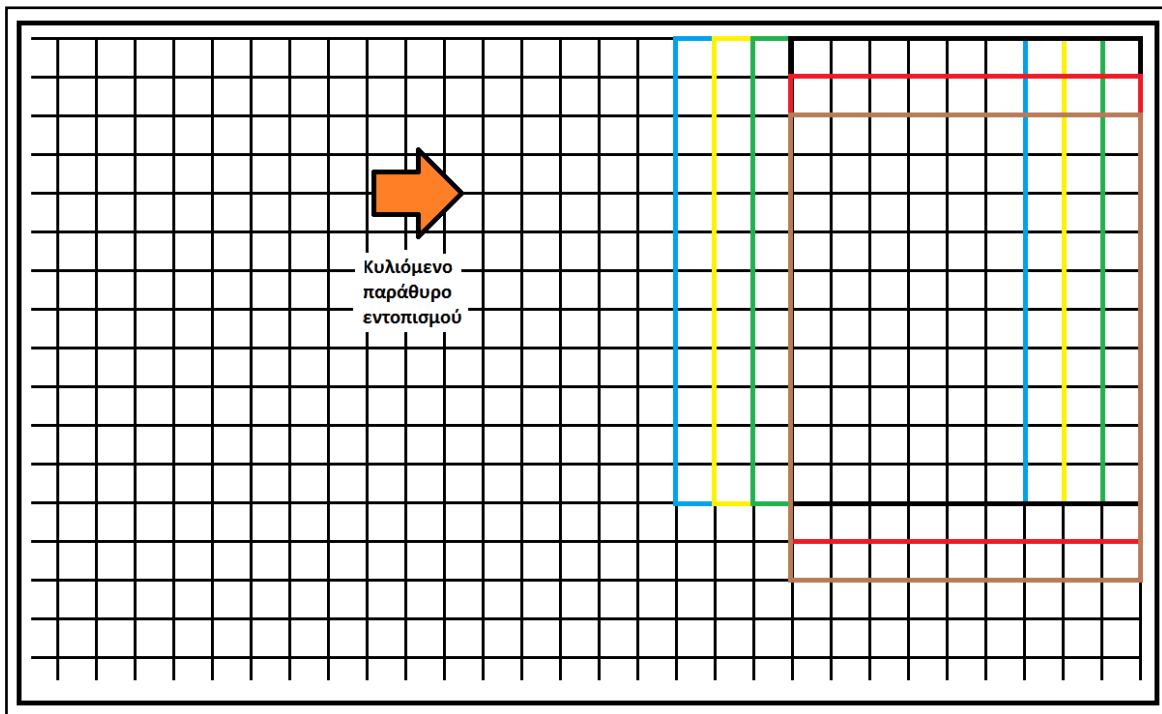
**Σχήμα 2.2** Εξαγωγή Διανύσματος Χαρακτηριστικών

## 2.3 Ιστογράμματα Προσανατολισμού της Βάθμωσης (Histograms of Oriented Gradients)

Η χρήση Ιστογραμμάτων Προσανατολισμού, συναντάται πρώτη φορά το 1995 όπου χρησιμοποιήθηκαν για αναγνώριση χειρονομιών [11]. Στην εργασία που παρουσιάστηκε το 2005 από τους Dalal & Triggs [4], προστέθηκαν πολλά καινούρια στοιχεία και δοκιμάστηκαν σε κάθε στάδιο του αλγορίθμου πολλές εναλλακτικές λύσεις καθώς και διάφορες ποσοτικές παράμετροι. Έτσι οι Dalal & Triggs [4] κατέληξαν σε ένα βέλτιστο μοντέλο με πολύ καλά αποτελέσματα.

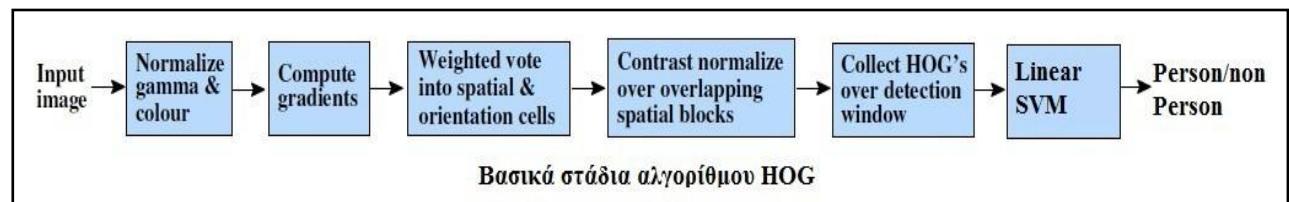
Σκοπός του αλγορίθμου είναι να εξάγει ένα χαρακτηριστικό διάνυσμα το οποίο θα κωδικοποιεί τις κλίσεις και τα μέτρα των εικονοστοιχείων της εικόνας. Αρχικά παράγονται τα διανύσματα από υποσημειωμένες εικόνες στο μέγεθος του παραθύρου εντοπισμού, συγκεκριμένου πλάτους και μήκους. Πρόκειται δηλαδή για εικόνες που είναι γνωστό αν περιέχουν ή όχι άνθρωπο και σκοπός είναι να εκπαιδευτεί με αυτές ο ταξινομητής. Τις εικόνες που είναι μεγαλύτερες από το παράθυρο εντοπισμού που έχουμε επιλέξει, τις εξετάζουμε ανά περιοχές. Μετακυλύουμε δηλαδή το παράθυρο εντοπισμού πάνω στην εξεταζόμενη εικόνα, εξάγουμε το χαρακτηριστικό διάνυσμα σε κάθε περιοχή και ανάλογα γίνεται η ταξινόμηση της κάθε περιοχής. Η περιοχή ενδιαφέροντος μεταφέρεται με συγκεκριμένο βήμα κάθε φορά οριζόντια, μέχρι το άκρο της εικόνας. Όταν φτάσει στο άκρο της εικόνα μετακινούμε το παράθυρο εντοπισμού κάθετα στην εικόνα και πάλι με συγκεκριμένο βήμα.

Στο Σχήμα 2.3 φαίνεται η κύλιση του παραθύρου ανίχνευσης οριζόντια και κάθετα. Έτσι επιτυγχάνεται η σάρωση όλης της εικόνας με αρκετά μεγάλη επικάλυψη (φτάνει και το 93,75% σε κάποιες περιπτώσεις). Η διαδικασία επαναλαμβάνεται και για μικρότερες κλίμακες του παραθύρου εντοπισμού, κάνοντας υπό-δειγματοληψία, με σκοπό την ανεύρεση πεζών με μικρότερες διαστάσεις μέσα στην εικόνα. Πεζοί με μικρότερες διαστάσεις μπορεί να είναι παιδιά ή απλά πεζοί που βρίσκονται στο βάθος της εικόνας.



**Σχήμα 2.3** Σάρωση εικόνας με το Παράθυρο Εντοπισμού

Τα κύρια στάδια του αλγορίθμου διαφαίνονται στον ακόλουθο Σχήμα (2.4) και ακολουθεί η αναλυτική περιγραφή κάθε σταδίου ξεχωριστά.



**Σχήμα 2.4** Βασικά σήματα υπολογισμού Ιστογραμμάτων Προσανατολισμένης Βάθμωσης

### 2.3.1 Γάμμα εξισορρόπηση και κανονικοποίηση χρώματος

Οι Dalal & Triggs [4] δοκίμασαν διαφορετικά χρωματικά συστήματα καταλήγοντας στο συμπέρασμα ότι τα καλύτερα αποτελέσματα λαμβάνονται από το RGB και το LAB ενώ το ασπρόμαυρο μειώνει την απόδοση κατά 1,5 %. Το πρώτο στάδιο είναι ένα φιλτράρισμα της εικόνας για την εξάλειψη διακυμάνσεων στη φωτεινότητα και τον περιορισμό της επίδρασης των σκιάσεων. Εάν γίνει square root γάμμα συμπίεση (gamma compression) τότε η απόδοση παρουσιάζει μια μικρή βελτίωση, η οποία όμως χαρακτηρίζεται ως μέτρια από

τους συγγραφείς. Με δεδομένη την κανονικοποίηση ανά περιοχές που γίνεται στη συνέχεια του αλγορίθμου και επιτυγχάνει περίπου τα ίδια και ίσως λίγο καλύτερα αποτελέσματα, οι επεξεργασίες σε αυτό το στάδιο μπορούν να παραληφθούν.

### 2.3.2 Υπολογισμός Βάθμωσης και Μέτρου

Τα HOGs όπως διαφαίνεται και από το όνομά τους είναι βασισμένα στην Βάθμωση (gradient) της εικόνας στα διάφορα σημεία της. Λέγοντας σημείο ψηφιακής εικόνας θα εννοούμε το εκάστοτε εικονοστοιχείο. Η Βάθμωση σε κάποιο σημείο της εικόνας εκφράζει τις απότομες μεταβολές της έντασης στην φωτεινότητα. Υπολογίζοντας τις Βαθμώσεις και κατανέμοντας τες με τη χρήση ενός ομοιόμορφου πλέγματος επιτυγχάνεται, εν τέλει η περιγραφή του σχήματος που αναζητούμε.

Η Βάθμωση σε μια εικόνα, λοιπόν, δεν είναι τίποτα άλλο από την παράγωγο της εικόνας σε κάποιο συγκεκριμένο σημείο. Εάν αυτό φαντάζει λίγο αυθαίρετο σαν έννοια, ίσως αντικαθιστώντας την λέξη “παράγωγο” με την πρόταση “ρυθμό μεταβολής της έντασης της φωτεινότητας”, να γίνεται λίγο πιο σαφές τι εννοούμε.

Ο υπολογισμός της Βάθμωσης σε ένα σημείο γίνεται με αφαίρεση δύο διαδοχικών στοιχείων ή δύο εικονοστοιχείων που απέχουν συγκεκριμένη απόσταση το ένα από το άλλο. Αυτό πρέπει να γίνεται με το ίδιο, προφανώς, βήμα στην οριζόντια και στην κάθετη κατεύθυνση. Στη ψηφιακή επεξεργασία εικόνας η Βάθμωση υπολογίζεται από τη συνέλιξη του εικονοστοιχείου με μάσκες (masks). Οι συγγραφείς δοκίμασαν μάσκες όπως  $[-1, 0, 1]$ , Sobel 3x3,  $[-1, 1], [1, -8, 0, 8, -1]$  και διάφορες άλλες. Όπως χαρακτηριστικά αναφέρουν στη δημοσίευσή τους “όσο πιο απλό τόσο πιο καλό”, για να καταλήξουν στις μάσκες του Σχήματος 2.5

$$G_x = [-1, 0, 1] \quad G_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

**Σχήμα 2.5** Μάσκες για τον υπολογισμό συντελεστών Βάθμωσης και Μέτρου

Το  $\frac{Gy}{Gx}$  μας δίνει την εφαπτομένη στο επιλεγμένο σημείο και το  $\sqrt{Gy^2 + Gx^2}$  αποτελεί το μέτρο στο σημείο αυτό.

$X_{-1,-1}$	$X_{-1,0}$	$X_{-1,+1}$
$X_{0,-1}$	$X_{0,0}$	$X_{0,+1}$
$X_{+1,-1}$	$X_{+1,0}$	$X_{+1,+1}$

**Πίνακας 2.1** Για να υπολογιστεί ή Βάθμωση σε ένα σημείο πρέπει να ληφθούν και τα γειτονικά του

Από την μάσκα που προτείνεται στην εργασία προκύπτει:

$$Gx = -1 * X_{0,-1} + 0 * X_{0,0} + 1 * X_{0,+1} = -X_{0,-1} + X_{0,+1}$$

$$Gy = -1 * X_{-1,0} + 0 * X_{0,0} + 1 * X_{+1,0} = -X_{-1,0} + X_{+1,0}$$

Αυτό το αναφέρουμε για να γίνει πιο σαφές ότι για τον υπολογισμό της κλίσης και του μέτρου σε κάποιο σημείο σημασία έχουν τα γειτονικά του εικονοστοιχεία.

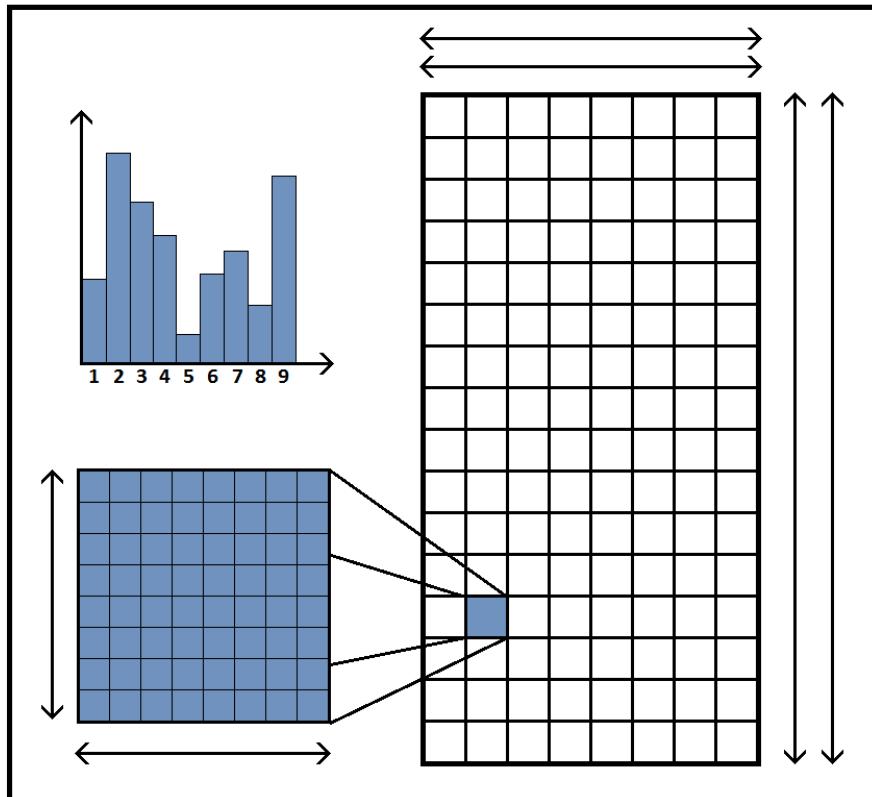
Τέλος να επισημάνουμε ότι για έγχρωμες εικόνες υπολογίζουμε τις κλίσεις κάθε χρωματικής συνιστώσας ξεχωριστά και επιλέγεται αυτή με τη μεγαλύτερη νόρμα.

### 2.3.3 Δημιουργία Ιστογραμμάτων Προσανατολισμού της Βάθμωσης

Μετά τον υπολογισμό των Βαθμώσεων και των μέτρων, δημιουργούμε ιστογράμματα κατατάσσοντας τα εικονοστοιχεία ανάλογα με την Βάθμωσή τους. Η κατηγοριοποίηση των Βαθμώσεων, μπορεί να γίνει με διαφορετικούς τρόπους. Οι διαφορές τους είναι στο πλήθος των bins (γωνιακή ομάδα) και στο εύρος του τριγωνομετρικού κύκλου που θα χρησιμοποιηθεί.

Το κάθε ιστόγραμμα δημιουργείται από τα μέτρα των εικονοστοιχείων μιας περιοχής που λέγεται cell. Η μέγιστη απόδοση του αλγορίθμου σύμφωνα με τους Dalal & Triggs [4],

προκύπτει για cell μεγέθους  $8 \times 8$  εικονοστοιχείων (Σχήμα 2.6). Κάθε περιοχή λοιπόν θα δημιουργεί ένα ιστόγραμμα με την εξής ιδιαιτερότητα: η συνεισφορά κάθε εικονοστοιχείου δεν είναι μία μονάδα στην θέση του bin που ανήκει, αλλά το μέτρο του. Για παράδειγμα αν ένα εικονοστοιχείο έχει μέτρο 3 και ανήκει στο bin 2, τότε στη θέση 2 του ιστογράμματος θα προστίθεται το 3.

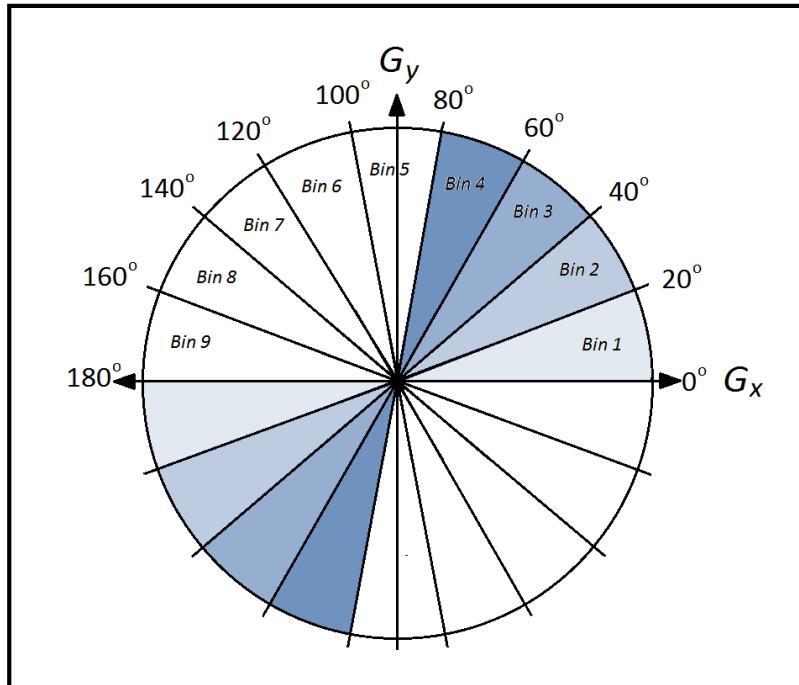


Σχήμα 2.6 Παράθυρο εντοπισμού, cell, ιστόγραμμα

### Επιλογή των bins

Όπως σημειώνεται στην εργασία ο αριθμός των bins επηρεάζει την απόδοση του περιγραφέα, η οποία αυξάνει μέχρι και τον αριθμό των εννέα bins. Από εκεί και πάνω η απόδοση δεν παρουσιάζει αξιοσημείωτες μεταβολές. Αυτό, όμως το αποδίδουν για συνολικό εύρος  $0 - 180^\circ$ . Ειδικά για ανθρώπους που παρουσιάζουν μια μεγάλη ανομοιομορφία λόγω ρούχων, η απόδοση μειώνεται όταν χρησιμοποιούνται bin και στα άλλα δυο τεταρτημόρια ( $180 - 360^\circ$ ). Όταν βρεθεί κλήση που ανήκει σε γωνία από  $180^\circ - 360^\circ$  τότε εντάσσεται στο bin της προσκείμενής της γωνίας. Στο Σχήμα 2.7 αποτυπώνονται

οπτικά τα βέλτιστα bin για τον εντοπισμό πεζών, εννέα τον αριθμό και είκοσι μοίρες το κάθε ένα.



**Σχήμα 2.7** Αντιστοιχία γωνιών - bin

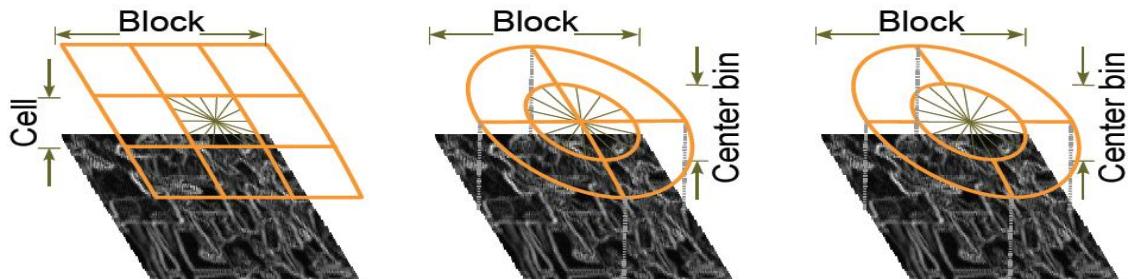
### 2.3.4 Κανονικοποίηση σε επικαλυπτόμενα block

Σε κάποιες “γειτονιές” εικονοστοιχείων παρουσιάζονται μεγάλες μεταβολές των επιπέδων φωτεινότητας χωρίς όμως να πρόκειται για διαφορετικό αντικείμενο της εικόνας. Τις περισσότερες φορές οφείλεται σε εναλλαγές του φωτισμού και σκιάσεις. Αυτό μπορεί να συμβεί είτε επειδή έχει παραληφθεί το πρώτο φιλτράρισμα είτε γιατί ενώ έγινε, οι διακυμάνσεις της έντασης ή της χρωματικότητας ήταν τέτοιες σε μέγεθος που δεν υπήρξε το επιθυμητό αποτέλεσμα.

Για να αντιμετωπίσουν αυτό το φαινόμενο, οι Dalal & Triggs [4], δοκίμασαν μια τοπική κανονικοποίηση των ιστογραμμάτων με επικαλυπτόμενες περιοχές. Επικαλυπτόμενες περιοχές σημαίνει ότι κάθε cell, συνεισφέρει στο τελικό αποτέλεσμα περισσότερες από μια φορές. Η επιλογή αυτή αποδείχτηκε, εν τέλει ουσιώδης για την καλύτερη απόδοση του περιγραφέα. Η κανονικοποίηση με επικάλυψη γίνεται σε μια περιοχή αποτελούμενη από

μια ομάδα cells, που ονομάζεται block. Είναι άξιο αναφοράς ότι η κανονικοποίηση με επικάλυψη μπορεί να αυξάνει την απόδοση, αυξάνει όμως αντίστοιχα και το μέγεθος του χαρακτηριστικού διανύσματος ανάλογα με το μέγεθος του block. Για παράδειγμα για block 2x2 το μέγεθος του χαρακτηριστικού διανύσματος γίνεται τέσσερις φορές μεγαλύτερο από το χαρακτηριστικό διάνυσμα χωρίς την επικάλυψη. Η ευρύτερη αυτή περιοχή που γίνεται η κανονικοποίηση αποτέλεσε αντικείμενο αξιολόγησης στην εργασία των Dalal & Triggs [4] ως προς τη γεωμετρία της αλλά και ως προς τα διαφορετικά χαρακτηριστικά της κάθε γεωμετρίας.

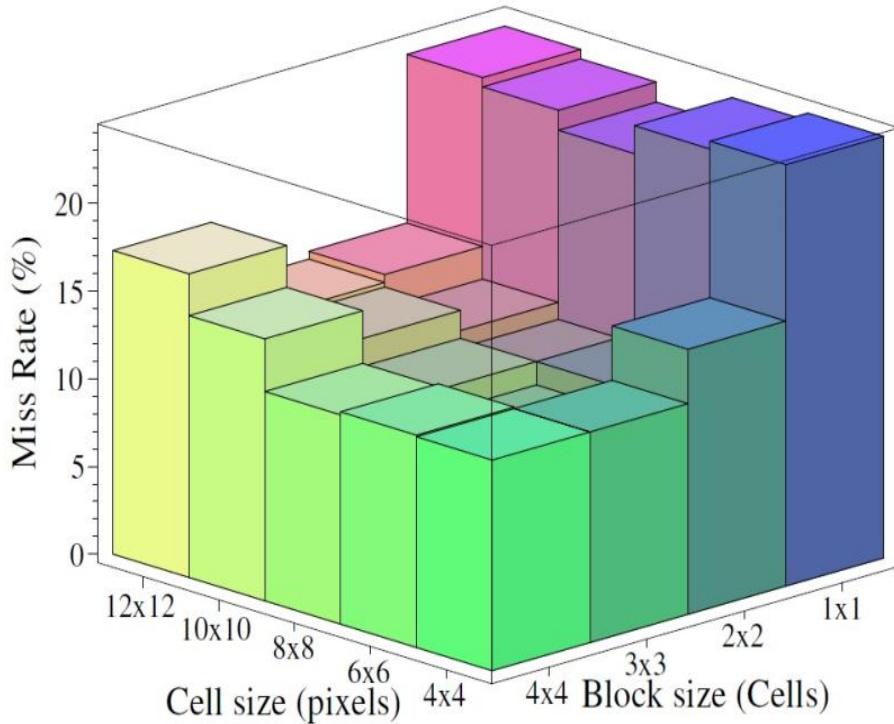
Οι δύο διαφορετικές γεωμετρίες που δοκιμάστηκαν είναι το παραλληλόγραμμο και ο κύκλος που αναφέρονται ως περιγραφέις R-HOG και C-HOG αντίστοιχα. Ο C-HOG μελετήθηκε σε δύο εκδοχές. Στο Σχήμα 2.8 φαίνονται ο R-HOG και οι δύο εκδοχές του C-HOG.



**Σχήμα 2.8** Αριστερά: R – HOG block, Δεξιά: Τα δύο είδη C – HOG

(Εικόνα από την διδακτορική διατριβή του Nanveed Dalal, σελ.32)

Ο R-HOG έχει τρεις παραμέτρους: το πλήθος των bins, τα pixels ανά cell και τα cells ανά block. Από τη μελέτη τους προέκυψε το Σχήμα 2.9, στην οποία φαίνονται οι επιδόσεις του περιγραφέα για διάφορους συνδυασμούς στα μεγέθη των cells και των blocks.



**Σχήμα 2.9** Απόδοση του αλγορίθμου για διάφορα μεγέθη των cells και blocks

(Εικόνα από την εργασία που παρουσίασαν οι Dalal & Triggs, σελ. 41)

Ο C-HOG μπορεί να έχει για κεντρικό cell ένα ενιαίο κυκλικό ή ένα κεντρικό cell που χωρίζεται σε κυκλικούς δίσκους. Η μελέτη των συγγραφέων, έδειξε ότι δεν υπάρχει αξιοσημείωτη διαφορά στη απόδοση του ταξινομητή για τα δύο cell. Εδώ οι παράμετροι είναι τέσσερις: ο αριθμός των bins των κλίσεων, ο αριθμός των bins των ακτινών, το μήκος της ακτίνας του κεντρικού τομέα που μετριέται σε εικονοστοιχεία, και ο παράγοντας επέκτασης των ακτινών.

Για την κανονικοποίηση των τιμών των εικονοστοιχείων έγιναν τέσσερις δοκιμές με:

- L2 νόρμα
- L2-Hys, L2-νόρμα ακολουθούμενη από clipping (περιορίζοντας τη μέγιστη τιμή των εικονοστοιχείων σε 0.2) και επανακανονικοποιώντας
- L1-νόρμα
- L1 ρίζα

Το αποτέλεσμα έδειξε ότι οι τρεις κανονικοποιήσεις έχουν τα ίδια αποτελέσματα ενώ η L1 νόρμα μειώνει αισθητά την απόδοση.

Τέλος δοκιμάστηκε ο αλγόριθμος χωρίς κάποια κανονικοποίηση και το αποτέλεσμα ήταν απόδοση πολύ μειωμένη ακόμα και σε σχέση με την L1 νόρμα.



## Κεφάλαιο 3

# Υλοποίηση του περιγραφέα Ιστογραμμάτων Προσανατολισμού της Βάθμωσης στο Matlab

### 3.1 Εισαγωγή

Το στάδιο της υλοποίησης ενός αλγορίθμου σε FPGA έπειτα του σταδίου προτυποποίησης αυτού. Κατά το στάδιο της προτυποποίησης γίνεται εξομοίωση του αλγορίθμου σε κάποιο μαθηματικό πρόγραμμα με σκοπό την λειτουργική του επιβεβαίωση. Τα πιο γνωστά προγράμματα που μπορούμε να κάνουμε κάτι τέτοιο είναι το Octave και το Matlab που και το καθένα αποτελεί μια interpreted γλώσσα προγραμματισμού. Οι interpreted γλώσσες δίνουν τη δυνατότητα στο χρήστη να αναπτύξει πολύ γρήγορα κάποιο πρόγραμμα που σκοπό έχει την λειτουργική επιβεβαίωση του αλγορίθμου και όχι την αποδοτική και τάχιστη εκτέλεσή του. Απαιτεί δηλαδή, πολύ μικρότερο χρόνο ανάπτυξης και σχετικά ευκολότερη υλοποίηση αλλά ο χρόνος εκτέλεσης του προγράμματος είναι κατά πολύ μεγαλύτερος. Με την εξομοίωση του αλγορίθμου δίνεται η δυνατότητα να προσεγγιστούν με πολύ μεγάλη ακρίβεια τα αποτελέσματα που θα προκύψουν από την υλοποίηση. Λόγω της ευελιξίας των προγραμμάτων αυτών υπάρχει η δυνατότητα να γίνουν πειραματισμοί με διαφορετικές εκδοχές του αλγορίθμου και να ληφθούν αποτελέσματα για διαφορετικές παραμέτρους. Ένα ακόμη πλεονέκτημα των εξομοιώσεων, είναι η δυνατότητα να γίνεται γνωστό το αποτέλεσμα σε όλα τα στάδια της υλοποίησης του FPGA. Δυστυχώς με την εξομοίωση δεν αποτυπώνεται η ροή των δεδομένων οπότε δεν υπάρχει ταύτιση αποτελεσμάτων σε σχέση

με το χρόνο. Στην εργασία αυτή χρησιμοποιήσαμε το Matlab και στο κεφάλαιο αυτό περιγράφονται τα στάδια και οι δοκιμές που ακολουθήσαμε για την υλοποίηση του περιγραφέα. Περιγράφονται επίσης οι απλοποιήσεις που κάναμε για να απλουστεύσουμε το κύκλωμα και να μειώσουμε το υπολογιστικό κόστος, καθώς και τα αποτελέσματα των αξιολογήσεων αυτών.

## 3.2 To Matlab

Το Matlab αποτελεί ίσως το πιο διαδεδομένο μαθηματικό εργαλείο. Είναι ουσιαστικά ένα περιβάλλον προγραμματισμού, γλώσσας υψηλού επιπέδου, κατάλληλο για ανάλυση και οπτικοποίηση των δεδομένων. Με το Matlab μπορεί κάποιος να αναπτύξει αλγορίθμους, να δημιουργήσει μαθηματικά μοντέλα, να κάνει προσομοιώσεις ακόμα και να δημιουργήσει εφαρμογές με γραφικό περιβάλλον.

Για να υπάρξει μια όσο το δυνατόν καλύτερη προσέγγιση του κυκλώματος με το Matlab, προτιμήθηκε κώδικας που θα ήταν όσο το δυνατόν πιο αντιπροσωπευτικός. Έτσι αποφεύχθηκε η χρήση έτοιμων συναρτήσεων από τις βιβλιοθήκες του προγράμματος. Η συνάρτηση τετραγωνικής ρίζας που χρησιμοποιήθηκε για μια πρώτη αξιολόγηση, παραλήφθηκε στην τελική εκδοχή του προγράμματος.

Πιο συγκεκριμένα, στο περιβάλλον του Matlab υλοποιήθηκε μια τροποποιημένη μορφή από αυτή του τελικού αλγορίθμου. Αυτό έγινε για να επιβεβαιωθούν τα αποτελέσματα αλλά και για να δημιουργηθεί ένα μοντέλο το οποίο αργότερα θα μπορούσε να παραμετροποιηθεί. Αρχικά για να γίνει πλήρως κατανοητός ο αλγόριθμος και ο τρόπος λειτουργίας του, έγινε μια σχεδόν πλήρης υλοποίηση όπως την προτείνουν οι Dalal & Triggs [4]. Χρησιμοποιήθηκαν οι βέλτιστες παράμετροι όπως τις αναφέρουν στην εργασία τους παρόλο που ήταν γνωστό εξ αρχής ότι κάποια κομμάτια του κώδικα είτε θα απαλείφονταν, είτε θα είχαν άλλη μορφή στο τέλος. Από την αρχική υλοποίηση εξήχθησαν αποτελέσματα και έγιναν συγκρίσεις με την υλοποίηση του αλγορίθμου που υπάρχει στη βιβλιοθήκη VL\_feat. Η VL\_feat θεωρείται από τις εγκυρότερες και πληρέστερες υλοποιήσεις του Περιγραφέα Προσανατολισμένων Βαθμώσεων, οπότε χρησιμοποιήθηκε ως “σημείο αναφοράς” καλής λειτουργίας για την δική μας. Επίσης, με τη VL\_Feat μας δόθηκε η

δυνατότητα να “οπτικοποιήσουμε” σε κάποια στάδια, τα αποτελέσματά που είχαμε λάβει μέχρι εκείνη τη χρονική στιγμή.

Στην παρούσα εργασία, ακολουθήθηκαν τα βήματα του αλγορίθμου όπως περιγράφονται στο κεφάλαιο 2. Εκτός των αρχικών περιορισμών έπρεπε να συμπεριληφθούν και πειραματισμοί με επιπλέον περιορισμούς που προέκυψαν κατά τη διάρκεια της υλοποίησης του Hardware.

Μια ουσιώδης λεπτομέρεια που πρέπει να καταστήσουμε σαφή πριν ξεκινήσουμε, γιατί θα τη συναντούμε πολύ συχνά, είναι ο τρόπος ροής των δεδομένων. Σε ένα σύστημα πραγματικού χρόνου τα δεδομένα ρέουν από μια κύρια είσοδο στο σύστημα και περνούν στα διάφορα υποκυκλώματα. Στην προσομοίωση, αντί να “κινούνται” τα δεδομένα κινούνται τα “εργαλεία” επεξεργασίας. Στο Matlab τα δεδομένα είναι τοποθετημένα σε πίνακες και η επεξεργασία γίνεται προσπελάζοντας τους πίνακες με επαναληπτικές δομές. Τα αποτελέσματα τοποθετούνται σε νέους πίνακες, αρχικά για να εξεταστούν ως προς την εγκυρότητά τους και αν τελικά είναι σωστά, χρησιμοποιηθούν στο επόμενο στάδιο.

### **3.2.1 Γάμμα εξισορρόπηση και κανονικοποίηση χρώματος**

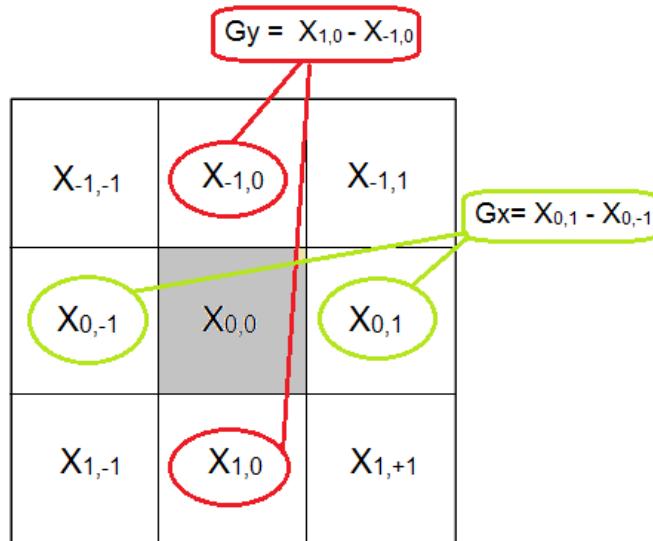
Την επεξεργασία της φωτεινότητας και του συντελεστή Γάμμα θα τα αποφύγουμε τελείως, όχι μόνο γιατί ξέρουμε εξ αρχής ότι η υλοποίηση τους είναι αρκετά σύνθετη αλλά κυρίως γιατί σύμφωνα με τους συγγραφείς, η συμβολή τους στην τελική απόδοση είναι αμελητέα. Αυτό που πρέπει να πούμε σ' αυτό το σημείο είναι ότι τις εικόνες πριν από την επεξεργασία τις μετατρέψαμε σε ασπρόμαυρες.

### **3.2.2 Εξαγωγή Βαθμώσεων και Μέτρων**

Το πρώτο βήμα του αλγορίθμου μας είναι η εξαγωγή των Βαθμώσεων. Η Βάθμωση υπολογίζεται και ως προς τον οριζόντιο και ως προς τον κάθετο άξονα. Ακολουθώντας τα μαθηματικά πρότυπα, από εδώ και στο εξής, τη Βάθμωση στον οριζόντιο άξονα θα την

αναφέρουμε ως  $Gx$  και στον κάθετο ως  $Gy$ , από τη λέξη Gradient και από τα γράμματα που συμβολίζονται οι άξονες στη Μαθηματική βιβλιογραφία.

Η Βάθμωση λοιπόν σε ένα σημείο προσεγγίζεται με τη διαφορά των γειτονικών του εικονοστοιχείων. Το  $Gx$  είναι η διαφορά του προηγούμενου από το επόμενο και το  $Gy$  είναι η διαφορά του από πάνω από το από κάτω.



**Σχήμα 3.1** Τα  $Gx$  και  $Gy$  ενός εικονοστοιχείου

Για τον υπολογισμό της Βάθμωσης έχουν προταθεί πολλές διαφορετικές μάσκες. Στην παρούσα εργασία χρησιμοποιήθηκε η μάσκα  $[-1 \ 0 \ 1]$  και αυτό γίνεται με την πράξη της συνέλιξης που στο Matlab είναι η εντολή *Sum (a.\*b)*, όπου -a- η εικόνα και -b- η μάσκα μας. Όπως προαναφέραμε όμως, θέλαμε να αποφύγουμε την χρήση συναρτήσεων οπότε και με μια επαναληπτική δομή *for* προσπελάσαμε τα στοιχεία ανά γραμμές και ένα προς ένα, και δημιουργήσαμε δύο νέους πίνακες, τον  $Gx$  και  $Gy$ . Εάν συμβολίσουμε το πλήθος των γραμμών με *lines* και των στηλών με *cols*, τότε η μαθηματική έκφραση της οριζόντιας και κάθετης Βάθμωσης για κάθε στοιχείο δίνεται από τους αντίστοιχους τύπους:

- $Gx_{i,j} = X_{i,j+1} - X_{i,j-1}$  όπου  $0 < i < \text{line} + 1$
- $Gy_{i,j} = X_{i+1,j} - X_{i-1,j}$  όπου  $0 < j < \text{cols} + 1$

Για τα ακραία εικονοστοιχεία υπάρχουν διάφορες τεχνικές, όπως να αγνοηθούν ή να επεκταθεί ο πίνακας με διάφορες τιμές. Εδώ επιλέχθηκε η προέκταση του πίνακα αντιγράφοντας την κάθε ακριανή γραμμή ή στήλη.

Αφού υπολογιστούν τα  $G_x$  και  $G_y$  για κάθε εικονοστοιχείο μπορεί πλέον να υπολογιστεί το μέτρο σε αυτά. Ο υπολογισμός του μέτρου προκύπτει από τον τύπο:

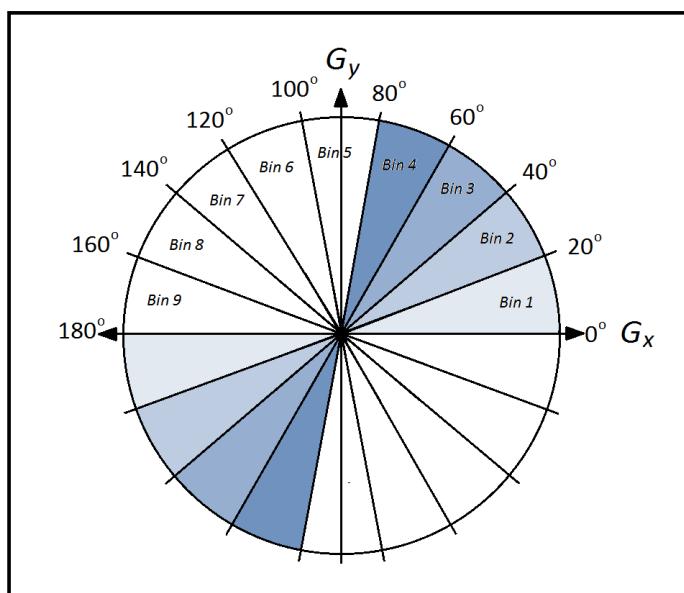
- $M_{i,j} = \sqrt{G_{xi,j}^2 + G_{yi,j}^2}$

Η γωνία κλίση (προσανατολισμός) δίνεται από τον υπολογισμό του τόξου της εφαπτομένης της κλίσης στο σημείο:

- $\text{Angle}_{i,j} = \text{Arctan} \frac{G_{yi,j}}{G_{xi,j}}$

### 3.2.3 Υπολογισμός του Bin

Όταν έχουμε υπολογίσει και τη γωνία, είμαστε σε θέση να κατηγοριοποιήσουμε το κάθε εικονοστοιχείο σε ένα από τα bins, με την αντιστοιχία που περιγράψαμε στο προηγούμενο κεφάλαιο. Για ευκολία του αναγνώστη, παραθέτουμε ξανά το σχήμα



**Σχήμα 3.2** Αντιστοιχία γωνιών – bin

Ο υπολογισμός των τόξων των εφαπτομένων σε κύκλωμα είναι εξαιρετικά πολύπλοκη υλοποίηση κυκλωματικά με σχετικά μεγάλη χρονική καθυστέρηση αφού γίνεται χρήση πινάκων αντιστοίχησης (Look-Up Tables). Για να απλοποιήσουμε λοιπόν τον υπολογισμό του κάθε bin κάναμε την ακόλουθη διαδικασία, με αποτέλεσμα να έχουμε πράξεις με μικρή κυκλωματική πολυπλοκότητα. Έχει ήδη αναφερθεί από το πρώτο κεφάλαιο ότι ο πολλαπλασιασμός υλοποιείται πολύ εύκολα με τους ενσωματωμένους πολλαπλασιαστές. Κατά συνέπεια, όταν είναι εφικτό πρέπει να προτιμάται η αντικατάσταση των διαιρέσεων από πολλαπλασιασμούς.

Αρχικά υπολογίσαμε, τις τιμές των εφαπτομένων στα όρια του κάθε bin και προέκυψε ο παρακάτω πίνακας.

μοίρες		εφαπτομένη
0	=	0
20	=	0.364
40	=	0.8391
60	=	1.7321
80	=	5.6713
100	=	-1.7321
120	=	-0.8391
140	=	-1.7321
160	=	-5.6713

Πίνακας 3.1 Τιμές εφαπτομένων στα όρια των Bins

Συγκρίνοντας λοιπόν την εκάστοτε τιμή  $\tan_{i,j} = \frac{G_{y,i,j}}{G_{x,i,j}}$ , με τα παραπάνω όρια βρίσκουμε σε ποιο bin ανήκει κάθε τιμή. Με μια διαίρεση διαδοχικές συγκρίσεις προκύπτει το bin. Ο κώδικας που υπολογίζει το bin κάθε εικονοστοιχείου είναι μια αλληλουχία εντολών ελέγχου if. Εάν δηλαδή, η γωνία που υπολογίσαμε είναι μεταξύ  $0 - 20^{\circ}$ , τότε αντιστοιχεί στο bin1, εάν είναι μεταξύ  $21 - 40^{\circ}$  τότε αντιστοιχεί στο bin2 και αντίστοιχα για τα υπόλοιπα.

Av	$\frac{Gy_{i,j}}{Gx_{i,j}} < 0.364$	$=> bin1$
Αλλιώς Av	$0.364 < \frac{Gy_{i,j}}{Gx_{i,j}} < 0.8391 => bin2$ κ.ο.κ	
Τέλος Av		

#### Ψευδοκάδικας υπολογισμού binμε χρήση διαιρεσης

Οι πράξεις απλοποιούνται περισσότερο αν εκμεταλλευτούμε τη συμμετρία που παρουσιάζεται ( $\tan(x) = -\tan(\pi-x)$ ). Λαμβάνοντας υπόψη το πρόσημο και την απόλυτη τιμή έχουμε ακόμα λιγότερες πράξεις και συνεπώς θα έχουμε μικρότερο και ταχύτερο κύκλωμα στην υλοποίηση. Για ακόμα καλύτερη απόδοση στο κύκλωμα μπορούμε να αντικαταστήσουμε την διαιρεση με πολλαπλασιασμό [2]. Η τελική μορφή που πήρε ο κώδικας για τον υπολογισμό των binsείναι:

Αν $Gx_{i,j}$ και $Gy_{i,j}$ ομόσημα (γωνία στο πρώτο ή το τρίτο τεταρτημόριο)		
Av	$Gy_{i,j} < 0.364 * Gx_{i,j}$	$=> bin1$
Αλλιώς Av	$0.364 * Gx_{i,j} < Gy_{i,j} < 0.8391 * Gx_{i,j}$	$=> bin2$ κ.ο.κ
Τέλος Av		
Αλλιώς Av $Gx_{i,j}$ και $Gy_{i,j}$ ετερόσημα (γωνία στο δεύτερο ή το τέταρτο τεταρτημόριο)		
Av	$Gy_{i,j} < 0.364 * Gx_{i,j}$	$=> bin9$
Αλλιώς Av	$0.364 * Gx_{i,j} < Gy_{i,j} < 0.8391 * Gx_{i,j}$	$=> bin8$ κ.ο.κ
Τέλος Av		
Τέλος Av		

#### Ψευδοκάδικας υπολογισμού binμε χρήση πολλαπλασιασμού

Ένα ζήτημα που μπορεί να προκύψει στον υπολογισμό των γωνιών είναι η διαιρεση με το 0, στις περιπτώσεις που  $Gx = 0$ . Αυτό δεν διευκρινίζεται στην εργασία των Dalal & Triggs [4], αλλά εμείς το ερμηνεύσαμε ως εξής:

- Εάν το  $Gx$  είναι 0 και το  $Gy$  είναι διάφορο του 0, τότε έχουμε μεγάλη μεταβολή στον κάθετο άξονα και καθόλου στον οριζόντιο οπότε κατατάσσουμε το εικονοστοιχείο στο bin5.

- Στην περίπτωση που  $Gx = Gy = 0$ , θεωρούμε ότι δεν υπάρχει “πληροφορία” και το κατατάσσουμε σε ένα bin10. Το bin10 δεν αντιστοιχεί σε κάποιο bin, όπως φαίνεται και στο σχήμα, είναι όμως ένας δείκτης για τα επόμενα στάδια.

### 3.2.4 Δημιουργία Ιστογραμμάτων

Αφού έχουμε υπολογίσει το binπου αντιστοιχεί κάθε εικονοστοιχείο είμαστε σε θέση να εξάγουμε τα ιστογράμματα ανά cell. Θα υπάρχουν οχτώ cell οριζόντια και δεκαέξι κάθετα. Θυμίζουμε ότι το cellπου επιλέχτηκε έχει διαστάσεις (8x8) και τα ιστογράμματα έχουν την ιδιαιτερότητα ότι, για κάθε bin που εμφανίζεται, δεν προστίθεται μια μονάδα στην αντίστοιχη θέση, αλλά το μέτρο του εκάστοτε εικονοστοιχείου. Αυτό σημαίνει ότι για κάθε bin ανατρέχουμε στην αντίστοιχη θέση του πίνακα των Μέτρων. Η βαρύνουσα συνεισφορά των κλίσεων με βάση τα μέτρα τους, είναι ένας τρόπος να συμπεριληφθεί το μέτρο στην κωδικοποίηση της εικόνας.

Για να μπορέσουμε να προγραμματίσουμε μια επαναληπτική δομή που θα κατατάσσει σωστά τα bin ανά cell, ίσως είναι χρήσιμο να προσεγγίσουμε πρακτικά το που ανήκει κάθε bin ανάλογα με τη θέση του. Έτσι για το πρώτο cell θέλουμε τα οχτώ πρώτα στοιχεία των οκτώ πρώτων γραμμών. Για το δεύτερο cellθέλουμε τη δεύτερη οχτάδα εικονοστοιχείων, των οχτώ πρώτων γραμμών. Η φόρμουλα που προκύπτει εν τέλει, μας δίνει τους συντελεστές. Αυτοί είναι η εξής: ο δείκτης γραμμών του κάθε cellείναι το ακέραιο μέρος της διαίρεσης της γραμμής με την τρέχουσα θέση αυξημένη κατά ένα. Ομοίως και για τις στήλες.

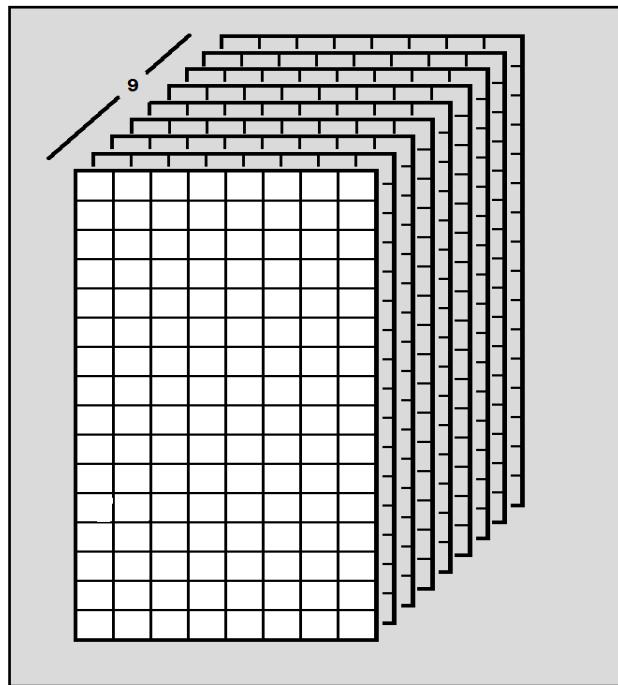
Στο Matlab η εντολή floor στρογγυλοποιεί στον πλησιέστερο μικρότερο ακέραιο. Έτσι προκύπτει:

$$\text{Μπλοκ}_\text{γραμμή} = \text{floor} ((i-1)/8) + 1$$

$$\text{Μπλοκ}_\text{στήλη} = \text{floor} ((j-1)/8) + 1$$

όπου  $i = \text{γραμμή}$  και  $j = \text{στήλη του bin}$

Η τιμή του binμας δείχνει σε ποια θέση μέσα στον πίνακα που περιέχει το ιστόγραμμα θα προσθέσουμε το αντίστοιχο Μέτρο. Το αποτέλεσμα αυτής της διαδικασία είναι ένα τρισδιάστατος πίνακας με ύψος 16, πλάτος 8 και ιστογράμματα των bins 9. Κάθε επίπεδο του πίνακα στο βάθος αντιστοιχεί στη θέση του ιστογράμματος. Για παράδειγμα το πρώτο επίπεδο έχει τις τιμές όλων των cells για το bin1 και κατά αντιστοιχία τα υπόλοιπα. Το επόμενο σχήμα είναι απεικόνιση του τρισδιάστατου πίνακα των ιστογραμμάτων.



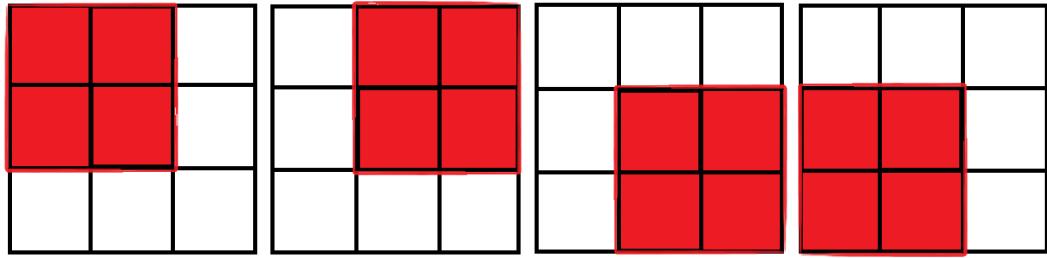
**Σχήμα 3.3** Τρισδιάστατος πίνακας ιστογραμμάτων

### 3.2.5 Κανονικοποίηση σε επικαλυπτόμενα blocks

Τελευταίο βήμα, για την δημιουργία του περιγραφέα είναι μια κανονικοποίηση των cells σε τοπικό επίπεδο, δηλαδή σε blocks. Το κάθε block αποτελείται από 4 cells. Αυτό πιο απλά σημαίνει, ότι γίνεται κανονικοποίηση όχι σε όλη την εικόνα αλλά ανά 4 cells τα οποία επικαλύπτονται. Για κάθε μια ομάδα από cells, όπως φαίνεται στην ακόλουθη εικόνα, γίνεται κανονικοποίηση με την L2- Norm που προκύπτει από την αντίστοιχη τετράδα.

$$\text{L2 - norm} \rightarrow v2 = \frac{v1}{\sqrt{\|v\|^2 + e^2}}, \text{ όπου}$$

$v2$  = νέα τιμή εικονοστοιχείου,  
 $v1$  = αρχική τιμή εικονοστοιχείου,  
 $\|v\|$  = Ευκλείδεια νόρμα  
 $e$  = πολύ μικρή τιμή για να μη προκύπτει  
 διαίρεση με 0



**Σχήμα 3.4** Ομάδες κανονικοποίησης των Cells

Το αποτέλεσμα που προκύπτει, είναι ένας πίνακας  $8 \times 16 \times 36$ . Για κάθε cell δηλαδή παίρνουμε  $4 \times 9$  τιμές, που είναι τα κανονικοποιημένα ιστογράμματα για κάθε block που συμμετέχει το cell.

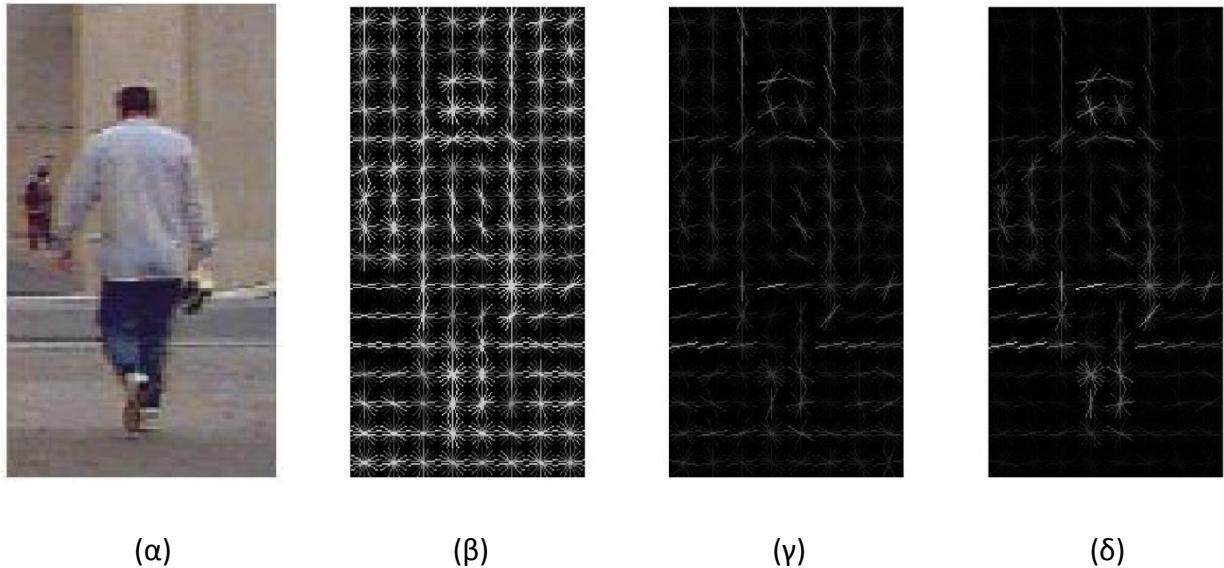
Σ' αυτό το σημείο μπορούμε να κάνουμε μια πρώτη οπτική αξιολόγηση του διανύσματος που έχουμε εξάγει, συγκρίνοντάς το με το αντίστοιχο της βιβλιοθήκης VL\_Feat. Αυτή η δυνατότητα μας παρέχεται από την βιβλιοθήκη συντάσσοντας:

```
imhog = vl_hog('render', hog, 'verbose', 'variant', 'dalaltriggs')
όπου "hog" είναι ο πίνακας που έχουμε εξάγει πριν τον μετατρέψουμε στο χαρακτηριστικό διάνυσμα.
```

Η συνάρτηση τυπώνει τις κλίσεις των διανυσμάτων πάνω σε ένα σύστημα αξόνων, ξεχωριστά για κάθε cell. Σε όποια σημεία της αρχικής εικόνας υπάρχει έντονη αλλαγή στη φωτεινότητα ή τη χρωματικότητα, αναμένεται συσσώρευση διανυσμάτων με συγκεκριμένο προσανατολισμό. Στα υπόλοιπα σημεία είναι λογικό να υπάρχει μια πιο ομαλή κατανομή των διανυσμάτων. Αυτό επιβεβαιώνεται στην οπτική αναπαράσταση των κλίσεων (Σχήμα 3.5), αφού διαφαίνεται το περίγραμμα του ανθρώπου και οι αλλαγές στο οδόστρωμα.

Για πειραματικούς λόγους έγινε και οπτική παρατήρηση του πίνακα που προκύπτει πριν από την κανονικοποίηση σε επικαλυπτόμενα blocks. Για να γίνει αυτό, εφόσον η είσοδος

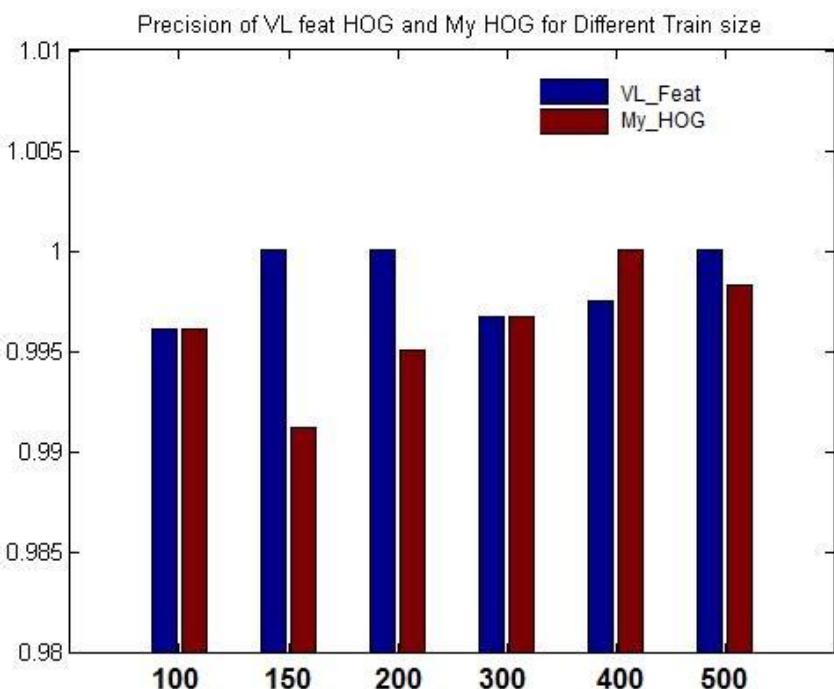
της συνάρτησης πρέπει να έχει διαστάσεις ( $8 \times 16 \times 36$ ) και ο πίνακας μέχρι εκείνη τη στιγμή είναι ( $8 \times 16 \times 9$ ), τον παραθέσαμε τέσσερις φορές ώστε να πετύχουμε το κατάλληλο μέγεθος στην τρίτη διάσταση χωρίς να επηρεάσουμε την κατανομή των προσανατολισμένων διανυσμάτων. Το αποτέλεσμα φαίνεται στο παρακάτω σχήμα. Σε δύο στοιχεία θα πρέπει να σταθούμε παρατηρώντας την παρακάτω εικόνα. Το πρώτο είναι η φωτεινότητα των προσανατολισμένων διανυσμάτων που μάλλον οφείλεται στο πρώτο στάδιο που εμείς έχουμε παραβλέψει. Το δεύτερο είναι η ομοιότητα των εικόνων που έχουν προκύψει από τα δικά μας διανύσματα χωρίς όμως να μπορούμε να πούμε ότι η ομοιότητα αυτή γίνεται “αντιληπτή” από τον ταξινομητή.



**Σχήμα 3.5** (α) αρχική εικόνα, (β) οι κλίσεις βασισμένες τα ιστογράμματα για τον περιγραφέα που εξάγεται από την VL\_Feat, (γ) οι κλίσεις βασισμένες τα ιστογράμματα για τον περιγραφέα μας στην πλήρη υλοποίηση και (δ) για την υλοποίηση χωρίς επικαλυπτόμενα blocks.

Μετά και από την κανονικοποίηση σε επικαλυπτόμενα blocks, “ξεδιπλώνουμε” τον πίνακα μας και προκύπτει το χαρακτηριστικό διάνυσμα που θα εισάγουμε στον ταξινομητή. Μπορούμε πλέον να κάνουμε μια αξιολόγηση, με βάση την Ακρίβεια ( $\text{precision} = \frac{TP}{TP+FP}$ ), συγκρίνοντας τον περιγραφέα που έχουμε εξάγει με αυτόν της VL\_Feat. Για τις αξιολογήσεις αυτές χρησιμοποιήσαμε τον ίδιο ταξινομητή με αυτόν που χρησιμοποιήθηκε στην εργασία των Dalal & Triggs [4], ένα γραμμικό Support Vectors Machine. Τα δεδομένα που χρησιμοποιήθηκαν για την αξιολόγηση ήταν από τη βάση

δεδομένων πεζών του Τεχνολογικού Ινστιτούτου Μασαχουσέτης (MIT). Η βάση περιέχει περίπου εννιακόσιες εικόνες με πεζούς και δημιουργήσαμε άλλες τόσες χωρίς την παρουσία πεζού. Η σύγκριση έγινε για έξι διαφορετικές σε πλήθος ομάδες εκπαίδευσης και αξιολόγησης. Η διαδικασία της ταξινόμησης αναλύεται σε επόμενο κεφάλαιο οπότε εδώ θα μας απασχόλησαν μόνο τα αποτελέσματα ως κριτήριο για την απόδοση του περιγραφέα μας.



**Διάγραμμα 3.1**Precision για τον Περιγραφέα της VL\_feat και την δικιά μας υλοποίηση.

Σύμφωνα με τους συγγραφείς η συγκεκριμένη βάση δίνει πολύ καλά αποτελέσματα, πράγμα που επιβεβαιώνεται και στην δική μας περίπτωση και αυτό οφείλεται στο ότι στη βάση του MIT δεν υπάρχει μεγάλη ποικιλία στις πόζες που έχουν οι πεζοί στις φωτογραφίες.

### 3.3 Απλοποιήσεις στην υλοποίηση του περιγραφέα

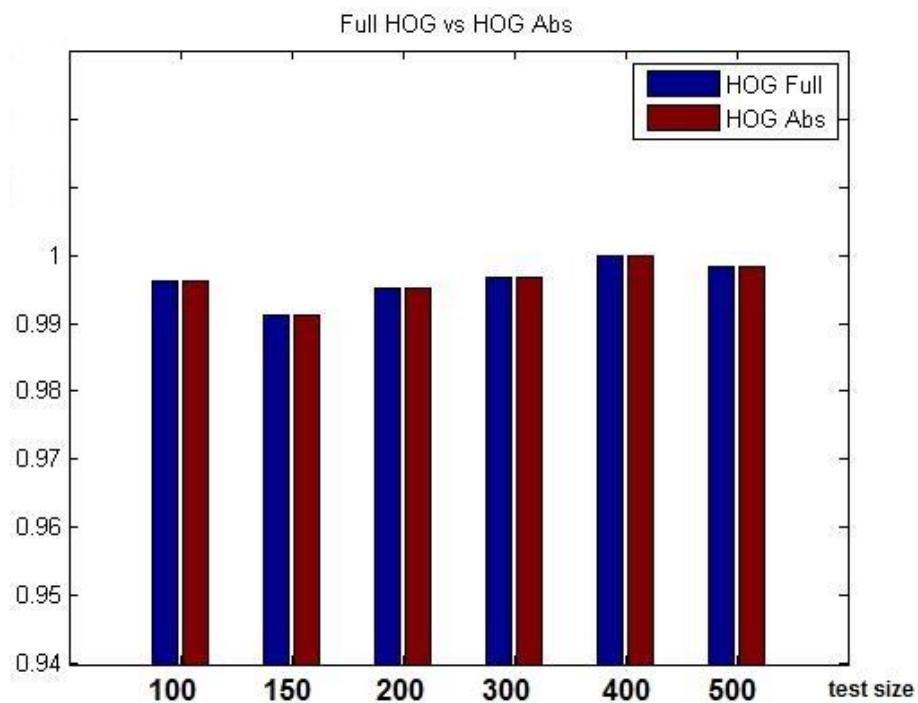
Όπως αναφέρθηκε στην αρχή του κεφαλαίου ο περιγραφέας στην πλήρη του μορφή είναι αρκετά σύνθετος για να υλοποιηθεί κυκλωματικά για σύστημα πραγματικού χρόνου λόγω των καθυστερήσεων που εισάγουν τα κυκλώματα. Κατά συνέπεια πρέπει να απλοποιηθεί είτε κάνοντας αλλαγές στον τρόπο εξαγωγής του διανύσματος είτε κάνοντας απαλοιφές επεξεργασιών. Κάθε πιθανή τροποποίηση έπρεπε πρώτα να υποστεί μια αξιολόγηση στο Matlab. Η αποδοχή των τροποποιήσεων έγινε ανάλογα με τα αποτελέσματα, αν και κάποιες φορές ο περιορισμός του χρόνου και του χώρου οδηγεί σε υποχρεωτικές αλλαγές που ενώ μειώνουν την απόδοση αισθητά, η εφαρμογή τους αποτελεί μονόδρομο. Τέλος να σημειώσουμε ότι αποδοχή μιας τροποποίησης σημαίνει ότι οι επόμενες δοκιμές την ή τις συμπεριλαμβάνουν.

### 3.3.1 Αντικατάσταση Μέτρων

Η πρώτη μετατροπή, που είναι σχεδόν τετριμμένη σε επεξεργασία εικόνων σε FPGAs, είναι η αντικατάσταση της Ευκλείδειας νόρμας με την L1με τις απόλυτες τιμές :

$$\sqrt{a^2 + b^2} \Rightarrow |a| + |b|$$

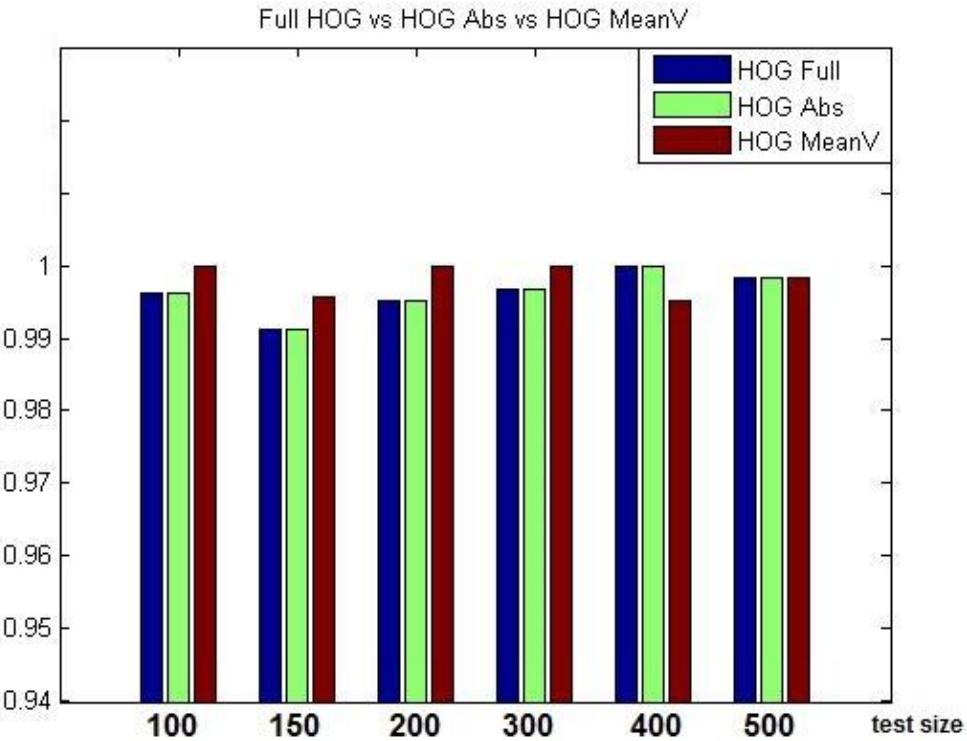
Με τη διαδικασία αυτή απαλλασσόμαστε από δύο διαδικασίες απαιτητικές σε κύκλωμα, με ανεπιθύμητη χρονική καθυστέρηση. Η πρώτη είναι ο υπολογισμός του τετραγώνου ενός αριθμού, ο οποίος γίνεται με διαδοχικούς πολλαπλασιασμούς. Η δεύτερη διαδικασία είναι ο υπολογισμός τετραγωνικής ρίζας που κυκλωματικά είναι μια επαναληπτική διαδικασία και κατά συνέπεια ιδιαίτερα χρονοβόρα. Ο υπολογισμός των απόλυτων τιμών κυκλωματικά σημαίνει ότι αγνοούμε το bit προσήμου, που είναι εξαιρετικά απλό και γρήγορο στην υλοποίηση. Η αλλαγή αυτή έχει αμυδρή επίπτωση στην τελική απόδοση.



**Διάγραμμα 3.2** *HOG full*: ο περιγραφέας στην πλήρη του μορφή  
*HOG Abs*: αντικατάσταση των μέτρων με άθροισμα απολύτων

### 3.3.2 Αντικατάσταση L2 Νόρμας με Μέσο όρο τιμών

Επόμενη δοκιμή είναι η αντικατάσταση τις L2 Νόρμας με τον μέσο όρο των τιμών των εικονοστοιχείων του block. Ο υπολογισμός της L2 Νόρμας περιέχει τετραγωνική ρίζα και διαιρεση. Με την αντικατάσταση της από τον Μέσο όρο απαλλασσόμαστε και από τη δεύτερη τετραγωνική ρίζα που περιείχαν οι υπολογισμοί μας. Τα αποτελέσματα της αξιολόγησης έδειξαν ότι και αυτή η αλλαγή μπορεί να υιοθετηθεί στους υπολογισμούς μας χωρίς σημαντική μείωση της απόδοσης στο σύστημά μας



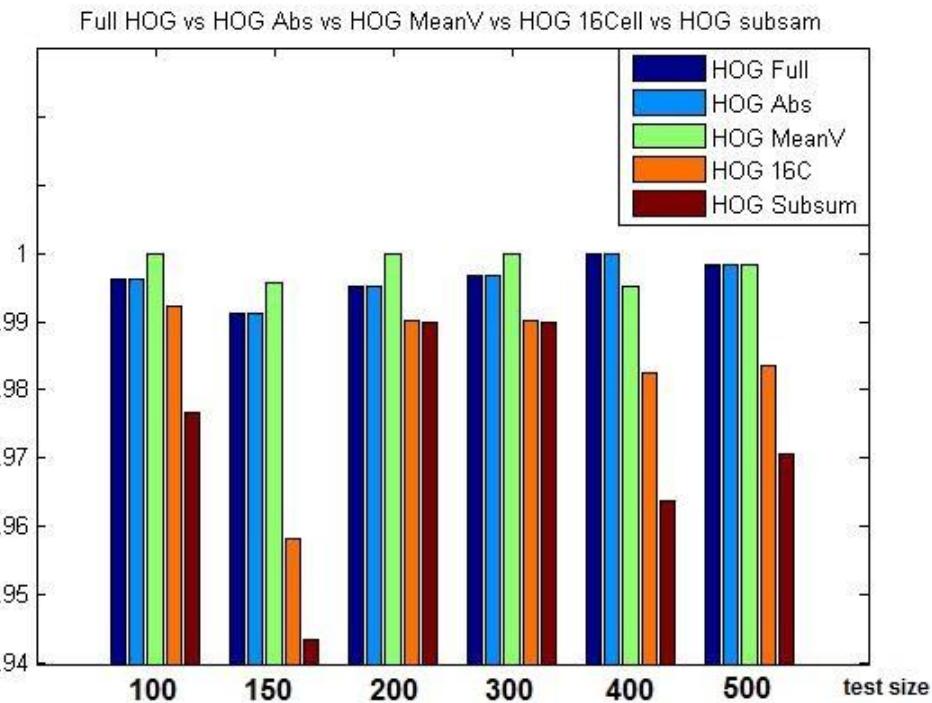
**Διάγραμμα 3.3** *HOG full*: ο περιγραφέας στην πλήρη του μορφή *HOG Abs*: αντικατάσταση των μέτρων με άθροισμα απολύτων *HOG MeanV*: ο *HOG Abs* με επιπλέον αλλαγή την αντικατάσταση της L2 Νόρμας με μέσο όρο των εικονοστοιχείων του Cell

### 3.3.3 Αύξηση του μεγέθους του Cell και υπο-δειγματοληψία

Σε μια προσπάθεια να μειώσουμε τον όγκο των δεδομένων μας δοκιμάσαμε να αυξήσουμε το μέγεθος του cell. Ο διπλασιασμός του cell οδηγεί σε υποτετραπλασιασμό του μεγέθους του χαρακτηριστικού μας διανύσματος αφού πλέον τα blockγίνονται ( $4 \times 8 = 32$ ) σε κάθε παράθυρο εντοπισμού και τα συνολικά μας δεδομένα  $32 \times 36 = 1152$ , από 4608.

Το ίδιο ακριβώς αποτέλεσμα στον όγκο δεδομένων του χαρακτηριστικού μας διανύσματος έχει και η υποδειγματοληψία της εικόνας. Σε αυτή την περίπτωση μικραίνει πάλι το πλήθος των cells αλλά αυτό οφείλεται στη σμίκρυνση του παράθυρου εντοπισμού που καταλήγει να έχει μέγεθος  $32 \times 64$ , δηλαδή πάλι θα αποτελείται από 32 cells. Τα αποτελέσματα της ταξινόμησης και στις δύο περιπτώσεις ήταν αρκετά μειωμένα και ως εκ τούτου δεν υιοθετήθηκε κάποια από τις δύο αλλαγές. Τα παραπάνω αποτελέσματα δεν μας εξέπληξαν

αφού, στην πρώτη περίπτωση γενικεύεται η πληροφορία του περιγραφέα μας ενώ στην δεύτερη αγνοείται το 50% της εικόνας και κατά συνέπεια της πληροφορίας μας.



**Διάγραμμα 3.4 HOG full:** ο περιγραφέας στην πλήρη του μορφή

*HOG Abs:* αντικατάσταση των μέτρων με άθροισμα απολύτων

*HOG MeanV:* ο HOG Abs με επιπλέον αλλαγή την αντικατάσταση της L2 Νόρμας με μέσο όρο των εικονοστοιχείων του Cell

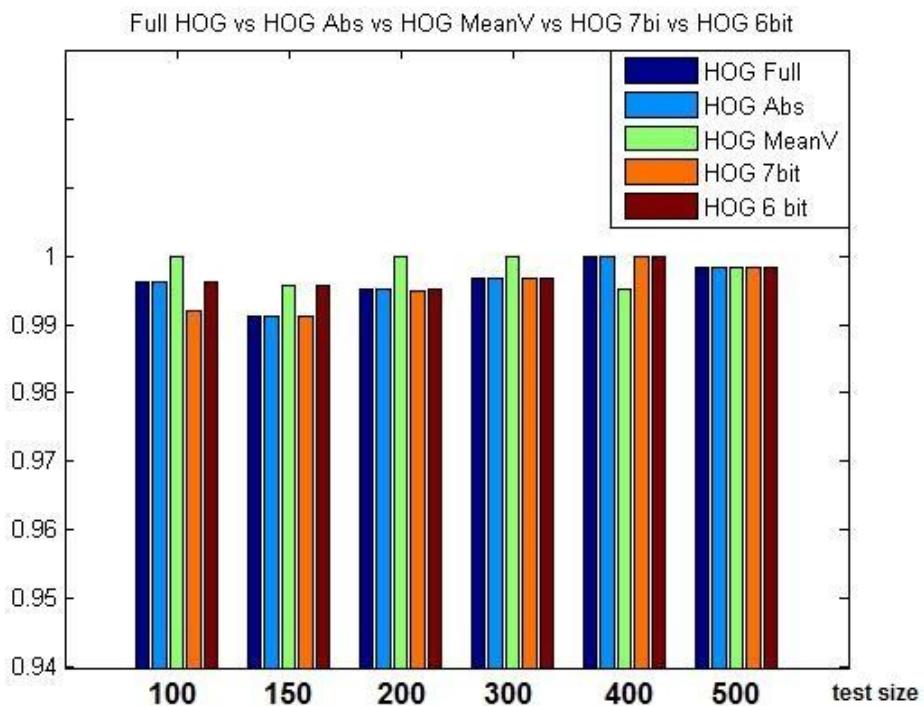
*HOG 16C:* ο HOG Abs με cell 16x16 εικονοστοιχεία

*HOG Subsam:* ο HOG Abs με υποδειγματοληψία στον υπολογισμό του παραθύρου εντοπισμού

### 3.3.4 Κβάντιση εικονοστοιχείων

Η μνήμη σε ένα FPGAmετριέται σε bitsκαι όχι σε bytes. Αυτό πρακτικά σημαίνει ότι αν μπορέσουμε να περιορίσουμε έστω και ένα bit από κάθε εικονοστοιχείο σε μια VGA εικόνα ( $640 \times 480 = 307.200$  εικονοστοιχεία) τότε έχουμε πετύχει μεγάλη εξοικονόμηση υλικών πόρων. Η κβάντιση σε επίπεδο υλικού είναι η απαλοιφή ενός ή περισσοτέρων least significant bits (LSB). Στο Matlab η κβάντιση επιτυγχάνεται με διαίρεση του αριθμού με το

$2^n$ , όπου  $n$  το πλήθος των bit που θέλουμε να περικόψουμε, και στρογγυλοποίηση του πηλίκου στον αμέσως μικρότερο ακέραιο.



**Διάγραμμα 3.5 HOG full:** ο περιγραφέας στην πλήρη του μορφή

**HOG Abs:** αντικατάσταση των μέτρων με άθροισμα απολύτων

**HOG MeanV:** ο HOG Abs με επιπλέον αλλαγή την αντικατάσταση της L2 Νόρμας με μέσο όρο των εικονοστοιχείων του Cell

**HOG 7bit:** ο HOG Abs με εικονοστοιχεία κβαντισμένα στα 7 bit

**HOG 6bit:** ο HOG Abs με εικονοστοιχεία κβαντισμένα στα 6 bit

Φαίνεται πως οι αλλαγές μας δεν έχουν μεγάλες επιπτώσεις στην επίδοση του περιγραφέα. Σαφώς γνωρίζουμε ότι η βάση που χρησιμοποιήσαμε είναι σχετικά εύκολη και μικρή σε όγκο δεδομένων και δεν μπορεί να μας πει με ακρίβεια τα αποτελέσματα που θα έχουμε σε ένα πραγματικό σύστημα όπου η εμφάνιση των πεζών μπορεί να αποτυπώνονται αρκετά διαφορετικά. Οι δοκιμές αυτές πάραυτα είναι ενδεικτικές για το πόσο χειρότερα ή καλύτερα είναι τα αποτελέσματα σε σχέση με την πλήρη μορφή του περιγραφέα που υλοποιήσαμε.



## Κεφάλαιο 4

### Ταξινόμηση

#### 4.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο περιγράφηκε η διαδικασία δημιουργίας ενός χαρακτηριστικού διανύσματος για την εύρεση πεζού σε μια εικόνα. Το χαρακτηριστικό διάνυσμα είναι η αναπαράσταση δεδομένων που προκύπτουν από την επεξεργασία κάποιου ή κάποιων χαρακτηριστικών της εικόνας. Η διαδικασία κατά την οποία γίνεται η μελέτη των χαρακτηριστικών που έχουν εξαχθεί από προηγούμενη επεξεργασία, με σκοπό να αποφασιστεί εάν το περιεχόμενο μιας εικόνας ανήκει ή όχι σε μια ήδη γνωστή κλάση λέγεται ταξινόμηση. Στην ταξινόμηση είναι εκ των προτέρων γνωστές οι κλάσεις και έχει προηγηθεί εκπαίδευση του ταξινομητή με δεδομένα τα οποία γνωρίζουμε που ανήκουν. Ταξινομητής είναι το μοντέλο το οποίο εκπαιδεύεται χρησιμοποιώντας χαρακτηριστικά διανύσματα γνωστής κλάσης, με σκοπό να χρησιμοποιήσει τη γνώση αυτή για να κατηγοριοποιήσει νέα δεδομένα. Υπάρχει πληθώρα ταξινομητών και κατηγοριοποιούνται ανάλογα με τον τρόπο που εξάγουν αποφάσεις. Η επιλογή ενός ταξινομητή έχει να κάνει με ένα συνδυασμό χαρακτηριστικών εκ των οποίων τα κυριότερα είναι η υπολογιστική του πολυπλοκότητα, η ακρίβειά του, η ευαισθησία του και η εγκυρότητά του.

Στην εργασία αυτή σκοπός είναι η ανίχνευση πεζών σε εικόνες χρησιμοποιώντας ένα κυλιόμενο σταθερού μεγέθους παράθυρο, το παράθυρο εντοπισμού. Για το σκοπό της

παρούσας εργασίας μελετήθηκαν δύο διαφορετικά μοντέλα επιβλεπόμενης εκμάθησης. Το πρώτο είναι αυτό των Support Vectors Machine, όπου χρησιμοποιείται στην εργασία των Dalal & Triggs [4]. Το δεύτερο μοντέλο είναι αυτό της Λογιστικής Παλινδρόμησης. Οι κύριες αιτίες για τις οποίες αποφασίστηκε η αλλαγή ταξινομητή και υλοποιήθηκε ταξινόμηση με Λογιστική Παλινδρόμηση (Logistic Regression), ήταν η ταχύτητα ταξινόμησης και η μείωση της υπολογιστικής πολυπλοκότητας. Στο κεφάλαιο αυτό γίνεται μια συνοπτική αναφορά στους ταξινομητές που χρησιμοποιήθηκαν για την παρούσα εργασία και παρουσιάζονται τα αποτελέσματα των δοκιμών.

## 4.2 Ταξινομητές

Οι ταξινομητές είναι μοντέλα ή αλγόριθμοι που επεξεργάζονται συγκεκριμένα χαρακτηριστικά για να κατατάξουν ένα υπό μελέτη αντικείμενο σε μια γνωστή εκ των προτέρων, κατηγορία. Προϋπόθεση για να γίνει η ταξινόμηση είναι η εκπαίδευση του ταξινομητή με ένα σύνολο δεδομένων τα οποία είναι γνωστό σε ποια κλάση ανήκουν. Σε μια υπεραπλουστευμένη αναπαράσταση ο ταξινομητής απαντάει στο εξής ερώτημα: "Αν η κατηγορία A έχει τα X χαρακτηριστικά και η κατηγορία B τα Y χαρακτηριστικά σε ποια από τις δύο ανήκουν τα νέα χαρακτηριστικά που θα του δώσω;".

Η ταξινόμηση που έπρεπε να γίνει στην εργασία αυτή είναι δυαδική ταξινόμηση και αποφαίνεται για την ύπαρξη πεζού σε μια εικόνα. Τα χαρακτηριστικά που τροφοδοτούν τους ταξινομητές είναι το χαρακτηριστικό διάνυσμα που έχει εξαχθεί σε προηγούμενη επεξεργασία της εικόνας. Το μήκος του διανύσματος αποτελεί το πλήθος των χαρακτηριστικών.

Υπάρχουν διάφορα ήδη ταξινομητών ανάλογα με τον τρόπο που εξάγεται το αποτέλεσμα. Στην εργασία αυτή θα αναλυθούν μόνο οι δύο στατιστικοί ταξινομητές που χρησιμοποιήθηκαν. Ο πρώτος που χρησιμοποιείται και από τους Dalal και Triggs [4] είναι ο Support Vectors Machine και ο δεύτερος που δοκιμάστηκε και εφαρμόστηκε τελικά είναι η Λογιστική Παλινδρόμηση.

Τα δεδομένα που χρησιμοποιήθηκαν για την εκπαίδευση είναι ο συνδυασμός των βάσεων Πεζών του MIT και της INRIA. Έτσι δημιουργήθηκε ένα σύνολο δεδομένων εκπαίδευσης με

6000 περίπου εικόνες. Οι τρεις χιλιάδες από αυτές ανήκαν στις βάσεις με τους πεζούς και αποτέλεσαν τα θετικά δείγματα εκπαίδευσης και οι υπόλοιπες δημιουργήθηκαν τεμαχίζοντας αρνητικές εικόνες μεγαλύτερων διαστάσεων για να αποτελέσουν τα αρνητικά δείγματα εκπαίδευσης. Η εκπαίδευση κατά την οποία δίνονται οι κλάσεις στις οποίες θα ταξινομηθούν τα δεδομένα, λέγεται επιβλεπόμενη εκμάθηση (supervised learning) ενώ αν πρέπει να δημιουργηθούν κλάσεις από τα δεδομένα τότε πρόκειται για εκπαίδευση χωρίς επίβλεψη (unsupervised learning).

Στη μάθηση με επίβλεψη το σύστημα “μαθαίνει” επαγωγικά μέσω ενός συνόλου δεδομένων  $\{x, y\}$  μια συνάρτηση  $f$ , η οποία αποτελεί την περιγραφή ενός μοντέλου. Υπάρχει πάντα κάποιος “επιβλέπων” ο οποίος παρέχει τη σωστή τιμή εξόδου για της συνάρτησης για τα δεδομένα που εξετάζονται. Αντίθετα, στη μάθηση χωρίς επίβλεψη το σύστημα δημιουργεί πρότυπα ανακαλύπτοντας συσχετίσεις ή ομάδες μεταξύ των δεδομένων για τα οποία η τιμή εξόδου για της συνάρτησης είναι γνωστή. Το αποτέλεσμα είναι ένα σύνολο προτύπων - περιγραφών, κάθε ένα από τα οποία περιγράφει ένα μέρος των δεδομένων.

## 4.3 Support Vector Machines

Τα Support Vector Machines είναι ένα είδος supervised, δυαδικού Ταξινομητή. Στην ελληνική βιβλιογραφία συναντάται ως Μηχανές Υποστήριξης Διανυσμάτων ή Μηχανές Διανυσμάτων Υποστήριξης. Στην εργασία αυτή θα χρησιμοποιηθεί η αγγλική συντομογραφία - SVM.

Οι SVM ταξινομητές επιλύοντας ένα Convex πρόβλημα βελτιστοποίησης, καταλήγουν στην εύρεση ενός υπερεπιπέδου, το οποίο διαχωρίζει το σύνολο των δεδομένων εκπαίδευσης με τέτοιο τρόπο ώστε τα δεδομένα που ανήκουν στην ίδια ομάδα να βρίσκονται στην ίδια μεριά του υπερεπιπέδου. Επειδή συνήθως το υπερεπίπεδο δεν είναι μοναδικό, ο ταξινομητής αναζητάει εκείνο το υπερεπίπεδο που απέχει όσο το δυνατόν περισσότερο από τα κοντινότερα δεδομένα εκπαίδευσης. Αναζητά δηλαδή το υπερεπίπεδο με τα μεγαλύτερα περιθώρια - maximal margin hyperplane. Όσο μεγαλύτερα είναι τα περιθώρια τόσο μικρότερο είναι το σφάλμα γενίκευσης του ταξινομητή.

Μια ακόμα κατηγοριοποίηση του ταξινομητή είναι σε γραμμικό (linear) και μη γραμμικό (non linear). Ακολουθώντας για μια ακόμα φορά τα βήματα των Dalal & Triggs [4] χρησιμοποιήθηκε μόνο ο γραμμικός ταξινομητής.

Από την πρώτη παρουσίαση των SVM μέχρι σήμερα έχουν παρουσιαστεί πολλές εναλλαγές τους καθιστώντας έτσι τους SVM ταξινομητές κατάλληλους για πολλά είδη ταξινόμησης ή κατηγοριοποίησης.

### 4.3.1 Γραμμικός SVM

Αν θεωρήσουμε ένα σύνολο δεδομένων εκπαίδευσης  $D$  με  $-n$  – πλήθος δεδομένων τα οποία περιγράφονται ως εξής :

$$D = \{(x_i, y_i) \mid x_i \in R^p, y_i \in \{-1, +1\}\}, \quad \text{όπου}$$

-  $i=1, \dots, n$

- **Δείναι σύνολο σημείων**

- το  $y_i$  είναι  $-1$  ή  $1$  ανάλογα την κλάση που ανήκει το διάνυσμα  $x_i$  που είναι  $p$  διαστάσεων

Σκοπός του ταξινομητή είναι χρησιμοποιώντας τα δεδομένα εκπαίδευσης να βρει το υπερεπίπεδο με τα μεγαλύτερα περιθώρια που διαχωρίζει τα δεδομένα που έχουν  $y=-1$  από αυτά που έχουν  $y=1$ . Κάθε υπερεπίπεδο μπορεί να γραφτεί σαν σύνολο των σημείων χπου ικανοποιεί την συνθήκη:

$$w \cdot x - b = 0,$$

όπου  $w \cdot x$  είναι το εσωτερικό γινόμενο του  $x$  με το κάθετο διάνυσμα του υπερεπιπέδου.

Μία παράμετρος  $\frac{b}{||w||}$  καθορίζει μια μετατόπιση από το αρχικό υπερεπίπεδο κατά μήκος του διανύσματος  $w$ . Στα γραμμικά διαχωρίσματα δεδομένα μπορούν να καθοριστούν δύο υπερεπίπεδα με τρόπο τέτοιο ώστε να μην υπάρχουν δεδομένα ανάμεσά τους και στη

συνέχεια να υπολογιστεί η μέγιστη απόσταση μεταξύ τους. Αυτά τα δύο υπερπίπεδα περιγράφονται από τις σχέσεις:

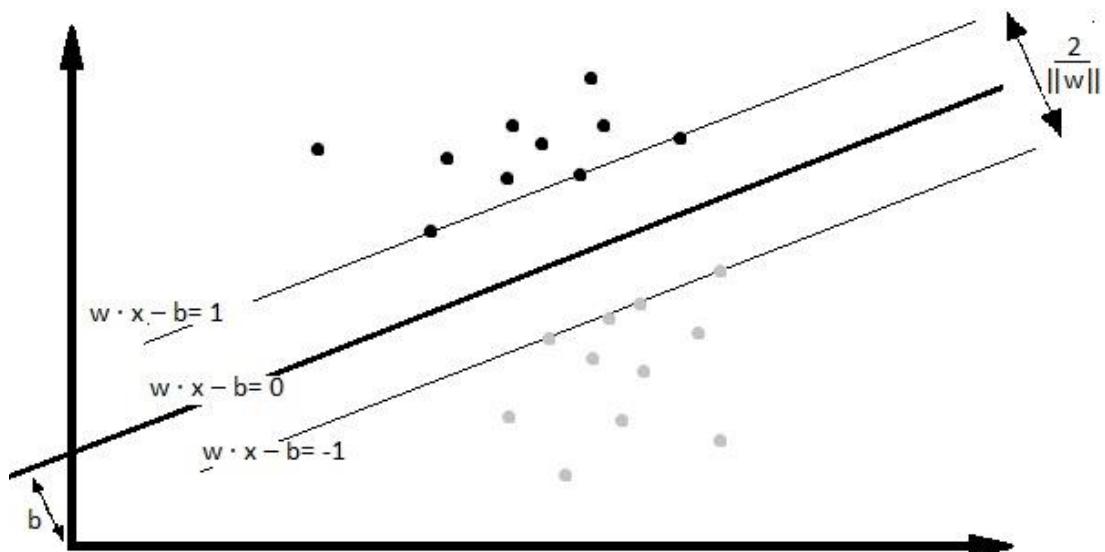
$$w \cdot x - b = -1$$

και

$$w \cdot x - b = 1$$

Η απόσταση ανάμεσα στα δύο τελευταία υπερπίπεδα είναι ίση με  $\frac{2}{\|w\|}$ . Συνεπώς η μέγιστη απόσταση δίνεται για τη μικρότερη δυνατή τιμή του  $w$  -  $b$  - η οποία υπολογίζεται με τη χρήση των πολλαπλασιαστών Lagrange.

Στο παρακάτω σχήμα (Σχήμα 4.1) γίνεται μια συνοπτική αναπαράσταση των όσων περιγράφηκαν προηγουμένως.



**Σχήμα 4.1** Υπερπίπεδα με μέγιστα περιθώρια.

Τα δεδομένα που είναι πάνω στα όρια των επιπέδων αποτελούν τα διανύσματα υποστήριξης.

Η κατάταξη ενός υπό μελέτη αντικειμένου σε μια από τις δύο περιοχές προκύπτει από το εσωτερικό γινόμενο των διανυσμάτων υποστήριξης με το χαρακτηριστικό διάνυσμα.

Η εκπαίδευση του γραμμικού SVMπου έγινε στο Matlabγια την πλήρη εκδοχή του περιγραφέα, έδωσε περισσότερα από 483 διανύσματα υποστήριξης, 4608 διαστάσεων το κάθε διάνυσμα. Τα μεγέθη αυτά καθιστούν σχεδόν αδύνατη την υλοποίηση SVMταξινομητή σε FPGA. Αφαιρώντας την διαδικασία κανονικοποίησης των block σε τοπικό επίπεδο, το

αποτέλεσμα ήταν 745 διανύσματα των 1152 διαστάσεων μέγεθος που είναι επίσης απαγορευτικό για υλοποίηση σε FPGA. Η αλλαγή ταξινομητή μπορεί να δώσει εξίσου καλά αποτελέσματα αλλά με πολύ μικρότερο κόστος σε υλικό και χρόνο επεξεργασίας. Για αυτό ακριβώς και επιλέχθηκε τελικώς η μέθοδος της ταξινόμησης με Λογιστική Παλινδρόμηση (Logistic Regression).

## 4.4 Λογιστική Παλινδρόμηση – Logistic Regression

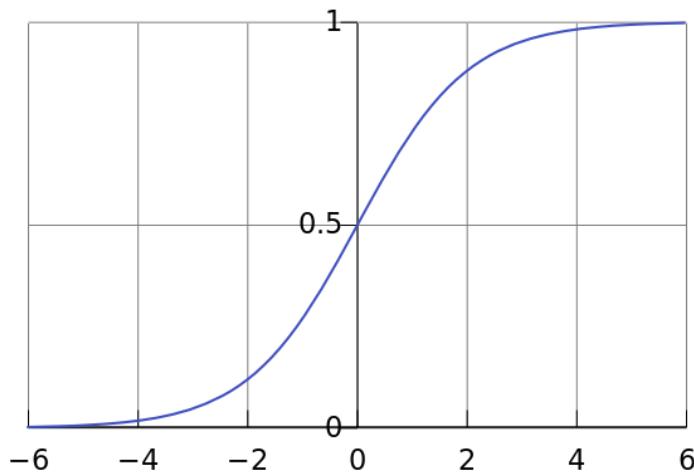
Η παλινδρόμηση χρησιμοποιείται στη Στατιστική και τη Θεωρία Πιθανοτήτων για να συσχετίσει μια εξαρτημένη μεταβλητή με μια ή περισσότερες ανεξάρτητες μεταβλητές. Η Λογιστική Παλινδρόμηση είναι μια μη γραμμική μορφή παλινδρόμησης βασισμένη στο Λογιστικό Μετασχηματισμό. Χρησιμοποιείται για να αποφασιστεί η ύπαρξη ή μη κάποιου χαρακτηριστικού. Είναι δηλαδή, όπως και ο SVM, ένας δυαδικός ταξινομητής.

Στην περίπτωσή μας χρησιμοποιούμε το πολλαπλό Λογιστικό μοντέλο, λόγω των πολλών ανεξάρτητων μεταβλητών. Η μορφή του περιγράφεται από την παρακάτω σχέση που είναι και αυτή που χρησιμοποιούμε στους υπολογισμούς μας:

$$Y = \frac{1}{1+e^{-\sum a*x}},$$

όπου  $a$  τα χαρακτηριστικά του κάθε δείγματος που επεξεργαζόμαστε και  $x$  τα βάρη που έχουν προκύψει από την εκπαίδευση του μοντέλου.

Η παραπάνω συνάρτηση είναι κατά βάση η σιγμοειδής συνάρτηση (Σχήμα 4.2). Για διάφορες τιμές του εκθέτη του  $e$ , η συνάρτηση δίνει μια τιμή στο  $Y$  μεταξύ του 0 και του 1. Η απόφαση λαμβάνεται με βάση μια τιμή που έχει οριστεί ως κατώφλι. Για τον καθορισμό του κατωφλίου ακολουθείται συγκεκριμένη διαδικασία η οποία περιγράφεται σε επόμενη παράγραφο.



**Σχήμα 4.2** Σιγμοειδής Συνάρτηση ([el.wikipedia.org](http://el.wikipedia.org))

Η χρήση της Λογιστικής Παλινδρόμησης αντί του ταξινομητή SVMέχει ως αποτέλεσμα τη μείωση κατά χλιάδες των πράξεων που απαιτούνται για να ληφθεί απόφαση σχετικά με την ύπαρξη ή όχι πεζού σε ένα παράθυρο εντοπισμού. Στην Λογιστική Παλινδρόμηση η εκπαίδευση επιστρέφει κάποια βάρη τα οποία είναι σε πλήθος όσα και τα χαρακτηριστικά του δεδομένου που εξετάζεται. Το γινόμενο των βαρών με τα αντίστοιχα χαρακτηριστικά του χαρακτηριστικού διανύσματος αποτελούν τον εκθέτη της προηγούμενης εξίσωσης.

- Ο υπολογισμός των βαρών γίνεται με τη χρήση της μεθόδου μέγιστης πιθανοφάνειας σε συνδυασμό με αριθμητικές μεθόδους για την εύρεση εκτιμητών. Είναι μια καθαρά μαθηματική διαδικασία η οποία δεν κρίνεται σκόπιμο να αναλυθεί σε αυτό το σημείο μιας και στη συγκεκριμένη εργασία η εύρεση των συντελεστών γίνεται με τη χρήση έτοιμης συνάρτησης του Matlab.

## 4.5 Σύγκριση Ταξινομητών

Ο SVM αποτελεί μια αξιόπιστη λύση στο θέμα της ταξινόμησης. Είναι όμως πολύ απαιτητικός σε υλικό και χρόνο εκτέλεσης. Η ταξινόμηση μια εικόνας γίνεται πολλαπλασιάζοντας τα Support Vectors που προκύπτουν από την εκπαίδευση με το διάνυσμα χαρακτηριστικών του παραθύρου που εξετάζουμε. Στις διάφορες εκπαιδεύσεις που έγιναν για να δοκιμαστεί ο ταξινομητής χρησιμοποιήθηκαν διαφορετικά σύνολα εκπαίδευσης. Το μικρότερο πλήθος Support Vectors που πήραμε χρησιμοποιώντας τον

πλήρη περιγραφέα ήταν 483 για διάνυσμα 4608 χαρακτηριστικών. Αυτό σημαίνει ότι για να γίνει ταξινόμηση ενός μόνο μέρους της εικόνας θα απαιτούνταν  $483 * 4608 = 2.225.664$  πολλαπλασιασμοί. Στην περίπτωση περιγραφέα χωρίς κανονικοποίηση σε επίπεδο block το πλήθος των Support Vectors ήταν 745, και το πλήθος των απαιτούμενων πολλαπλασιασμών για αντίστοιχη με πριν εικόνα  $745 * 1152 = 858.240$ .

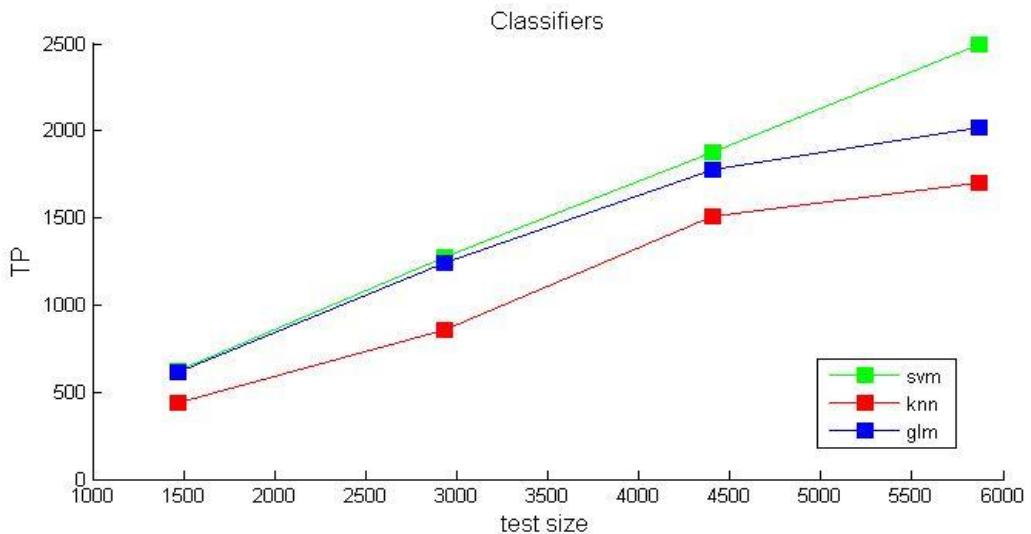
Τα προηγούμενα μεγέθη καθιστούν την υλοποίηση του SVM σε FPGA ιδιαίτερα περίπλοκη και θα προϋπέθετε μεγάλη έρευνα και δοκιμές. Για το λόγο αυτό κρίθηκε αναγκαία η επιλογή κάποιου άλλου ταξινομητή. Η ιδέα της λογιστικής παλινδρόμησης επικράτησε για την αποτελεσματικότητα που έχει σε σχέση με την απλή λειτουργία και την μικρή απαίτηση σε πράξεις. Οι πολλαπλασιασμοί που απατούνται για την ταξινόμηση με Λογιστική Παλινδρόμηση είναι όσοι και το μήκος του χαρακτηριστικού διανύσματος.

Πριν την τελική απόφαση έγιναν συγκρίσεις μεταξύ του SVM, της Λογιστικής Παλινδρόμησης και άλλης μιας μεθόδου της KNN. Η KNN μέθοδος ταξινομεί λαμβάνοντας υπόψη τους κοντινότερους “γείτονες” και χρησιμοποιήθηκε απλά και μόνο ως εργαλείο σύγκρισης, με σκοπό να υπάρξει άλλο ένα σημείο αναφοράς πέραν του SVM. Οι ακόλουθες γραφικές παραστάσεις αποτυπώνουν τα αποτελέσματα των δοκιμών.

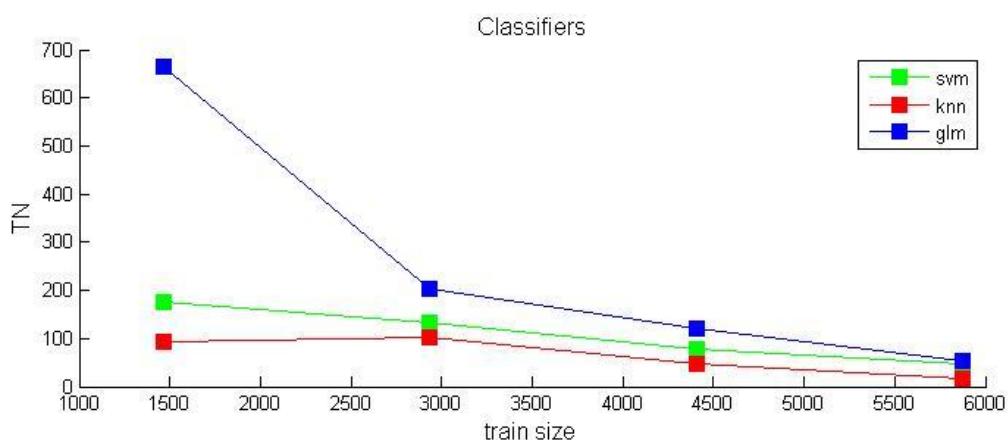
Οι πρώτες συγκρίσεις αφορούσαν τις σωστές ταξινομήσεις εικόνων με παρουσία (True Positives–TP) και χωρίς παρουσία πεζού (True Negatives–TN). Οι γραφικές παραστάσεις με τους παραπάνω συντελεστές δεν αποτελούν κάποια επίσημη μονάδα αξιολόγησης ταξινομητών, αλλά είναι ενδεικτικές αφού από αυτές προκύπτει η ακρίβεια (Accuracy) ενός ταξινομητή. Η ακρίβεια υπολογίζεται από τον τύπο:

$$\text{Accuracy} = \frac{(TP + TN)}{n},$$

όπου η το πλήθος των δειγμάτων που εξετάστηκαν

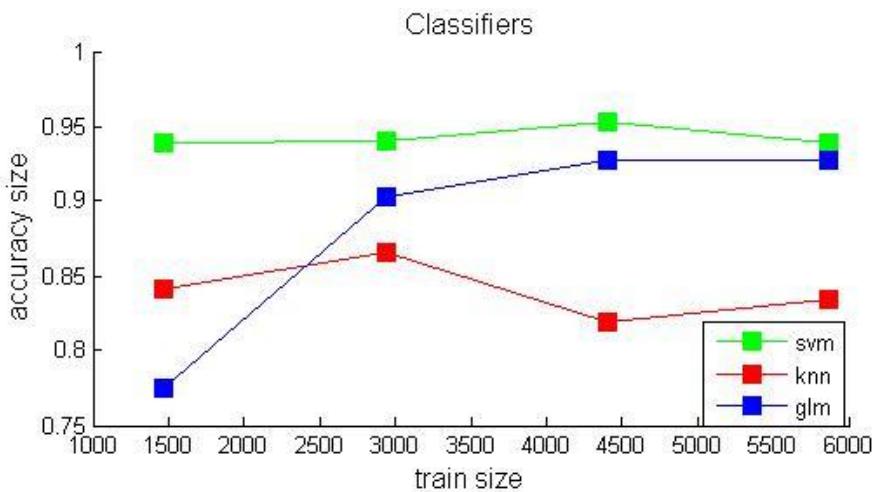


**Σχήμα 4.3** Σωστές εκτιμήσεις παρουσίας πεζού σε εικόνα συναρτήσει του πλήθους εκπαίδευσης



**Σχήμα 4.4** Σωστές εκτιμήσεις απουσίας πεζού από εικόνα συναρτήσει του πλήθους εκπαίδευσης

Στο ακόλουθο σχήμα παρουσιάζεται η ακρίβεια των ταξινομητών σε σχέση με το πλήθος εκπαίδευσης. Φαίνεται πως όσο αυξάνει το πλήθος των δεδομένων εκπαίδευσης τόσο βελτιώνεται η ακρίβεια του ταξινομητή με Λογιστική Παλινδρόμηση.



**Σχήμα 4.5** Γραφική Παράσταση της ακρίβειας συναρτήσει του πλήθους των δεδομένων εκπαίδευσης.

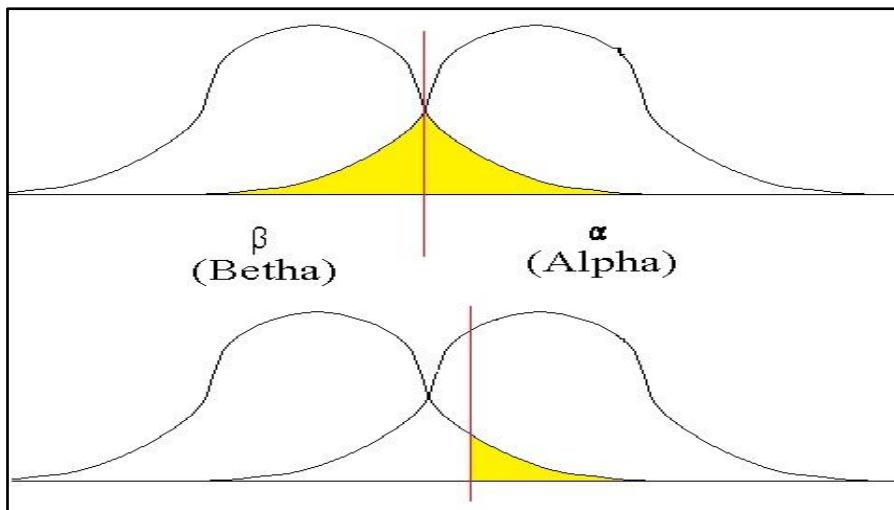
Η ακρίβεια που έχει εν τέλει η ταξινόμηση με Λογιστική Παλινδρόμηση σε σχέση με του SVM κρίνεται ικανοποιητική και μάλιστα σε σχέση με την πολυπλοκότητα της υλοποίησης και την πολυπλοκότητα χρόνου που έχει φαίνεται πως είναι κατά πολύ καταλληλότερη επιλογή για την συγκεκριμένη εφαρμογή.

## 4.6 Υπολογισμός Κατωφλίου

Ο υπολογισμός του κατωφλίου σε μια ταξινόμηση έχει βαρύνουσα σημασία γιατί καθορίζει το όριο απόφασης του ταξινομητή. Σύμφωνα με τις βασικές αρχές της θεωρίας πιθανοτήτων το όριο απόφασης είναι εκείνο το σημείο όπου ελαχιστοποιείται η πιθανότητα σφάλματος. Στο Σχήμα 4.6 φαίνεται στην επάνω γραφική παράσταση επιλογή ενός κατωφλίου στο σημείο ελαχίστου σφάλματος και στην κάτω γραφική παράσταση φαίνεται η επιλογή ενός κατωφλίου μετατοπισμένο δεξιά. Η μετατόπιση αυτή υποδηλώνει πως για να επικρατήσει το ενδεχόμενο Alpha θα πρέπει η συνάρτηση πιθανότητας να είναι τόσο μεγαλύτερη από αυτή της Beta όσο είναι και στη γραφική παράσταση.

Το κατώφλι στην παρούσα εργασία μελετήθηκε με βάση τις καμπύλες ROC (Receiver Operating Characteristic). Μελετήθηκαν και επιλέχθηκαν διάφορες τιμές κατωφλίων που

προέκυψαν από την μελέτη των ROCκαμπυλών, οι οποίες τιμές θα επιλέγονται μέσω των διακοπτών δύο θέσεων της πλατφόρμας DE2 – 70.



**Σχήμα 4.6** Επιλογή κατωφλίων

#### 4.6.1 Καμπύλες ROC

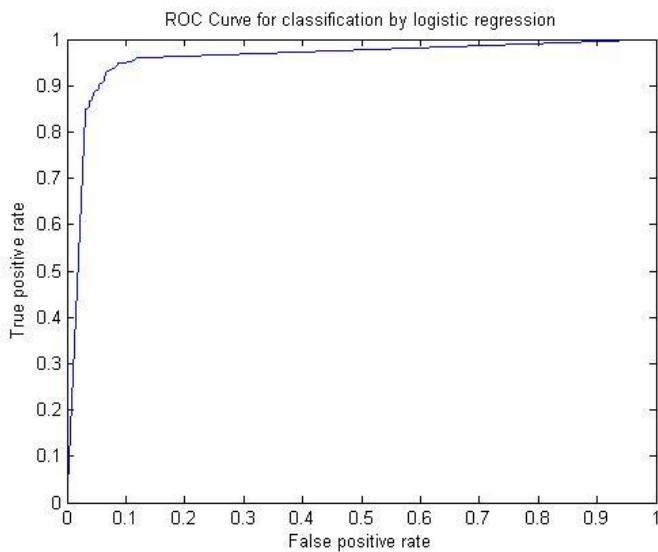
Οι καμπύλες ROC ορίζονται στο μοναδιαίο τετράγωνο με κάτω αριστερή γωνία το (0,0) και πάνω αριστερή το (1,1). Είναι μια γραφική παράσταση που αποτυπώνει την απόδοση ενός ταξινομητή λαμβάνοντας υπόψη το όριο ταξινόμησης (κατώφλι).

Στην παρούσα εργασία μελετήθηκαν οι καμπύλες ROC για διάφορα σύνολα εκπαίδευσης που διέφεραν μεταξύ τους ως προς το πλήθος των δεδομένων. Με την εντολή perfcurve του Matlab παίρνουμε τη γραφική παράσταση ROC αλλά υπολογίζεται από το πρόγραμμα και η βέλτιστη τιμή ορίου απόφασης. Στις παρακάτω εικόνες παρουσιάζονται οι γραφικές παραστάσεις όπως προέκυψαν από το Matlab.

1. Πλήθος δεδομένων εκπαίδευσης = 4698

Πλήθος δεδομένων ελέγχου= 1174

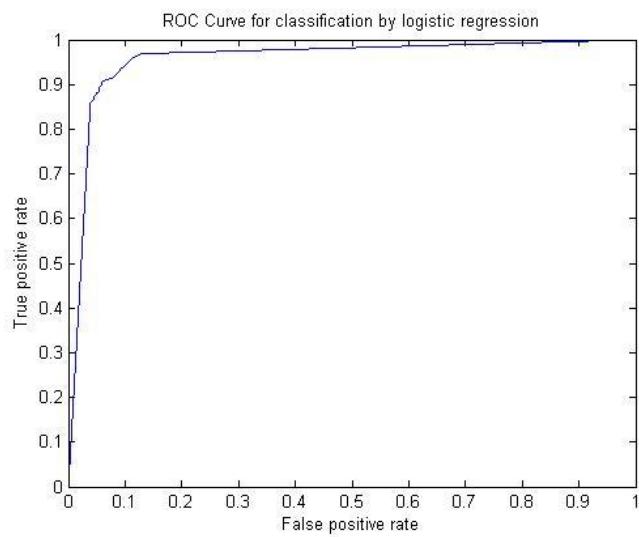
Βέλτιστο όριο απόφασης = 0,9574



2. Πλήθος δεδομένων εκπαίδευσης = 4111

Πλήθος δεδομένων ελέγχου= 1761

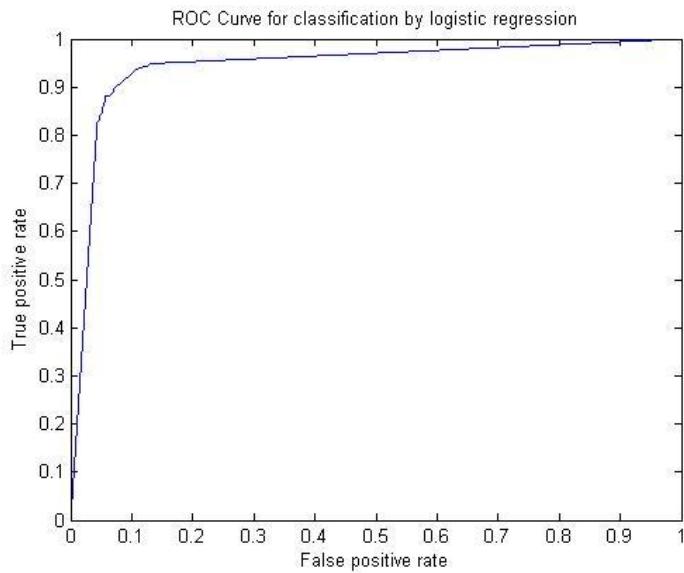
Βέλτιστο όριο απόφασης = 0,9574



3. Πλήθος δεδομένων εκπαίδευσης = 3524

Πλήθος δεδομένων ελέγχου= 2348

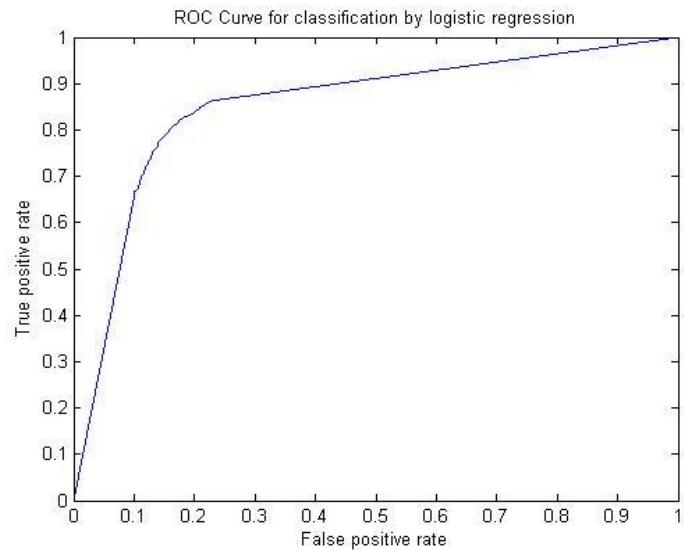
Βέλτιστο όριο απόφασης = 0,9438



4. Πλήθος δεδομένων εκπαίδευσης = 1762

Πλήθος δεδομένων ελέγχου= 4110

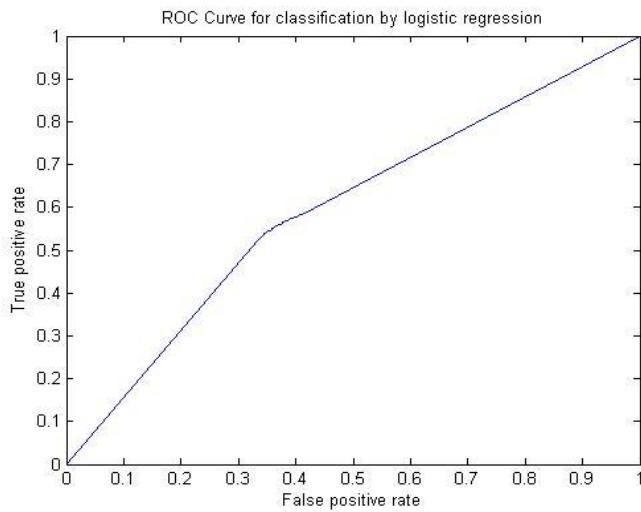
Βέλτιστο όριο απόφασης = 0,8732



5. Πλήθος δεδομένων εκπαίδευσης = 588

Πλήθος δεδομένων ελέγχου= 5284

Βέλτιστο όριο απόφασης = 0,5782



Φαίνεται ότι όσο μικραίνει το πλήθος των δεδομένων εκπαίδευσης η καμπύλη ROC τείνει να γίνεται ευθεία που σημαίνει ότι τίθεται θέμα καλής λειτουργίας του ταξινομητή. Η εκπαίδευση για την τελική εφαρμογή, δηλαδή την υλοποίηση, απ' όπου θα προκύψουν και οι συντελεστές που θα περαστούν στο FPGA, είναι αυτοί της πρώτης ROC καμπύλης.

## Κεφάλαιο 5

### Η υλοποίηση σε VHDL

#### 5.1 Εισαγωγή

Υλοποίηση του περιγραφέα σε υλικό, είναι η δημιουργία κυκλωμάτων που πραγματοποιούν τις μαθηματικές και λογικές πράξεις που προηγήθηκαν στο Matlab, με λογικά και αριθμητικά κυκλώματα. Για λόγους ευκολίας και καλύτερου ελέγχου της επεξεργασίας των δεδομένων, το κύκλωμα χωρίστηκε σε υπομονάδες ή αλλιώς υποκυκλώματα, όπου η έξοδος του ενός τροφοδοτεί το επόμενο. Η επεξεργασία των δεδομένων γίνεται παράλληλα κάνοντας χρήση της τεχνικής της σωλήνωσης (pipeline). Για να γίνει σωστά η παράλληλη επεξεργασία πρέπει τα δεδομένα να είναι απόλυτα συγχρονισμένα. Μικρές αποκλίσεις δεν δημιουργούν απλώς μικρά σφάλματα, αλλά απόλυτα λανθασμένα αποτελέσματα.

Στην υλοποίηση αυτή τελικός σκοπός ήταν να αποθηκευθούν σε μια μνήμη RAMοι συντεταγμένες για τις περιοχές που εντοπίστηκε πεζός. Οι συντεταγμένες αυτές δείχνουν το πάνω αριστερά εικονοστοιχείο μιας περιοχής  $64 \times 128$  εικονοστοιχείων. Η μνήμη αυτή διαβάζεται από ένα υποκύκλωμα και σχεδιάζει περιγράμματα καθορισμένου μεγέθους, με πάνω αριστερά σημείο αυτό που διαβάζει από τη μνήμη RAM.

## 5.2 Έλεγχος Ορθής Λειτουργίας

Πριν ξεκινήσει η περιγραφή των κυκλωμάτων κρίνεται απαραίτητο να γίνει μια αναφορά στον έλεγχο ορθής λειτουργίας της κάθε υπομονάδας. **Ενδελεχής έλεγχος πρέπει να γίνεται μετά από την ολοκλήρωση κάθε υποκυκλώματος.** Ο σωστός έλεγχος πρέπει να είναι σχολαστικός και κατά τη διάρκειά του να ελέγχεται το κύκλωμα, για όλες τις πιθανές καταστάσεις που μπορεί να περιέλθει. Ίσως αυτό να είναι κάποιες φορές αδύνατο οπότε το κύκλωμα θα πρέπει να ελέγχεται για πάρα πολλές τυχαίες καταστάσεις αλλά και μεταβάσεις λειτουργίας. Οι κυμματομορφές του Quartus είναι ένα εργαλείο που βοηθάει στο σωστό έλεγχο.

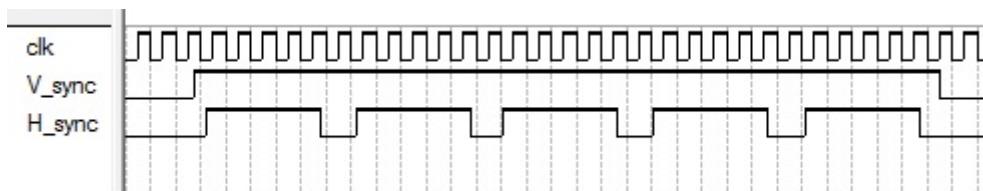
Ενδέχεται πολύ συχνά το πλήθος των υπολογισμών που απαιτείται για να ελεγχθεί η ορθή λειτουργία σε μια υλοποίηση να είναι τεράστιο. Ακόμα και τότε πρέπει ένα μεγάλο μέρος να ελεγχθεί ένα προς ένα. Σε τέτοιες περιπτώσεις ίσως είναι χρήσιμο και οικονομικό σε χρόνο να ελέγχεται η ορθή λειτουργία του κυκλώματος με τη βοήθεια κάποιου προγράμματος στο Matlab.

Για την εργασία αυτή ο χρόνος που δαπανήθηκε για τον έλεγχο των κυκλωμάτων ήταν πολύ περισσότερος από αυτόν που χρειάστηκε για να προγραμματιστούν τα κυκλώματα. Δεν θα γίνει ξανά αναφορά σε λεπτομερή έλεγχο πάνω στο κύκλωμα αφού όλα τα κυκλώματα ελέγχθηκαν και συγκρίθηκαν με τα αποτελέσματα του Matlab. ΜΟΝΟ όταν ήταν απόλυτα σωστά ξεκινούσε ο προγραμματισμός για το επόμενο υποκύκλωμα.

Για τον έλεγχο υποκυκλωμάτων που έπρεπε τα δεδομένα να είναι μικρογραφίες εικόνων, γράφτηκε κώδικας στο Matlabo οποίος έφτιαχνε τυχαίες εικόνες και για το Matlab και για τη VHDL μεσω μνήμης ROM (Testbench). Ήτσι υπήρχαν τα ίδια ακριβώς δεδομένα και στα δύο προγράμματα και ο έλεγχος των κυκλωμάτων γινόταν με μια σχετικά καλή ταχύτητα αλλά κυρίως με μεγάλη εγκυρότητα.

### 5.3 Λήψη εικονοστοιχείων

Το πρώτο πράγμα που έπρεπε να υπολογιστεί είναι οι Βαθμώσεις, δηλαδή τα  $G_x$  και  $G_y$  ακριβώς έγινε και στην υλοποίηση στο Matlab. Πριν γίνει όμως αυτό, εφόσον εδώ δεν υπήρχε ένας πίνακας με τις τιμές των εικονοστοιχείων της εικόνας, έπρεπε να διαβαστούν τα δεδομένα από τη είσοδο βίντεο της πλατφόρμας και να συγχρονιστούν. Τα δεδομένα εισέρχονται σειριακά με ρυθμό ένα δεδομένο ανά παλμό ρολογιού και συγχρονίζονται από δύο σήματα ελέγχου. Το πρώτο σήμα είναι για οριζόντιο συγχρονισμό ( $H_{sync}$ ) και σηματοδοτεί την αλλαγή της γραμμής και το δεύτερο ( $V_{sync}$ ) σηματοδοτεί την αλλαγή του καρέ (frame). Έτσι λοιπόν όταν υπάρχει ακμή του ρολογιού προς τα πάνω (rising edge) και αν τα δύο σήματα είναι ίσα με '1', γίνεται λήψη ενός εικονοστοιχείου. Αν το  $H_{sync}$  γίνει ίσο με '0' και το  $V_{sync}$  παραμείνει ίσο με '1' τότε ακολουθεί η λήψη της επόμενης γραμμής. Όταν και τα δύο γίνουν ίσα με '0' δηλώνεται το τέλος του καρέ. Στην παρακάτω εικόνα αποτυπώνονται τα σήματα συγχρονισμού για λήψη εικόνας 5x5.



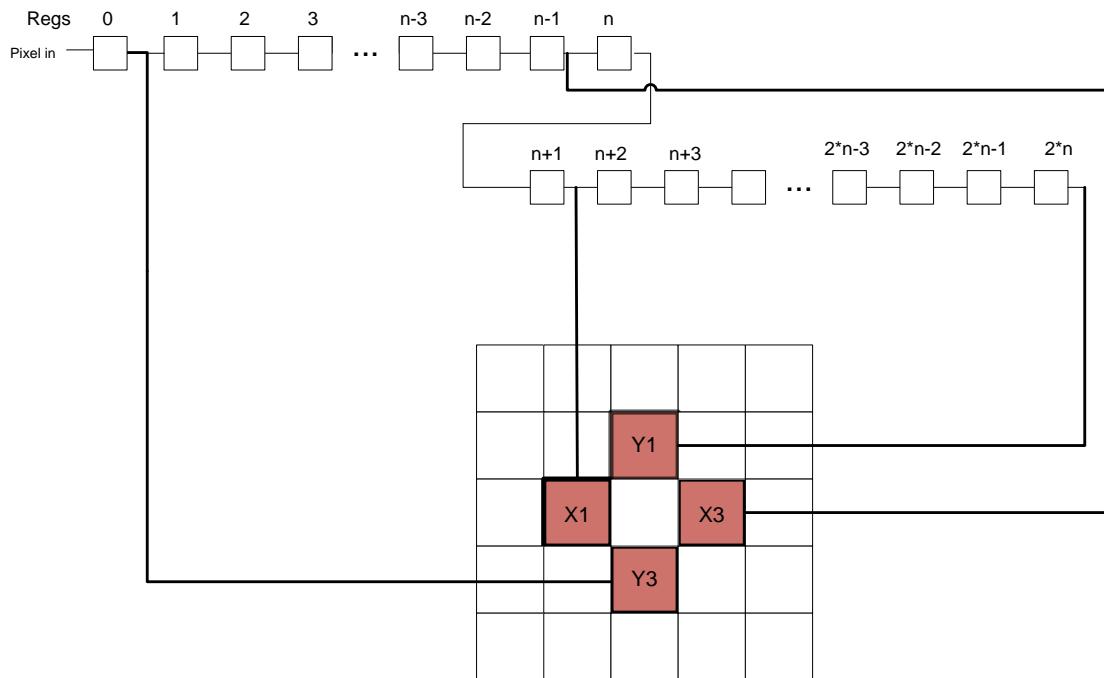
**Σχήμα 5.1** Τα σήματα συγχρονισμού για εικόνα 5x5

Για να υπολογιστεί η Βάθμωση σε κάποιο σημείο της εικόνας πρέπει να έχουν ληφθεί και τα τέσσερα γειτονικά του εικονοστοιχεία. Αυτό σημαίνει ότι πρέπει με κάποιο τρόπο να μετρώνται τα εικονοστοιχεία που εισέρχονται. Για εξοικονόμηση υλικών πόρων αλλά και ταχύτητας οι υπολογισμοί γίνονται κατά τη διάρκεια της λήψης των εικονοστοιχείων. Μόλις τελειώσει η επεξεργασία του τρέχοντος εικονοστοιχείου, το αποτέλεσμα προωθείται στο επόμενο υποκύκλωμα και ξεκινάει η επεξεργασία του επόμενου εικονοστοιχείου.

Να σημειωθεί εδώ ότι μια εικόνα εισέρχεται κατά γραμμές ξεκινώντας από το πάνω δεξιά εικονοστοιχείο. Αυτό σημαίνει πρακτικά ότι για να αρχίσει ο υπολογισμός των  $G_x$  και  $G_y$  για ένα εικονοστοιχείο πρέπει να έχει ληφθεί και το ακριβώς από κάτω του. Αν, για παράδειγμα, μια εικόνα περιέχει  $N$  εικονοστοιχεία ανά γραμμή τότε η κάθετη Βάθμωση

στην θέση του  $M$  εικονοστοιχείου μπορεί να υπολογιστεί μετά από τη λήψη του  $M+N$  εικονοστοιχείου. Η Βάθμωση στον οριζόντιο άξονα μπορεί να υπολογιστεί νωρίτερα, για την ακρίβεια, μόλις ληφθεί το επόμενο από το  $M$  εικονοστοιχείο. Στην εργασία αυτή προτιμήθηκε να γίνεται ο υπολογισμός του  $G_y$  και του  $G_x$  για ένα εικονοστοιχείο, την ίδια χρονική στιγμή.

Για επιτευχθεί ο συγχρονισμός των δεδομένων, που εισέρχονται σε διαφορετικές χρονικές στιγμές, χρησιμοποιήθηκαν γραμμές καταχωρητών. Κάθε καταχωρητής ισοδυναμεί με καθυστέρηση ενός ωρολογιακού παλμού και η έξοδος του ενός αποτελεί την είσοδο για τον επόμενο. Το κύκλωμα που υλοποιήθηκε περιγράφεται στο ακόλουθο σχήμα (Σχήμα 5.2). Είναι σημαντικό να διευκρινιστεί ότι οι καταχωρητές μπορούν να διαβαστούν οποιαδήποτε χρονική στιγμή.



**Σχήμα 5.2** Συστάδες καταχωρητών για το συγχρονισμό εικονοστοιχείων

Στην παρούσα υλοποίησή χρησιμοποιήθηκαν παραμετροποιημένα κυκλώματα. Αυτό σημαίνει ότι το μήκος της γραμμής, το πλήθος των γραμμών καθώς και άλλα μεγέθη μπορούν να αλλαχθούν χωρίς να αλλάζει η γενική συμπεριφορά του κάθε υποκυκλώματος. Τα παραμετροποιημένα κυκλώματα επιτρέπουν επίσης δοκιμές με τεχνητές εικόνες για την ορθή λειτουργία τους, αφού θα ήταν αδύνατο να ελεγχθεί η ορθή λειτουργία για κάθε ένα στοιχείο σε μια εικόνα μεγέθους  $480 \times 640$ . Μια ακόμη σημαντική επισήμανση είναι η

ύπαρξη μιας παραμέτρου που δίνει την δυνατότητα να μην συμπεριλαμβάνονται στην γενική επεξεργασία κάποιες σειρές και στήλες περιμετρικά της εικόνας. Αυτό έγινε για να αποφευχθεί το πρόβλημα των ακριανών στοιχείων που αναφέρθηκε στο Κεφάλαιο 3, δεδομένου ότι σε αυτά τα σημεία της εικόνας δεν υπάρχει σημαντική πληροφορία.

Αφού τα εικονοστοιχεία επιτεύχθηκε να λαμβάνονται στις κατάλληλες τετράδες, μπορούσαν ν να γίνουν οι αφαιρέσεις για τον υπολογισμό των Βαθμώσεων. Για τις πράξεις σε αυτό το στάδιο χρησιμοποιήθηκαν οι βιβλιοθήκες της Altera αλλά για να ληφθεί τελικό αποτέλεσμα χρειάστηκε να γίνουν και μερικές ακόμη μικρές παρεμβάσεις. Ο λόγος που έγινε αυτό, είναι γιατί θέλαμε η αναπαράσταση του αποτελέσματος να γίνεται με τον εξής τρόπο: Ο αριθμός να αναπαρίσταται με οκτώ δυαδικά ψηφία. Το όγδοο ψηφίο (Most Significant Bit)να αποτελεί το πρόσημο (1 για αρνητικό αριθμό και 0 για θετικό), ενώ τα υπόλοιπα αναπαριστούν την απόλυτη τιμή του αποτελέσματος σε απλό δυαδικό σύστημα. Αυτή η αναπαράσταση διευκόλυνε τον μετέπειτα υπολογισμό του bin.

Συνεπώς γράφτηκε κώδικας όπου σαν εισόδους έχει δύο θετικούς αριθμούς. Στην δική μας περίπτωση εισάγονται τα εφτά σημαντικότερα ψηφία από τα οχτώ, των  $X_1 - X_3$  και  $Y_1 - Y_3$ ,έτσι ώστε να γίνεται η κβάντιση που δοκιμάστηκε επιτυχώς στην εξομοίωση. Στο κύκλωμα που υλοποιήθηκε γίνεται μια φορά αφαίρεση που μας επιστρέφει την απόλυτη τιμή της διαφοράς των δύο αριθμών και παράλληλα γίνεται άλλη μια αφαίρεση απ' όπου προκύπτει το πρόσημο. Συνενώνοντας τα προηγούμενα αποτελέσματα λαμβάνονται τα  $Gx$  και  $Gy$  στη μορφή που θέλουμε.

## 5.4 Κύκλωμα υπολογισμού bin

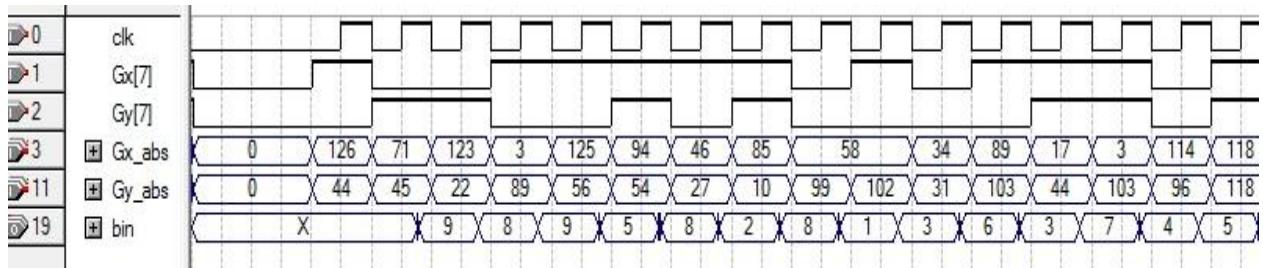
### 5.4.1 Υπολογισμός του bin

Μετά τον υπολογισμό των Βαθμώσεων χρειάζεται ένα υποκύκλωμα που υπολογίζει το binγια δύο αριθμούς  $Gx$  και  $Gy$ . Σαν εισόδους παίρνει τους παλμούς του ρολογιού και τις τιμές των Βαθμώσεων και εξάγει το binμετά από ενάμιση ωρολογιακό παλμό. Αρχικά γίνονται τέσσερις παράλληλοι πολλαπλασιασμοί για να δημιουργηθούν οι τιμές που χρησιμοποιούνται στις συγκρίσεις. Θυμίζουμε ότι ο υπολογισμός γίνεται εξετάζοντας

αρχικά τα πρόσημα των Βαθμώσεων και μετά κάνοντας συγκρίσεις. Κάνοντας απλές πράξεις μετατρέπεται η διαίρεση σε πολλαπλασιασμό, ο οποίος γίνεται για να υπολογιστούν οι τέσσερις οριακές τιμές των bin, εξ αρχής.

$$0.364 < \frac{Gy_{i,j}}{Gx_{i,j}} < 0.8391 \quad \Leftrightarrow \quad 0.364 * Gx_{i,j} < Gy_{i,j} < 0.8391 * Gx_{i,j}$$

Κατά συνέπεια με τέσσερις πολλαπλασιασμούς και παράλληλες συγκρίσεις υπολογίζεται το bin. Στο ακόλουθο σχήμα φαίνονται οι είσοδοι και έξοδοι του υποκυκλώματος καθώς και η χρονική του καθυστέρηση.



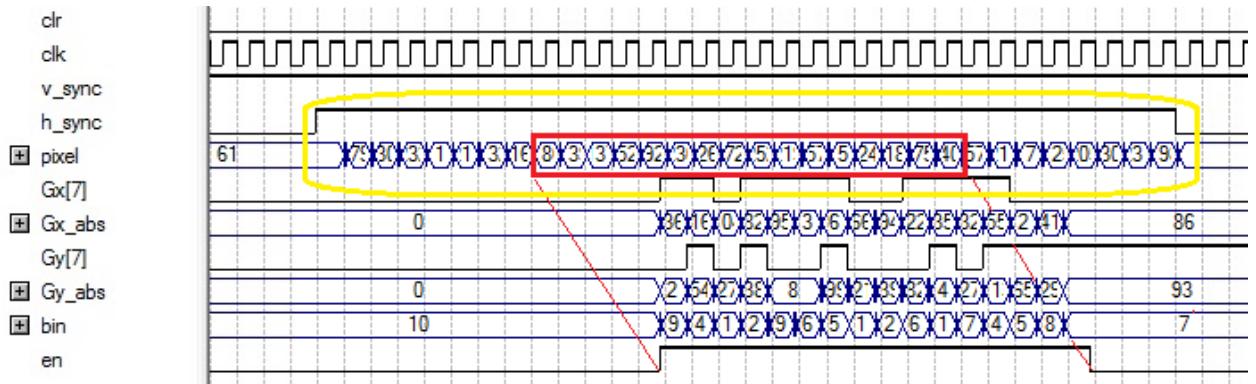
**Σχήμα 5.3** Κυματομορφή υπολικυλώματος υπολογισμού των bin. Τα  $G_x(7)$  και  $G_y(7)$  αναπαριστούν το πρόσημο των Βαθμώσεων και τα  $G_x_{abs}$  και  $G_y_{abs}$  απόλυτες τιμές τους

#### **5.4.2 Συγχρονισμός του αποτελέσματος**

Το επόμενο υποκύκλωμα είναι κύκλωμα συγχρονισμού. Οι διαδοχικές επεξεργασίες και υπολογισμοί αυξάνουν το ενδεχόμενο σφάλματος από ετεροχρονισμό των δεδομένων. Σε αυτό το κύκλωμα συνδυάζονται τα προηγούμενα υποκυκλώματα. Ως είσοδοι ορίστηκαν οι τιμές των εικονοστοιχείων, τα σήματα συγχρονισμού ( $H\_sync$  και  $V\_sync$ ), το ρολόι ( $clk$ ) και ένα σήμα “καθαρισμού” ( $clr$ ) των τιμών που βρίσκονται στους καταχωρητές (Σχήμα 5.4). Σκοπός είναι με αυτό το κύκλωμα, να συγχρονίζονται στην έξοδο τα  $Gx$ ,  $Gy$ ,  $bin$  και ένα σήμα το οποίο θα είναι ίσο με ‘1’ όσο εξάγονται δεδομένα.

Από το σημείο αυτό και για τη συνέχεια της υλοποίησης, τα σήματα εισόδου έπρεπε να είναι μικρές αναπαραστάσεις εικόνων έτσι ώστε να μπορεί να γίνει μια εκτίμηση της απόδοσης του κυκλώματος. Σε αυτό το σημείο γίνεται κατανοητή και η ανάγκη να δημιουργούνται κυκλώματα με παραμετροποιημένα μεγέθη. Έτσι για να ελεγχθεί ένα

κύκλωμα δεν είναι ανάγκη να εισάγεται μια εικόνα 640 x480, αφού μπορεί να γίνει έλεγχος σωστής λειτουργία με μικρότερες εικόνες διαφόρων διαστάσεων. Αυτός είναι ο μόνος τρόπος να εξασφαλιστεί ότι στα επόμενα στάδια της υλοποίησης θα εισάγονται τα σωστά δεδομένα.



**Σχήμα 5.4** Σήματα εξόδου bin, Gx, Gy, en. Με κίτρινο χρώμα σημειώνεται η λήψη δεδομένων γραμμής με κόκκινο τα δεδομένα που υπόκεινται σε επεξεργασία. Τα 8 πρώτα και τα 8 τελευταία εικονοστοιχεία της γραμμής έχουμε επιλέξει να μην υπόκεινται σε επεξεργασία.

Flow Status	Successful - Fri Aug 23 05:23:32 2013
Quartus II Version	7.2 Build 203 02/05/2008 SP 2 SJ Full Version
Revision Name	bin_sync
Top-level Entity Name	bin_sync
Family	Cyclone II
Device	EP2C70F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	648 / 68,416 (< 1 %)
Total combinational functions	147 / 68,416 (< 1 %)
Dedicated logic registers	586 / 68,416 (< 1 %)
Total registers	586
Total pins	34 / 422 (8 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	4 / 300 (1 %)
Total PLLs	0 / 4 (0 %)

**Σχήμα 5.5** Compilation Report – Flow Summary

Στο σημείο αυτό είναι μια καλή ευκαιρία να ελεγχθεί το πόσα στοιχεία από το FPGA χρησιμοποιεί μέχρι στιγμής το κύκλωμα που έχει υλοποιηθεί. Επειδή σε αυτό το σημείο συνδυάζονται όλα τα υποκυκλώματα που έχουν υλοποιηθεί μέχρι αυτή τη στιγμή το

Compilation Report (Σχήμα 5.5) δίνει μια συνολική εικόνα για τους πόρους του συστήματος. Τα λογικά στοιχεία που χρησιμοποιούνται δεν ξεπερνούν το 1%. Στην αναφορά φαίνονται και οι τέσσερις πολλαπλασιαστές που χρησιμοποιούνται για τον υπολογισμό των Bin.

Προφανώς ο έλεγχος των δεδομένων με το χέρι ή με υπολογιστή τσέπης, ακόμα και για μικρογραφίες εικόνων, είναι εξαιρετικά δύσκολη διαδικασία. Από την υλοποίηση που έχει προηγηθεί στο Matlab και παραμετροποιώντας κατάλληλα τον κώδικα μπορεί να γίνει πολύ πιο εύκολη η διαδικασία ελέγχου. Απαραίτητη προϋπόθεση είναι η εικόνα που θα δημιουργηθεί στο Matlab για να ελεγχθεί η υλοποίησή του κυκλώματος VHDL, να είναι ακριβώς ίδια με αυτή που θα εισαχθεί στις κυμματομορφές του Quartus. Μόνο όταν υπάρξει βεβαιότητα για την ορθότητα των αποτελεσμάτων μπορεί να ξεκινήσει το επόμενο στάδιο της υλοποίησης. Σε διαφορετική περίπτωση είναι πολύ πιθανό να ξεκινήσει ένα “ντόμινο” λαθών που κάνει σχεδόν αδύνατη την αποσφαλμάτωση.

## 5.5 Δημιουργία Ιστογραμμάτων

### 5.5.1 Γενική λειτουργία υποκυκλώματος

Από το προηγούμενο κύκλωμα τέσσερα σήματα αποτέλεσαν τις εξόδους: Τρία για τα δεδομένα  $G_x$ ,  $G_y$ , bin και ένα σήμα En που σηματοδοτεί την αποστολή δεδομένων. Τα σήματα αυτά εισέρχονται στο επόμενο υποκύκλωμα το οποίο δημιουργεί τα ιστογράμματα. Ο υπολογισμός των τιμών των ιστογραμμάτων γίνεται για κάθε cell μεγέθους  $8 \times 8$  εικονοστοιχείων. Αφού όμως τα δεδομένα έρχονται ανά σειρά, για να τελειώσει ο υπολογισμός του πρώτου ιστογράμματος πρέπει να έχει ληφθεί το όγδοο bin της όγδοης σειράς από την προηγούμενη υπομονάδα.

Συνοπτικά η διαδικασία έχει ως εξής: εισέρχονται στο υποκύκλωμα τα οχτώ πρώτα bin και με αυτά δημιουργείται ένα προσωρινό ιστόγραμμα το οποίο αντιστοιχεί στο πρώτο cell. Με την είσοδο του ένατου δεδομένου το προσωρινό ιστόγραμμα αποθηκεύεται στη μνήμη. Μόλις εισαχθεί το ένατο δεδομένο αρχίζει να υπολογίζεται το δεύτερο προσωρινό ιστόγραμμα του δεύτερου cell. Μετά την λήψη και του δεκάτου έκτου δεδομένου

αποθηκεύεται το δεύτερο προσωρινό ιστόγραμμα και συνεχίζεται η διαδικασία μέχρι το τέλος της γραμμής.

Όσο γίνεται η λήψη της επόμενης γραμμής καλούνται από τη μνήμη τα προσωρινά ιστογράμματα με τη σειρά που αποθηκεύτηκαν και συνεχίζεται η επεξεργασία τους. Αυτή η διαδικασία επαναλαμβάνεται μέχρι και την όγδοη σειρά με την οποία ολοκληρώνονται τα πρώτα ιστογράμματα και αποθηκεύονται στη μνήμη. Η διαδικασία επαναλαμβάνεται από την αρχή για κάθε επόμενη γραμμή από cell.

Το bin δείχνει κάθε φορά σε ποια θέση του ιστογράμματος πρέπει να προστεθούν τα  $|Gy|$  και  $|Gx|$ . Αυτό το σκεπτικό προέκυψε η δημιουργία συστοιχίας εννέα καταχωρητών. Οι καταχωρητές αυτοί θα περιέχουν τις τιμές κάθε θέσης του ιστογράμματος που δημιουργείται. Όταν έρχεται μια τιμή  $|Gx| + |Gy|$ , πάει και προστίθεται σε εκείνο τον καταχωρητή που υποδεικνύεται από την αντίστοιχη τιμή του Bin. Η ανάγκη που υπάρχει να αποθηκεύονται και να ανακαλούνται οι τιμές των καταχωρητών αλλά και να αρχικοποιούνται σε κάποιες δεδομένες χρονικές στιγμές, χωρίς να διακόπτεται η διαδικασία δημιουργίας ιστογραμμάτων, καθιστά αναγκαία τη χρήση και δεύτερης συστοιχίας καταχωρητών. Έτσι όταν η μια συστοιχία υπολογίζει τα ιστογράμματα η άλλη επικοινωνεί με τη μνήμη για την αποθήκευση ιστογραμμάτων ή τη λήψη ιστογραμμάτων με σκοπό την αρχικοποίηση της.

### 5.5.2 Δεδομένα και μνήμη

Πριν ξεκινήσει ο σχεδιασμός των κυκλωμάτων ήταν απαραίτητη μια μελέτη πάνω στο μέγεθος των δεδομένων που θα επεξεργαστούν τα υποκυκλώματα. Στην προκείμενη περίπτωση τα δεδομένα που υπόκεινται σε επεξεργασία είναι το άθροισμα δύο αριθμών των εφτά bit, δηλαδή ένας αριθμός των οχτώ bit. Για κάθε ιστόγραμμα γνωρίζαμε ότι έπρεπε να αθροιστούν εξήντα τέσσερις αριθμοί των οχτώ bit. Στην σπάνια λοιπόν περίπτωση όπου όλα τα αθροίσματα  $|Gx| + |Gy|$  θα είχαν τη μέγιστη τιμή, δηλαδή 256 και όλα τα μέτρα θα αντιστοιχούσαν σε μια θέση του ιστογράμματος τότε η τιμή που θα προέκυπτε για την αναπαράσταση κάθε θέσης, θα ήταν:

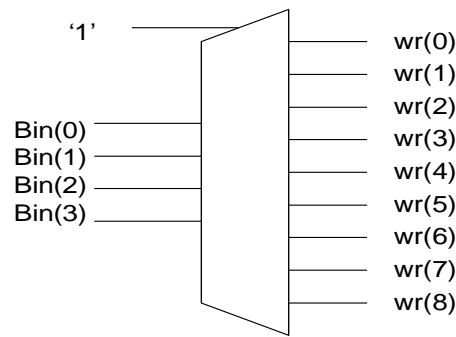
$$N = \log_2 (256 * 64) = \log_2 (16.384) = 14\text{bit}$$

Αν για κάθε θέση του ιστογράμματος πρέπει να δεσμευτεί θέση μνήμης των δεκατεσσάρων bit τότε για κάθε ιστόγραμμα απαιτούνται  $14 * 9 = 126$  bit. Για την εφαρμογή της εργασίας αυτής αποφασίστηκε κάθε ιστόγραμμα να καταλαμβάνει μια θέση μνήμης μεγέθους 126 bit.

Το συνολικό μέγεθος της μνήμης πρέπει είναι ίσο με το πλήθος των cell επί το πλήθος των bit που χρειάζεται για να αναπαρασταθεί κάθε cell. Η εικόνα που μπορεί να επεξεργαστεί η εφαρμογή μας είναι μεγέθους  $640 \times 480$  εικονοστοιχεία δηλαδή  $80 \times 60 = 4800$  cell. Κατά το σχεδιασμό αποφασίστηκε να μην συμπεριλαμβάνονται τα cell των άκρων της εικόνας στην επεξεργασία οπότε το πλήθος των cell διαμορφώνεται ως εξής  $78 \times 58 = 4524$ . Ακόμα και αν κάποια στιγμή σε μελλοντική χρήση αποφασιστεί να περικοπούν περισσότερα cell από τα άκρα της εικόνας το μέγεθος αυτό θα καλύπτει τις ανάγκες για μνήμη.

### 5.5.3 Συστοιχίες Καταχωρητών

Όπως προαναφέρθηκε η τιμή του κάθε Bin δείχνει σε ποιον καταχωρητή, από την συστοιχία που δημιουργεί εκείνη τη στιγμή το ιστόγραμμα, θα προστεθούν τα Gx και Gy. Την ίδια στιγμή η άλλη συστοιχία, πρέπει να ενεργοποιείται από ένα σήμα εισόδου (wr\_init) που θα πραγματοποιεί την αρχικοποίηση των τιμών των καταχωρητών. Το wr\_init αρχικοποιεί τους καταχωρητές της συστοιχίας για να είναι έτοιμοι να ξεκινήσουν τους υπολογισμούς μόλις έρθει η κατάλληλη χρονική στιγμή. Αυτό σημαίνει ότι η λειτουργία κάθε συστοιχίας καθορίζεται από ένα κύκλωμα το οποίο έχει ως εισόδους έναν αποπολυπλέκτη, το wr\_init και ένα σήμα sl που καθορίζει τι κάνει η κάθε συστοιχία. Ο αποπολυπλέκτης έχει σαν είσοδο την τιμή του Bin, δηλαδή τέσσερα bit, και έξοδο εννέα bit, wr(0-8).



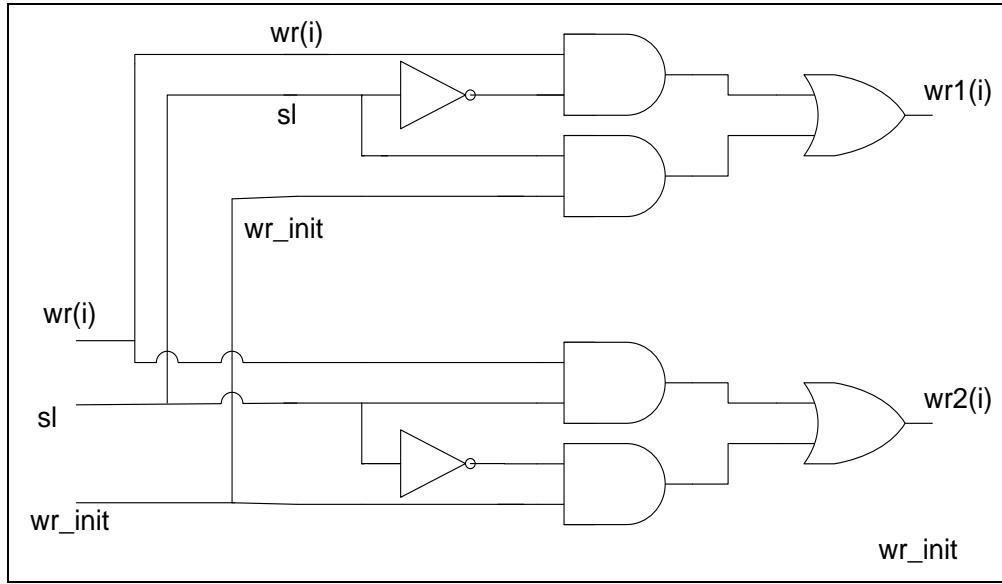
### Σχήμα 5.6 Αποπολυπλεξία του bin

Ο λογικός πίνακας για τα παραπάνω σήματα που καθορίζουν τον τρόπο λειτουργίας των συστοιχιών είναι ο ακόλουθος:

SI	wr_init	Wr(i)	WR1(i)	WR2(i)
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

**Πίνακας 5.1** Πίνακας αλήθειας για τα σήματα ενεργοποίησης των καταχωρητών των Συστάδων Υπολογισμού Ιστογραμμάτων.

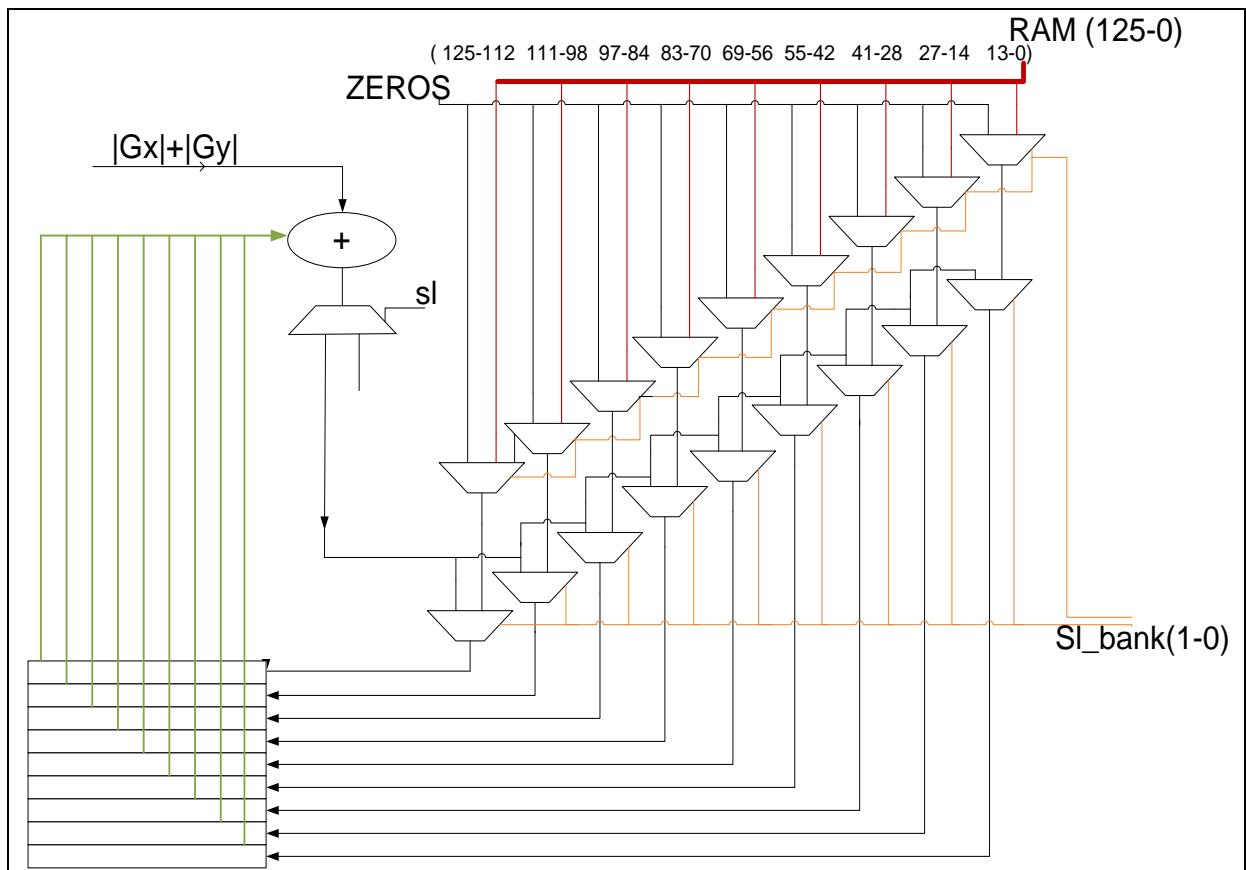
Από τον Πίνακα 5.1 προκύπτει το λογικό κύκλωμα του σχήματος 5.7



**Σχήμα 5.7** Λογικό κύκλωμα που αντιστοιχεί στον Πίνακα 5.1

Από τα σήματα ελέγχου και ενεργοποίησης των καταχωρητών καθορίζεται ποια θα είναι τα δεδομένα στις εισόδους τους. Κάθε καταχωρητής πρέπει να μπορεί να δεχτεί τριών ειδών δεδομένα: τον αθροιστή που αθροίζει την προηγούμενη τιμή του καταχωρητή με τη είσοδο  $|Gx| + |Gy|$ , τη μνήμη RAM για να μπορεί να συνεχίσει την δημιουργία ιστογράμματος και τέλος την τιμή “0” για τις περιπτώσεις που ξεκινάει νέο ιστόγραμμα.

Πριν η κάθε συστοιχία καταχωρητών ξεκινήσει τη διαδικασία υπολογισμού ιστογράμματος, πρέπει να αρχικοποιηθεί. Η αρχικοποίηση γίνεται με μηδενικές τιμές όταν η γραμμή που θα εισέλθει είναι η πρώτη του τρέχοντος cell και με δεδομένα από τη μνήμη σε οποιαδήποτε άλλη περίπτωση. Για την επιλογή της εισόδου γίνεται χρήση δύο διαδοχικών πολυπλεκτών (Σχήμα 5.8)



**Σχήμα 5.8** Είσοδοι στις συστοιχίες καταχωρητών

Το σήμα  $SI_{bank}$  είναι αυτό που ελέγχει τους πολυπλέκτες και καθορίζει ποια θα είναι η είσοδος. Το υποκύκλωμα του σχήματος 5.8 υπάρχει δύο φορές στο κύκλωμα, μια για κάθε συστοιχία καταχωρητών. Η διαφορά μεταξύ τους είναι η αντιστροφή του σήματος  $sl_{bank}$ , έτσι ώστε να έχουν αντίστροφους τρόπους λειτουργίας αφού ποτέ η ίδια τιμή, από όπου και να προέρχεται, δεν πάει ταυτόχρονα και στις δύο συστοιχίες. Στον πίνακα 5.2 καταγράφονται οι λειτουργίες των συστοιχιών ανάλογα με τα σήματα ελέγχου. Από αυτούς του συνδυασμούς επιλέχθηκαν οι καταστάσεις που εξυπηρετούσαν την λειτουργία του κυκλώματος αφού μόνο σε αυτές τις περιπτώσεις οι δύο συστοιχίες κάνουν ταυτόχρονα τις διεργασίες που πρέπει. Οι υπόλοιπες καταστάσεις δεν παρουσιάζονται ποτέ κατά τη λειτουργία του κυκλώματος.

Sl	Wr_init	Sl_bank	Συστοιχία 1	Συστοιχία 2
0	0	0 0	αθροίσματα	X
0	0	0 1	αθροίσματα	X
0	0	1 0	αρχικοποίηση	X
0	0	1 1	αρχικοποίηση	X
0	1	0 0	αθροίσματα	Others =>'0'
0	1	0 1	αθροίσματα	RAM
0	1	1 0	αρχικοποίηση	Others =>'0'
0	1	1 1	αρχικοποίηση	RAM
1	0	0 0	X	αρχικοποίηση
1	0	0 1	X	αρχικοποίηση
1	0	1 0	X	αθροίσματα
1	0	1 1	X	αθροίσματα
1	1	0 0	RAM	αρχικοποίηση
1	1	0 1	Others =>'0'	αρχικοποίηση
1	1	1 0	RAM	αθροίσματα
1	1	1 1	Others =>'0'	αθροίσματα

**Πίνακας 5.2** Με κίτρινο χρώμα έχουν γραμμοσκιαστεί οι καταστάσεις που χρησιμοποιήθηκαν από την μονάδα ελέγχου.

Ο έλεγχος των σημάτων ελέγχου (sl, wr\_init, sl\_bank) και των σημάτων εγγραφής στη μνήμη, δηλαδή των σημάτων που καθορίζουν τις λειτουργίες των συστοιχιών και την συνεργασία τους με τη μνήμη, γίνεται από μια μονάδα ελέγχου. Η μονάδα ελέγχου είναι ένα σύνολο από συγκριτές, μετρητές και καταχωρητές. Οι μετρητές είναι τέσσερις. Μόλις μηδενίσει ένας μετρητής αυξάνει κατά ένα τον επόμενο του, με τη σειρά που αναγράφονται παρακάτω:

1. Μετράει δεδομένα που εισέρχονται ανά ένα και μέχρι οχτώ. Έτσι σηματοδοτείτε η αλλαγή του cell σε οριζόντια κατεύθυνση
2. Μετράει οχτάδες δεδομένων, δηλαδή cell και φτάνει μέχρι το πλήθος των cell, οπότε σηματοδοτεί την αλλαγή γραμμής.
3. Μετράει μέχρι οχτώ γραμμές, για να σηματοδοτεί την τελευταία γραμμή του κάθε cell.

4. Μετράει οχτάδες γραμμών μέχρι το πλήθος των cellτης στήλης. Σηματοδοτεί το τέλος του καρέ.

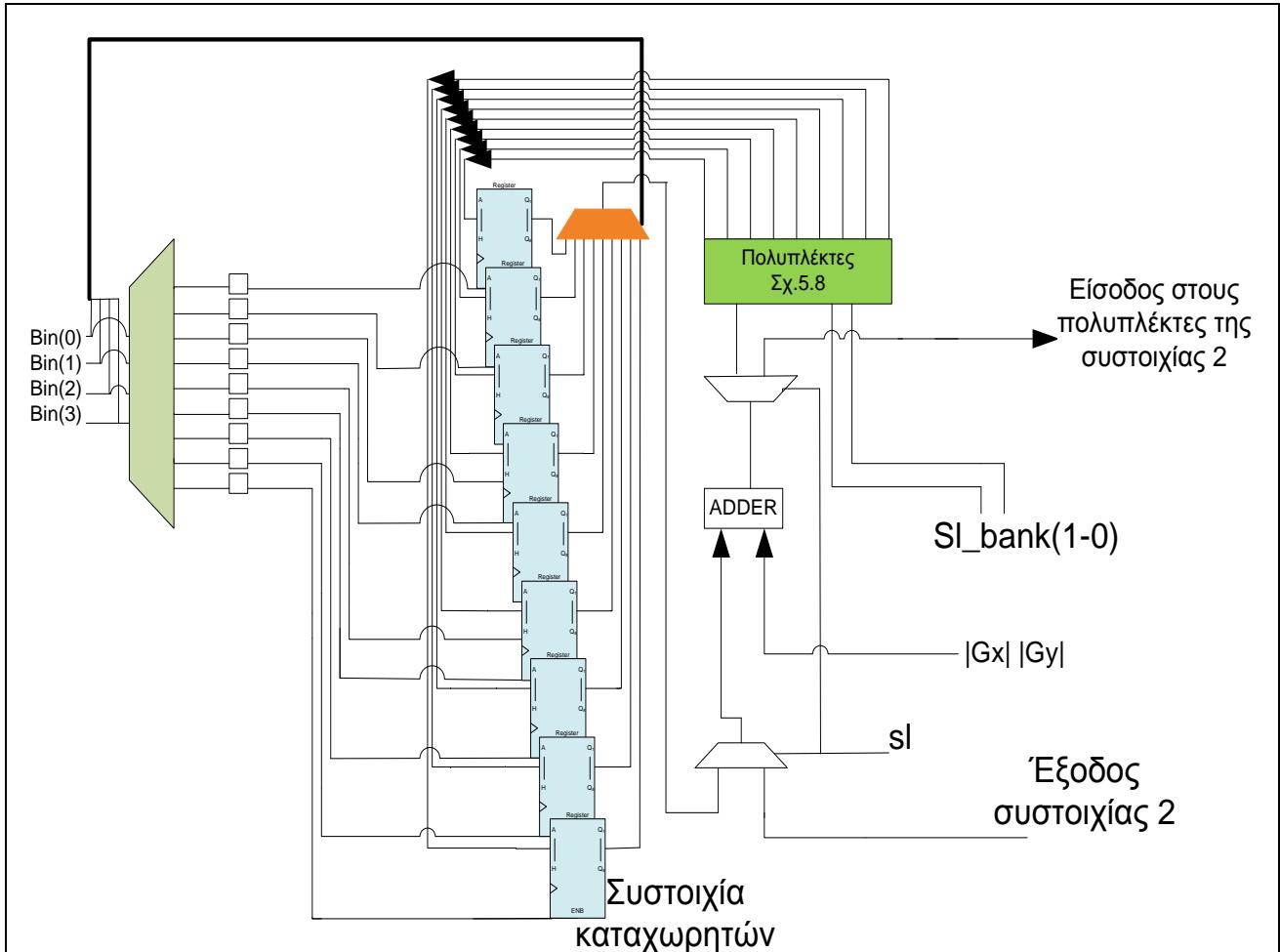
Για τη δημιουργία των σημάτων ελέγχου έγινε πρώτα η εισαγωγή τους σαν με το χέρι σήματα εισόδου. Όταν επιτεύχθηκε η ορθή λειτουργία του υποκυκλώματος ξεκίνησε η υλοποίηση της μονάδας ελέγχου, χρησιμοποιώντας τα σήματα εισόδου σαν σημεία αναφοράς. Η υλοποίηση της μονάδας ελέγχου, αν και είναι χρονοβόρα και σύνθετη δεν περιέχει κάποια ιδιαίτερη δυσκολία και αυτό είναι ο λόγος που δεν γίνεται εκτενής αναφορά σε αυτή.

#### **5.5.4 Άθροιση Μέτρων**

Μετά την αρχικοποίηση των καταχωρητών, τα μέτρα που εισέρχονται από το προηγούμενο υποκύκλωμα πρέπει να προστεθούν στην υπάρχουσα τιμή που περιέχει εκείνος ο καταχωρητής που “δείχνει” η τιμή του bin. Η πράξη της πρόσθεσης γίνεται ασύγχρονα. Το κομμάτι εκείνο του υποκυκλώματος που πραγματοποιεί την πρόσθεση φαίνεται στο Σχήμα 5.9.

Στην είσοδο και την έξοδο του αθροιστή υπάρχουν ένας πολυπλέκτης και ένας αποπολυπλέκτης αντίστοιχα, οι οποίοι ανάλογα με την τιμή του σήματος -sl- καθορίζουν με ποια με συστοιχία θα συνεργαστεί ο αθροιστής.

Ο αποπολυπλέκτης της τιμής του bin, ενεργοποιεί τον κατάλληλο καταχωρητή, από τον οποίο εξέρχεται μια τιμή η οποία θα αποτελέσει τη μια είσοδο του αθροιστή. Η άλλη είσοδος του αθροιστή είναι, όπως έχει ήδη, το  $|Gx| + |Gy|$ . Μόλις γίνει η πρόσθεση η τιμή επιστρέφεται στον καταχωρητή μέσω του κυκλώματος του σχήματος 5.8 .



**Σχήμα 5.9** Η είσοδος του adder είναι η έξοδος ενός αποπολυπλέκτη. Ο αποπολυπλέκτης του εισάγει στον adder την τιμή εκείνου του καταχωρητή που δείχνει το bin. Μετά την πρόσθεση η τιμή διαμέσου του κυκλώματος του σχ. 5.8, εισέρχεται στον ίδιο καταχωρητή ο οποίος ενεργοποιείται από τον αποπολυπλέκτη (πράσινο χρώμα)

## 5.6 Ανάγνωση μνήμης

Μόλις ολοκληρωθεί ο υπολογισμός των ιστογραμμάτων και αποθηκευθούν στη μνήμη RAM, τότε μπορεί να ξεκινήσει η διαδικασία της ταξινόμησης. Για να γίνει η ταξινόμηση πρέπει να υπάρχει η δυνατότητα να διαβαστούν τα ιστογράμματα με σειρά τέτοια ώστε να αντιστοιχούν σε κυλιόμενο παράθυρο εντοπισμού. Όπως περιγράφεται αναλυτικά στο δεύτερο κεφάλαιο, ο ταξινομητής επεξεργάζεται ένα κομμάτι της εικόνας μεγέθους ίσου με το παράθυρο εντοπισμού και μετά σύρεται δεξιά.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95

**Σχήμα 5.10** Μικρογραφία εικόνας. Κάθε τετράγωνο αντιστοιχεί σε ένα cell και οι αριθμοί αντιστοιχούν στις θέσεις μνήμης που θα αποθηκευτούν οι τιμές του ιστογράμματος. Τα περιγράμματα αναπαριστούν παράθυρα εντοπισμού

Στο προηγούμενο σχήμα (Σχήμα 5.10) παρουσιάζεται μια εικόνα μεγέθους  $8 \times 12$  cell που θα βοηθήσει στην περιγραφή του τρόπου ανάγνωσης των ιστογραμμάτων από τη μνήμη. Το παράθυρο εντοπισμού στην συγκεκριμένη εικόνα είναι μεγέθους  $3 \times 5$  cell. Το υποκύκλωμα που υλοποιήθηκε είναι παραμετροποιήσιμο οπότε μπορεί να χρησιμοποιηθεί για εικόνες και παράθυρα εντοπισμού οποιουδήποτε μεγέθους. Κάθε τετραγωνάκι της εικόνα μπορεί να θεωρηθεί ως ένα cell και ο αριθμός που αναγράφεται ως η θέση μνήμης που αποθηκεύεται το αντίστοιχο ιστόγραμμα. Ξεκινώντας από τη θέση 0 και διαβάζοντας τόσες θέσεις μνήμης όσες είναι τα cell κατά μήκος της εικόνας εντοπισμού, δηλαδή τρεις, έχει διαβαστεί η πρώτη γραμμή. Για τη δεύτερη γραμμή γίνεται ακριβώς η ίδια διαδικασία αλλά η μνήμη που θα διαβαστεί θα είναι προσαυξημένη κατά το πλήθος των cell κατά μήκος όλης της εικόνας, δηλαδή 8.

Την πρώτη φορά δηλαδή διαβάστηκαν οι θέσεις 0, 1 και 2 ενώ τη δεύτερη οι 0+8, 1+8, 2+8. Για την τρίτη γραμμή θα προστεθεί άλλη μια φορά το πλήθος των cell γραμμής.

Δεδομένου ότι οι μετρητές ξεκινάνε από την τιμή μηδέν, η θέση της μνήμη προκύπτει από την τιμή του πρώτου μετρητή αν προστεθεί η τιμή του δεύτερου πολλαπλασιασμένη με το πλήθος των *cell* κατά μήκος της εικόνας.

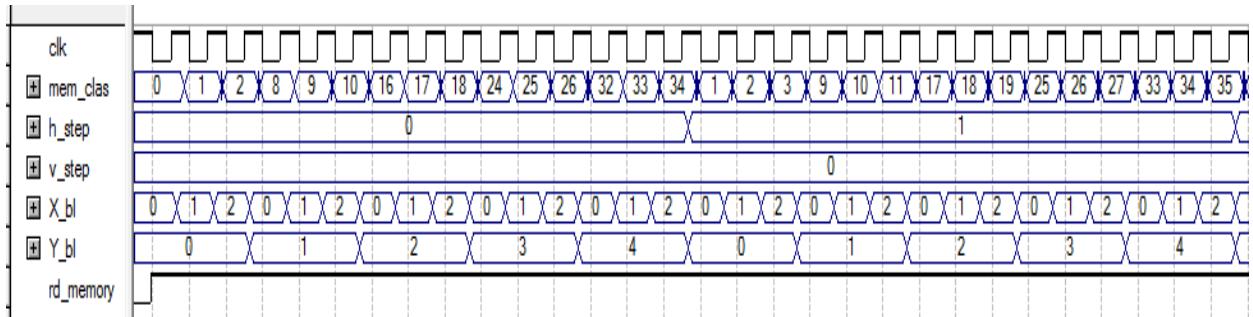
Η διαδικασία θα πραγματοποιηθεί τόσες φορές όσες τα *cell* που προκύπτουν για το ύψος του παραθύρου εντοπισμού. Μέχρι στιγμής έχει γίνει η ανάγνωση ενός παραθύρου εντοπισμού με τη χρήση δύο μετρητών. Ο ένας μετράει οριζόντια τα *cell* και ο άλλος μετρητής τα κάθετα και όταν μηδενίζει ο πρώτος αυξάνει κατά ένα ο δεύτερος.

Όταν ολοκληρωθεί η διαδικασία για το πρώτο παράθυρο εντοπισμού, πρέπει αυτό να μεταφερθεί μια θέση δεξιά, δηλαδή στο πράσινο πλαίσιο που διακρίνεται στο Σχήμα 5.9. Για να διαβαστούν τώρα αυτές οι θέσεις μνήμης, αρκεί να προστίθεται στον πρώτο μετρητή κάθε φορά το ένα. Για μια επιπλέον ολίσθηση θα πρέπει να προστίθεται το δύο και ούτω καθ' εξής. Εάν δηλαδή μπει στο υποκύλωμα άλλος ένας μετρητής ο οποίος μετράει το πλήθος των οριζόντιων ολισθήσεων και προστίθεται στην τελική τιμή τότε έχει συμπεριληφθεί και η ολίσθηση στον υπολογισμό της θέσης μνήμης. Ο μετρητής αυτός θα πρέπει να μηδενίζει όταν φτάσει την τιμή: πλήθος των *cell* της γραμμής όλης της εικόνας – το πλήθος των *cell* κατά μήκος του παραθύρου εντοπισμού. Στο παράδειγμα του σχήματος 5.9 είναι  $8 - 3 = 5$  ολισθήσεις. Κάθε φορά που μηδενίζει ο μετρητής των οριζόντιων ολισθήσεων πρέπει να γίνεται μια μετακύλιση του παραθύρου εντοπισμού προς τα κάτω. Άρα καθίσταται απαραίτητη η χρήση ενός ακόμη μετρητή, ο οποίος για κάθε κάθετη ολίσθηση που μετράει θα πρέπει να προσθέτει τόσες φορές το πλήθος των γραμμών για να προκύψει η σωστή θέση μνήμη.

Η τελική σχέση που προκύπτει και αποδίδει την θέση μνήμης κάθε χρονική στιγμή είναι η ακόλουθη:

$$\boxed{Mem = v\_step * H\_bl\_size + Y\_bl * H\_bl\_size + h\_step + X\_bl, \\ \text{όπου } v\_step \text{ είναι το πλήθος των κάθετων ολισθήσεων, } h\_step \text{ είναι} \\ \text{το πλήθος των οριζόντιων ολισθήσεων, } H\_bl\_size \text{ και } V\_bl\_size \text{ είναι το} \\ \text{μέγεθος της εικόνας σε } cell \text{ οριζόντια και κάθετα αντίστοιχα}}$$

Στο παρακάτω σχήμα φαίνεται η κυμματομορφή που προκύπτει από την υλοποίηση σε VHDL του κώδικα ανάγνωσης μνήμης. Η κυμματομορφή αφορά μεγέθη ίδια με του παραδείγματος που χρησιμοποιήθηκε, δηλαδή του σχήματος 5.10



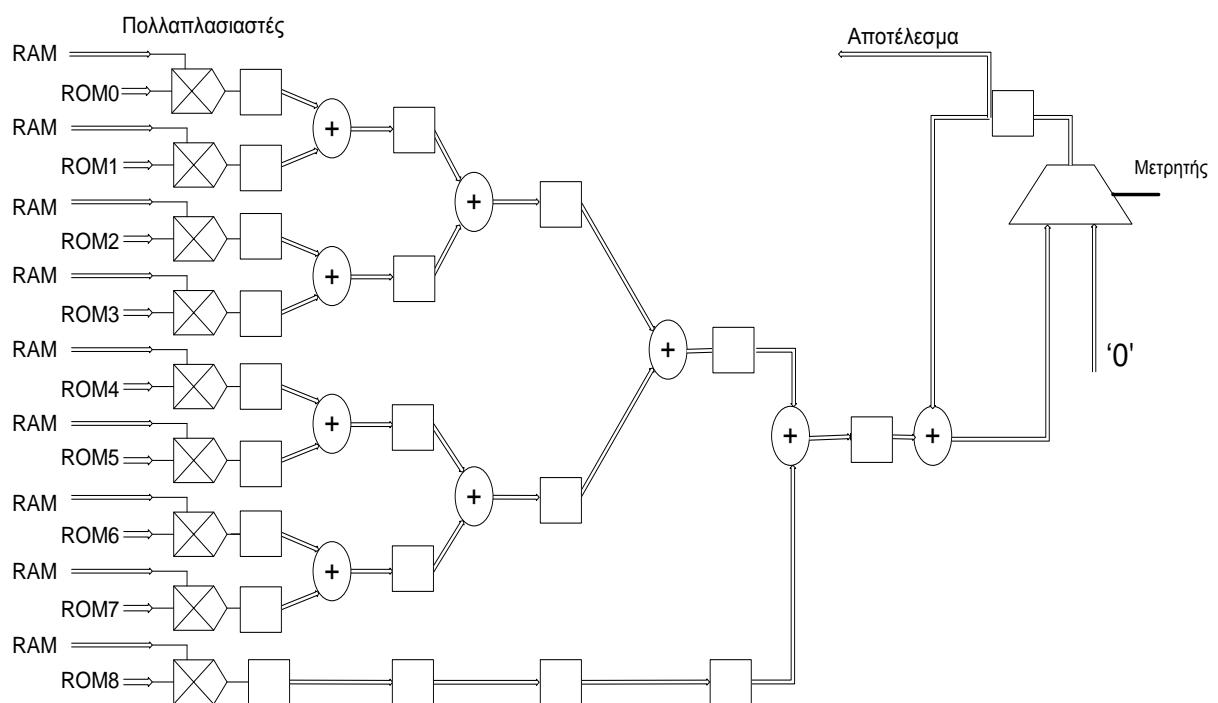
**Σχήμα 5.11** Οι θέσεις μνήμης με τη σειρά που πρέπει να διαβάζονται για την διαδικασία της ταξινόμησης

## 5.7 Κύκλωμα Ταξινόμησης

Όπως περιγράφηκε στο προηγούμενο κεφάλαιο η διαδικασία της ταξινόμησης γίνεται πολλαπλασιάζοντας τους συντελεστές που έχουν προκύψει από τη διαδικασία εκμάθησης του ταξινομητή, με τις αντίστοιχες τιμές του χαρακτηριστικού διανύσματος, δηλαδή τις τιμές των ιστογραμμάτων. Οι τιμές των ιστογραμμάτων διαβάζονται από την RAM από το υποκύκλωμα της προηγούμενης παραγράφου.

Οι τιμές από την εκμάθηση του ταξινομητή, δηλαδή τα βάρη, έχουν προκύψει από την διαδικασία που έγινε στο Matlab. Οι τιμές αυτές γράφτηκαν σε αρχεία κειμένου, με κώδικα που γράφτηκε στο Matlab, τα οποία αποτελούν αρχεία αρχικοποίησης μνημών ROM. Με το Mega Wizard Plugin Manager δημιουργήθηκαν εννέα μνήμες ROM χωρητικότητας 128 θέσεων των δώδεκα bit, οι οποίες αρχικοποιήθηκαν με τα προηγούμενα αρχεία. Επειδή τα βάρη όπως προκύπτουν από το Matlab είναι και αρνητικοί και θετικοί αριθμοί μικρότεροι της μονάδας, πολλαπλασιάστηκαν με  $2^{16} = 16.384$  και μετατράπηκαν σε αριθμούς Προσημασμένου Συμπληρώματος ως προς Δύο.

Σε κάθε μνήμη ROM οι τιμές με τα βάρη αποθηκεύτηκαν με τέτοιο τρόπο, ώστε τα δεδομένα να ανασύρονται παράλληλα από όλες τις μνήμες. Έτσι ο αριθμός που προσκομίζεται από τη ROM0 πολλαπλασιάζεται με τα πρώτα δεκατέσσερα bit της RAM, από τη ROM1 πολλαπλασιάζεται με τα επόμενα δεκατέσσερα bit της RAM και ούτω καθ' εξής. Το γινόμενο που προκύπτει είναι ένας αριθμός των είκοσι έξι bit. Πριν αθροιιστούν τα γινόμενα μεταξύ τους κόβονται τα δέκα λιγότερα σημαντικά ψηφία από κάθε αριθμό. Είναι δηλαδή σαν να διαιρούνται με το  $2^{10} = 1024$ . Αναγκαστικά προστίθενται μόνο δύο αριθμοί κάθε φορά, οπότε γίνεται χρήση παράλληλων αθροιστών οι οποίοι με τη σειρά τους τροφοδοτούν άλλους αθροιστές. Όταν εν τέλει έχουν αθροιστεί όλα τα γινόμενα μεταξύ τους το αποτέλεσμα μπαίνει σε ένα buffer και προστίθεται με τις τιμές που ακολουθούν. Οι τιμές που ακολουθούν έρχονται συνεχώς και ανά ένα παλμό λόγω του Pipeline. Οι προσθέσεις είναι τόσες όσες τα ιστογράμματα στο παράθυρο εντοπισμού. Τα αποτελέσματα από τις προσθέσεις διέρχονται από ένα πολυπλέκτη. Για το πρώτο ιστογράμμα κάθε παραθύρου εντοπισμού η έξοδος του πολυπλέκτη είναι μηδέν. Για όλα τα άλλα ιστογράμματα η έξοδος του πολυπλέκτη είναι η έξοδος του αθροιστή. Το κύκλωμα που υλοποιεί τον ταξινομητή είναι αυτό του σχήματος 5.12



**Σχήμα 5.12** Το κύκλωμα για την ταξινόμηση

Μόλις η συγκεκριμένη διαδικασία ολοκληρωθεί για όλα τα ιστογράμματα του παραθύρου εντοπισμού και αυτά προστεθούν μεταξύ τους, τότε προκύπτει ο εκθέτης της σχέσης:

$$T = \frac{1}{1 + e^{-\sum a * x}}$$

Όταν η τιμή του κλάσματος είναι μεγαλύτερη από την τιμή του κατωφλίου  $T$  τότε ο ταξινομητής “εντοπίζει” κάποιον πεζό στη θέση του παραθύρου εντοπισμού. Για την τελική απόφαση όμως δε χρειάζεται να υπολογιστεί όλο το κλάσμα. Μπορεί να μελετηθεί απλά για ποιές τιμές του εκθέτη το κλάσμα έχει τιμή μεγαλύτερη από αυτή του κατωφλίου. Ο υπολογισμός αυτός μπορεί να γίνει είτε από τη γραφική του παράσταση (γραφική παράσταση σιγμοειδούς συνάρτησης) είτε λύνοντας αλγεβρικά την παραπάνω σχέση, με το  $T$  να είναι γνωστό. Το αποτέλεσμα που προκύπτει λύνοντας αλγεβρικά είναι:

$$\sum a * x = \ln\left(\frac{T}{1-T}\right), \text{ όπου } T \text{ το κατώφλι απόφασης}$$

Στον υπολογισμό πρέπει να συμπεριληφθεί ο πολλαπλασιασμός στα βάρη και η μετέπειτα διαίρεση που έγινε στα γινόμενα των δεδομένων της ROM και της RAM. Αρχικά έχουν πολλαπλασιαστεί με το 16.384 και κάποια στιγμή στη συνέχεια διαιρέθηκαν με το 1024, οπότε το άθροισμα γινομένων πρέπει να υπολογίσουμε ότι θα είναι 16 φορές μεγαλύτερο απ' το πραγματικό. Για το κατώφλι έχουμε ορίσει στον κώδικα κάποιες σταθερές από τις οποίες επιλέγεται μια κάθε φορά, μέσω τέσσερις διακοπτών δύο θέσεων που βρίσκονται στο DE2-70.

Το τελευταίο υποκύκλωμα που έπρεπε να υλοποιηθεί είναι αυτό που αποθηκεύει σε μια μνήμη RAM τα πάνω αριστερά εικονοστοιχεία των περιοχών που εντοπίστηκε πεζός. Συνεπώς όταν το αποτέλεσμα των αθροισμάτων ολοκληρωθεί και ξεπεράσει την τιμή του κατωφλίου, πρέπει να γράφεται στη μνήμη η θέση των δύο εικονοστοιχείων που προαναφέρθηκαν. Γνωρίζοντας για ποιο παράθυρο εντοπισμού γίνεται η ταξινόμηση δεν είναι δύσκολο να υπολογιστεί η τιμή των εικονοστοιχείων. Το υποκύκλωμα του ταξινομητή

περιλαμβάνει ήδη ένα μετρητή ο οποίος μετράει τα ιστογράμματα του κάθε παραθύρου εντοπισμού, για να καλεί τις αντίστοιχες θέσει μνήμης από τη ROM. Κάθε φορά λοιπόν, που θα μηδενίζεται θα αυξάνει ένα μετρητή κατά ένα.

Με τον τελευταίο μετρητή της προηγούμενης παραγράφου, μετριούνται οι οριζόντιες ολισθήσεις. Κάθε οριζόντια ολίσθηση σημαίνει μετατόπιση κατά οχτώ εικονοστοιχεία. Επειδή όμως το οχτώ αποτελεί δύναμη του δύο αρκεί να συμπληρωθούν τρία μηδενικά στη θέση των λιγότερο σημαντικών ψηφίων και να προστεθεί ο αριθμός των ακραίων εικονοστοιχείων που δεν συμπεριλαμβάνονται στις επεξεργασίες. Ο μετρητής αυτός μηδενίζει όταν γίνει ίσος με: *το πλήθος των cell της γραμμής όλης της εικόνας – το πλήθος των cell κατά μήκος του παραθύρου εντοπισμού*, με το ίδιο σκεπτικό που χρησιμοποιήθηκε στην ανάγνωση της μνήμης. Όταν μηδενίσει ο προηγούμενος αυξάνει κατά ένα ο μετρητής των κάθετων μετατοπίσεων. Ο οποίος λειτουργεί όπως ακριβώς και ο προηγούμενος. Του προσαρτώνται δηλαδή, τρία μηδενικά ως λιγότερο σημαντικά ψηφία και στην τιμή του προστίθεται επίσης το πλήθος των ακραίων εικονοστοιχείων που δεν συμπεριλαμβάνονται στις επεξεργασίες.

Όλο υποκύκλωμα του ταξινομητή καθώς και τα κυκλώματα των μνημών ROM βρίσκονται στο παράρτημα προγραμμάτων.

## 5.8 Συνολικό κύκλωμα

Η δημιουργία ενός κυκλώματος από τα επί μέρους υποκυκλώματα έγινε με την εισαγωγή των υποκυκλωμάτων ως Components σε projecttης Altera. Η διασύνδεση των κυκλωμάτων έγινε με τη χρήση Portmapping και generic mapping για τις μεταβλητές τους.

Τα αποτελέσματα απέχουν από αυτά του Matlab και αυτό οφείλεται σε προβλήματα συγχρονισμού που προκύπτουν από τη σύνθεση του τελικού κυκλώματος. Επίσης παρουσιάζονται κάποια προβλήματα στην αποτελεσματικότητα της ταξινόμησης. Αυτό πιθανόν να οφείλεται στις στρογγυλοποιήσεις που αναγκαστικά γίνονται λόγω της περιορισμένου υλικού.

Μια συνοπτική ανάλυση των λογικών στοιχείων που χρησιμοποιεί το τελικό κύκλωμα φαίνεται στο παρακάτω σχήμα

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Thu Aug 22 07:57:16 2013
Quartus II Version	7.2 Build 203 02/05/2008 SP 2 SJ Full Version
Revision Name	DE2_70_TV
Top-level Entity Name	DE2_70_TV
Family	Cyclone II
Total logic elements	11,268
Total combinational functions	2,896
Dedicated logic registers	11,268
Total registers	11268
Total pins	562
Total virtual pins	0
Total memory bits	638,110
Embedded Multiplier 9-bit elements	43
Total PLLs	1

**Σχήμα 5.13** Συνοπτική ανάλυση κυκλώματος

## Αποτέλεσμα

Όλα τα υπόκυκλώματα που δημιουργήθηκαν λειτούργησαν όπως ακριβώς είχαν σχεδιαστεί. Οι κυματομορφές τους έχουν διατρεχτεί αμέτρητες φορές μέχρι να υπάρξει η βεβαιότητα ότι όλα είναι σωστά. Στην τελική «συναρμολόγηση» των υποκυκλωμάτων της εργασίας αυτής με το πρόγραμμα της Altera που κάνει τη διαχείριση εικόνων από το Video IN, κάτι δεν λειτουργεί κατά τα επιθυμητά. Επειδή η αποσφαλμάτωση μπορεί να έχει απρόβλεπτη διάρκεια και δεδομένου ότι η προσπάθεια και ο χρόνος που έχει καταβληθεί για την εκπόνηση της εργασίας αυτής σε συνδυασμό με τα μέχρι τώρα αποτελέσματα υπερκαλύπτουν τις ανάγκες μιας διπλωματικής εργασίας, αποφασίστηκε να γίνει η παρουσίασή της χωρίς την τελική εικόνα. Το σίγουρο είναι ότι αποτελεί πλέον προσωπικό στοιχημα τη ορθή λειτουργία του συστήματος και αργά ή γρήγορα αυτό θα γίνει. Πάραυτα οι περισσότεροι κώδικες υπάρχουν στα παραρτήματα της εργασία για όποιον θέλει να πειραματιστεί με το σύστημα ή να χρησιμοποιήσει μέρη από αυτό. Πληροφορίες και διευκρινίσεις δίνονται στο e-mail: antoge@upatras.gr.

## Συμπεράσματα – Προτάσεις

Η υλοποίηση σε FPGA οποιασδήποτε διαδικασίας που έχει μια σχετική υπολογιστική πολυπλοκότητα δεν είναι καθόλου εύκολη υπόθεση. Η σπαζοκεφαλιά της υλοποίησης ξεκινάει από την πλήρη κατανόηση του ζητούμενου. Μόνο όταν η διαδικασία έχει γίνει απόλυτα κατανοητή μπορεί να αρχίσει η υλοποίηση αφού, προσεγγίσεις, ασάφειες και διαισθητικές λύσεις δεν έχουν θέση στα FPGA. Εκτός όμως από την σε βάθος κατανόηση που είναι απαραίτητη, μια υλοποίηση χρειάζεται να της αφιερώσει κάποιος αρκετό χρόνο. Χρόνο για να μελετήσει τις προδιαγραφές, να κάνει τις εξομοιώσεις στο Matlab, να δοκιμάσει απλοποιήσεις που επιβάλλονται ή προσφέρονται και στο τέλος να ξεκινήσει την υλοποίηση.

Η υλοποίηση σε FPGA, των Ιστογραμμάτων Προσανατολισμένων Βαθμώσεων με τις βέλτιστες παραμέτρους όπως τις έχουν προτείνει οι Dalal & Triggs [4] είναι μιας τεράστιας δυσκολίας πρόκληση, με τα σημερινά δεδομένα. Η χρήση SVM, οι τετραγωνικές ρίζες, η ευκλείδεια Νόρμα, η κανονικοποίηση σε επίπεδο block αυξάνουν πάρα πολύ την υπολογιστική πολυπλοκότητα.

Στην υλοποίηση της παρούσας εργασίας έγιναν αρκετές απλοποιήσεις όπως η αντικατάσταση των μέτρων με απόλυτες τιμές, η κβάντιση των δεδομένων, η απόρριψη της κανονικοποίησης σε επίπεδο μπλοκ και τέλος η αλλαγή του ταξινομητή. Κάποιες από τις προηγούμενες αλλαγές θα μπορούσαν να αποφευχθούν αν ενισχύονταν οι παράλληλες διεργασίες έτσι ώστε να εξικονομηθεί χώρος στη μνήμη αλλά και να μπορεί να γίνεται η ταξινόμηση ταυτόχρονα με τις άλλες διεργασίες. Η εξέλιξη των αριθμητικών μεθόδων και η ανάπτυξη ταξινομητών σε FPGA θα αποτελέσουν σίγουρα εφαλτήριο για ακόμη ταχύτερες εφαρμογές επεξεργασίας εικόνων και βίντεο σε πραγματικό χρόνο.

Όσο δουλευόταν αυτή η εργασία πολλές ιδέες για βελτιώσεις ή τροποποιήσεις παρουσιάστηκαν. Πολλές από αυτές περιφέρονται γύρω από την παράλληλη επεξεργασία. Η πολυπλοκότητα της πλήρους μορφής του περιγραφέα θα μπορούσε να ξεπεραστεί με δύο ίδια κυκλώματα διαμορφωμένα έτσι ώστε να αξιοποιούνται τα λογικά στοιχεία του FPGA που δεν χρησιμοποιούνται ή με δύο διαμορφωμένα FPGAs που θα μοιράζονται το

φόρτο εργασίας. Επίσης δοκιμές πάνω στις παραμέτρους για τον υπολογισμό του περιγραφέα και στο είδος του ταξινομητή ήταν και είναι στο επίκεντρο του ενδιαφέροντος όσων ασχολούνται με το θέμα αυτό. Τέλος μια πιο θεωρητική μελέτη για το που είναι αποδοτικότερο να αφιερώνονται περισσότεροι πόροι ενός FPGA: στην εξαγωγή των χαρακτηριστικών ή στην ταξινόμηση?



# Βιβλιογραφία

- [1]: Ryoji Kadota, Hiroki Sugano, Masayuki Hiromoto, Hiroyuki Ochi , “Hardware Architecture for HOG Feature Extraction”, Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, pp1330-1333, 2009.
- [2]: Tam Phuong Cao, Guang Deng, Mulligan David, “Implementation of real-time pedestrian detection on FPGA”, Image and Vision Computing, 2008.
- [3]: Sebastian Bauer, Ulrich Brunsmann και Stefan Schlotterbeck - Macht, “FPGA Implementation of a HOG-based Pedestrian Recognition System”, MPC-Workshop July 2009.
- [4]: Navneet Dalal and Bill Triggs, “Histograms of Oriented Gradients for Human Detection”, 2005.
- [5] Piotr Dollar, Christian Wojek, Bernt Schiele, Pietro Perona, “Pedestrian Detection: An Evaluation of the State of the Art IEEE Transactions on Pattern Analysis and Machine, pp.10 2012.
- [6]: David Hogg, “Model-based vision: a program to see a walking person”, Vol.1, pp. 5 – 20, 1983.
- [7]: Michael Oren, Constantine Papageorgiou, Pawan Sinha, Edgar Osuna, Tomaso Poggio “Pedestrian Detection Using Wavelet Templates”, CPVR Puerto Rico, 1997.
- [8]: Paul Viola, Michael Jones, “Robust Real-time Object Detection”, Second International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning ad Sampling, 2001.
- [9]: Bastian Leibe, Ales Leonardis, Bernt Schiele, “Combined Object Categorization and Segmentation with an Implicit Shape Model”, ECCV’04 Workshop on Statistical Learning in Computer Vision, Prague, 2004.
- [10]: Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio, “Example-Based Object Detection in Images by Components”, IEEE Transactions on Pattern Analysis and Machine, VOL. 23, NO. 4, 2001.
- [11]: W. Freeman and M. Roth, “Orientation histograms for hand gesture recognition”, I EEE Intl. Wkshp. on Automatic Face and Gesture Recognition, 1995.



# Παράρτημα Προγραμμάτων Matlab

## Κώδικας Υπολογισμού HOG στην πλήρη του εκδοχή

```

function [bin_hist_normed]=my_hog2(I) % single eikona 64 x 128
% arhikopoiiseis pinakon
Gx_Im=single(zeros(128,64));
Gy_Im=single(zeros(128,64));
Mag_Im=single(zeros(128,64));
bin_Im=zeros(128,64);
bin_Im2=zeros(128,64);
tan_Im=double(zeros(128,64));
weights=zeros(16,8);
bin_hist=zeros(16,8,9);
bin_hist_normed=zeros(16,8,36);

bins=9;
cellsize=8;
block_size= 4;
[C_size,L_size]=size(I);

% 1-> gamma normalization- Optional
% 2-> Compute Gradient
for i= 2:C_size-1
    for j= 2:L_size-1
        Gx_Im(i,j) = I(i,j+1)-I(i,j-1);
    end
end
for i= 2:C_size-1
    for j= 2:L_size-1
        Gy_Im(i,j) = I(i+1,j)-I(i-1,j);
    end
end

Gx_Im(1,:)= Gx_Im(2,:);
Gx_Im(C_size,:)= Gx_Im(C_size-1,:);
Gx_Im(:,1)= Gx_Im(:,2);
Gx_Im(:,L_size)= Gx_Im(:,L_size-1);

Gy_Im(1,:)= Gy_Im(2,:);
Gy_Im(C_size,:)= Gy_Im(C_size-1,:);
Gy_Im(:,1)= Gy_Im(:,2);
Gy_Im(:,L_size)= Gy_Im(:,L_size-1);

% Weighted Vote (magnitude) into spatial & orientation Bins
for i=1:C_size
    for j=1:L_size
        Mag_Im(i,j)=((Gx_Im(i,j)*Gx_Im(i,j)) + (Gy_Im(i,j)*Gy_Im(i,j)))^(1/2);
    end
end

% ipologizo klisi k dimiourgo pinaka me antistoiha bin (1-9)
for i=1:C_size
    for j=1:L_size

        hist_line = floor((i-1)/8)+1;
        hist_col = floor((j-1)/8)+1;

        tan_Im(i,j)= Gy_Im(i,j) ./ Gx_Im(i,j);

        if Gy_Im(i,j)== 0 && Gx_Im(i,j)> 0;
            bin_Im(i,j)=1;
        elseif Gy_Im(i,j)== 0 && Gx_Im(i,j)< 0;
            bin_Im(i,j)=9;
        else
            bin_Im(i,j)=mod((hist_line-1)*8+hist_col,9);
        end
    end
end

```

```

elseif Gy_Im(i,j) ~= 0 && Gx_Im(i,j) == 0;
    bin_Im(i,j)=5;
elseif (Gy_Im(i,j)>0 && Gx_Im(i,j)>0) || (Gy_Im(i,j)<0 && Gx_Im(i,j)<0)
    if tan_Im(i,j)<0.364
        bin_Im(i,j)=1;
    elseif tan_Im(i,j)<.8391
        bin_Im(i,j)=2;
    elseif tan_Im(i,j)<1.7321
        bin_Im(i,j)=3;
    elseif tan_Im(i,j)<5.6713
        bin_Im(i,j)=4;
    else bin_Im(i,j)=5;
    end
elseif (Gy_Im(i,j)>0 && Gx_Im(i,j)<0) || (Gy_Im(i,j)<0 && Gx_Im(i,j)>0)
    if tan_Im(i,j)>-0.364
        bin_Im(i,j)=9;
    elseif tan_Im(i,j)>-.8391
        bin_Im(i,j)=8;
    elseif tan_Im(i,j)>-1.7321
        bin_Im(i,j)=7;
    elseif tan_Im(i,j)>-5.6713
        bin_Im(i,j)=6;
    else bin_Im(i,j)=5;
    end
end
bin=bin_Im(i,j);
if bin~=0;
    bin_hist(hist_line,hist_col,bin)= bin_hist(hist_line,hist_col,bin)+
Mag_Im(i,j);
    end
end
C = cat(2, bin_hist(:,:,1,:), bin_hist(:,:,2,:));
C = cat(1, C(:,:,1), C(:,:,2), C(:,:,3));
bin_hist2=C;
for i=2:17
    for j=2:9
        for k=1:9
            bin_hist_normed(i-1,j-1,k)
            =bin_hist2(i,j,k)/(sqrt((bin_hist2(i,j,k).^2)+(bin_hist2(i-
1,j,k).^2)+(bin_hist2(i,j-1,k).^2)+(bin_hist2(i-1,j-1,k).^2))+0.2);
            bin_hist_normed(i-1,j-1,k+9)
            =bin_hist2(i,j,k)/(sqrt((bin_hist2(i,j,k).^2)+(bin_hist2(i,j,k).^2)+(bin_hist2(i,j+1,k).^2)+(bin_hist2(i,j+1,k).^2))+0.2);
            bin_hist_normed(i-1,j-
1,k+18)=bin_hist2(i,j,k)/(sqrt((bin_hist2(i,j,k).^2)+(bin_hist2(i,j+1,k).^2)+(bin_h
ist2(i+1,j+1,k).^2)+(bin_hist2(i+1,j,k).^2))+0.2);
            bin_hist_normed(i-1,j-
1,k+27)=bin_hist2(i,j,k)/(sqrt((bin_hist2(i,j,k).^2)+(bin_hist2(i+1,j,k).^2)+(bin_h
ist2(i+1,j-1,k).^2)+(bin_hist2(i,j-1,k).^2))+0.2);
        end
    end
end

```

## Εντοπισμός πεζών σε VGA Εικόνα

```

Im=imread('C:\Users\antoge\Documents\hep\THESSIS_FPGA\pedestrians128x64.tar\INRIAP
erson\Train\pos\person_and_bike_129.png');
Im_or=Im;
I=rgb2gray(Im);
I=single(I);

point =0;
point2=0;
%for scale=1:1
%    I=I(1:scale:end, 1:scale:end);

```

```

[ipsos,platos]=size(I);

for i=1:16:ipsos-127
    for j=1:16:platos-63
        det_pic= I(i:i+127, j:j+63);
        hog_vector = my_hog_quant_nb_noNAN(det_pic);
        [x,y,z]=size(hog);
        n=1;
        for k= 1:(x)
            for l= 1:(y)
                for m=1:(z)
                    hog_vector(1,n)=hog(k,l,m);
                    n=n+1;
                end
            end
        end

        test_data(1,:)=hog_vector;

        group=round(glmval(glm_mdel, test_data, 'logit'));
        if group ==1
            point=point+1;
            coord1(1,point)=j;
            coord1(2,point)=i;
        end

        end
    end
%end

for i=1:point
    for vert=1:128
        Im_or(coord1(2,i)+vert, coord1(1,i),1)= 153;
        Im_or(coord1(2,i)+vert, coord1(1,i),2)= 255;
        Im_or(coord1(2,i)+vert, coord1(1,i),3)= 0;
        Im_or(coord1(2,i)+vert, coord1(1,i)+64,1)= 153;
        Im_or(coord1(2,i)+vert, coord1(1,i)+64,2)= 255;
        Im_or(coord1(2,i)+vert, coord1(1,i)+64,3)= 0;
    end
    for hor=1:64
        Im_or(coord1(2,i), coord1(1,i)+hor,1)= 153;
        Im_or(coord1(2,i), coord1(1,i)+hor,2)= 255;
        Im_or(coord1(2,i), coord1(1,i)+hor,3)= 0;
        Im_or(coord1(2,i)+128, coord1(1,i)+hor,1)= 153;
        Im_or(coord1(2,i)+128, coord1(1,i)+hor,2)= 255;
        Im_or(coord1(2,i)+128, coord1(1,i)+hor,3)= 0;
    end
end
% sub scale detection
%for i=1:point2
%    for vert=1:64
%        Im_or(coord2(2,i)+vert, coord2(1,i),1)= 204;
%        Im_or(coord2(2,i)+vert, coord2(1,i),2)= 51;
%        Im_or(coord2(2,i)+vert, coord2(1,i),3)= 0;
%        Im_or(coord2(2,i)+vert, coord2(1,i)+32,1)= 204;
%        Im_or(coord2(2,i)+vert, coord2(1,i)+32,2)= 51;
%        Im_or(coord2(2,i)+vert, coord2(1,i)+32,3)= 0;
%    end
%    for hor=1:32
%        Im_or(coord2(2,i), coord2(1,i)+hor,1)= 204;
%        Im_or(coord2(2,i), coord2(1,i)+hor,2)= 51;
%        Im_or(coord2(2,i), coord2(1,i)+hor,3)= 0;
%        Im_or(coord2(2,i)+64, coord2(1,i)+hor,1)= 204;
%        Im_or(coord2(2,i)+64, coord2(1,i)+hor,2)= 51;
%        Im_or(coord2(2,i)+64, coord2(1,i)+hor,3)= 0;
%    end
%end

```

```
imshow(Im_or);
```

## Bin selection - Κώδικας ελέγχου

```
for i=-127:127
    for j=-127:127

        if i== 0 && j> 0;
            bin_Im(i+128,j+128)=1;
        elseif i== 0 && j< 0;
            bin_Im(i+128,j+128)=9;
        elseif i~= 0 && j== 0;
            bin_Im(i+128,j+128)=5;
        elseif (i>0 && j>0) || (i<0 && j<0)
            if abs(i) < abs(j) * 0.359375 % binary -> 0000.010111
                bin_Im(i+128,j+128)=1;
            elseif abs(i) < abs(j) * 0.828125 % binaty -> 0000.110101
                bin_Im(i+128,j+128)=2;
            elseif abs(i) < abs(j) * 1.71875 % binary -> 0001.101110
                bin_Im(i+128,j+128)=3;
            elseif abs(i) < abs(j) * 5.65625 % binary -> 0101.101010
                bin_Im(i+128,j+128)=4;
            else bin_Im(i+128,j+128)=5;
            end
        elseif (i>0 && j<0) || (i<0 && j>0)

            if abs(i) < abs(j) * 0.359375
                bin_Im(i+128,j+128)=9;
            elseif abs(i) < abs(j) * 0.828125
                bin_Im(i+128,j+128)=8;
            elseif abs(i) < abs(j) * 1.71875
                bin_Im(i+128,j+128)=7;
            elseif abs(i) < abs(j) * 5.65625
                bin_Im(i+128,j+128)=6;
            else bin_Im(i+128,j+128)=5;
            end
        end
    end
end
figure;
mesh(bin_Im);
```

## Οπτικοποίηση των χαρακτηριστικών διανυσμάτων με VL-Feat (απαιτείται VL\_feat βιβλιοθήκη)

```
clear;
clc;
Im=imread('C:\Users\antoge\Documents\hep\THESSIS_FPGA\pedestrians128x64.tar\pedestrians128x64\per00001.ppm');
I=rgb2gray(Im);
I=single(I);
hog = vl_hog(I, 8, 'verbose', 'variant', 'dalaltriggs');

my_hog = single(my_hog_quant(Im));
my_hog = single((my_hog/ max((my_hog(:))*.2)));
my_hog_nb = single(my_hog_quant_nb(Im));
my_hog_nb = single((my_hog_nb/ max((my_hog_nb(:))*.2)));

imhog = vl_hog('render', hog, 'verbose', 'variant', 'dalaltriggs');
my_imhog = vl_hog('render', my_hog, 'verbose', 'variant', 'dalaltriggs') ;
my_hog_nb= cat(3,
my_hog_nb(:,:, :, :),my_hog_nb(:,:, :, :),my_hog_nb(:,:, :, :));
imhog_nb = vl_hog('render', my_hog_nb, 'verbose', 'variant', 'dalaltriggs') ;
figure; subplot(1,3,1);
imshow(imhog);
```

```
subplot(1,3,2);
imshow(my_imhog);
subplot(1,3,3);
imshow(imhog_nb);
```

## Εκπαίδευση SVM ταξινομητή

```
%function [svm] =train_my_hog (folder)
%train_size=1436;
%test_size=1936-train_size;
folder_data='all';
[train_pos,test_pos,train_neg,test_neg]=data_info(folder_data);

all_data=train_pos+test_pos+train_neg +test_neg;
train_size=train_pos+train_neg;
bins=9;
cellsize=8;
block_size=4;
hogs=zeros(train_size,((128/cellsize)*(64/cellsize)
*(bins*block_size)));

X=ls
(strcat('C:\Users\antoge\Documents\hep\THESSIS_FPGA\train_dataset\'%
, folder_data, '\pos_neg\'));

for i= 3:train_size+2
    str = strcat(
'C:\Users\antoge\Documents\hep\THESSIS_FPGA\train_dataset\', folder_
data, '\pos_neg\', X(i,1:26));
    img=imread(str);
    img_gray=rgb2gray(img);
    img_gray=single(img_gray);

    hog = my_hog_quant(img_gray);
    [x,y,z]=size(hog);
    m=1;
    for j= 1:(x)
        for k= 1:(y)
            for l=1:(z)
                hog_vector(l,m)=hog(j,k,l);
                m=m+1;
            end
        end
    end

    hogs(i-2,:)=hog_vector;
end

lbls_pos= ones(1,train_pos);
lbls_neg= zeros(1,train_neg);
lbls= [lbls_neg lbls_pos];
lbls=lbls';
svm=svmtrain(hogs,lbls);
```

## Κώδικας Ελέγχου Ορθής Λειτουργία του κυκλώματος υπολογισμού Ιστογραμμάτων

```

hists= zeros ((lines- 6)/8,(cols- 6)/8 ,9);

for i= 4: i= lines - 3
    for j=4: j=cols -3
        hist_line = floor((i-4)/8)+1;
        hist_col = floor((j-4)/8)+1;
            hists (hist_line,hist_col, bin_Im(i,j))=
            hists(hist_line,hist_col,bin_Im(i,j))+
            abs(Gy_table(i,j))+abs(Gx_table(i,j));
    end
end
[x,y,z]=size(hists);
n=1;
for k= 1:(x)
    for l= 1:(y)
        for m=1:(z)
            histos(1,n)=hists(k,l,m);
            n=n+1;
        end
    end
end

```

## Δημιουργία αρχείων αρχικοποίησης ROM για ταξινόμηση

```

for rom_number= 0 : 8
    num = num2str(rom_number);
string = strcat('rom_mem',num,'.mif');
fid=fopen(string,'wt');
fprintf(fid,'-- external_mem.mif --\n');
fprintf(fid,'depth =128; -- 7-bit address ---> 0-127 dec \n');
fprintf(fid,'width = 12; -- 8-bit data \n');
fprintf(fid,'address_radix = dec; \n');
fprintf(fid,'data_radix = bin; \n');
fprintf(fid,'content \n');
fprintf(fid,'begin \n');

for i = 2:9:1153
    index = (i-2)/9;
    mem_line = floor((i-1)/9);
    fprintf(fid,'%d : ',mem_line);
    fprintf(fid,'%c',f(i+rom_number,:));
    fprintf(fid,'; \n');
    name = genvarname(strcat('mem',num2str(rom_number)));
    eval([name ' (index +1, 1)= d(i + rom_number)' ]);
end

fprintf(fid,'end; \n');
fid=fclose('all');
end
% for rom_number= 0 : 8
%     num = num2str(rom_number);
% string = strcat('rom_mem',num,'.mif');

```

```
% delete(string);  
% end
```

## ROC Curves

```
% load('hogs_nb_noNaN_5872.mat')  
% load('lbls_5872')  
cv = cvpartition(lbls, 'HoldOut', 0.7);  
xtrain = hogs_nb_noNaN_5872(cv.training,:);  
xtest = hogs_nb_noNaN_5872(cv.test,:);  
  
ytrain = lbls(cv.training);  
ytest = lbls(cv.test);  
  
b = glmfit(xtrain,ytrain,'binomial'); % logistic regression  
p = glmval(b,xtest,'logit'); % get fitted probabilities for  
scores  
  
[X,Y,auc, optimalPoint] = perfcurve( ytest,p,1);  
plot(X,Y)  
xlabel('False positive rate'); ylabel('True positive rate')  
title('ROC Curve for classification by logistic regression')
```



# Παράρτημα Προγραμμάτων VHDL

## Line Register

```

libraryieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity line_buf is
    generic (
        h_det_bl      : integer := 2;           -- platos detection window, se
blocks
        v_det_bl      : integer := 4;           -- ipsos detection window se blocks
        H_blk_size    : integer := 10;          -- platos eikonas se blocks
        V_blk_size    : integer := 6;           -- ipsos eikonas se blocks
        counter_8     : integer := 3;
        counter_60    : integer := 6;
        counter_80    : integer := 7;
        cols_width    : integer := 10;          -- bit gia na aNaparastisoume to 624
        lines_width   : integer := 9;           -- bit gia na aNaparastisoume to 464
        bits_rom_cnt: integer := 7;
        bias          : integer := 20;
        threshold     : integer := -2000;       -- katofli taxinomisis
        No_cols       : integer := 32;          -- stiles
        No_lines      : integer := 24;          -- grammes
        lines_cut     : integer := 8;           -- perigamma
        pixel_width   : integer := 8
    );
port (
    ser_data      : in std_logic_vector (pixel_width-1 downto 0);
    clk, clr : in std_logic;
    sign_en : out std_logic;
    init   : buffer std_logic;
    v_sync,h_sync : in std_logic;
    Gx, Gy   : out std_logic_vector (pixel_width-1 downto 0)
--counter_cols : out std_logic_vector(cols_width -1 downto 0);
--counter_lines : out std_logic_vector(lines_width -1 downto 0)
);
end line_buf;
-----
```

Architecture line\_buf of line\_buf is

```

component byte_buf
    port (
    clk, clr: in std_logic;
    enable : in std_logic;
```

```

byte_in : in std_logic_vector (7 downto 0);
byte_out: out std_logic_vector (7 downto 0));
end component;

component bit_buf
    port (
clk, clr: in std_logic;
bit_in  : in std_logic;
bit_out: out std_logic);
end component;

component n_bits_sub
    port(
A,B   : IN std_logic_vector(pixel_width-2 downto 0);
sub     : OUT std_logic_vector(pixel_width-1 downto 0);
clk      : IN std_logic);
end component;
-----
subtype BYTE is std_logic_vector(7 downto 0);
type transfer_byte is array (2*No_cols downto 0) of BYTE;
signal out_sign      : transfer_byte;
signal first_byte    : std_logic_vector(7 downto 0);
signal valid, valid2 : std_logic;
signal valid_en      : std_logic;
signal delayed_V_sync : std_logic;
signal col_cnt       : std_logic_vector (cols_width-1 downto 0);
signal line_cnt      : std_logic_vector (lines_width-1 downto 0);
signal Gx_temp, Gy_temp : std_logic_vector (pixel_width-1 downto 0);
signal temp_s1, temp_s2 : std_logic;
signal Gx1, Gx3       : std_logic_vector (pixel_width-1 downto 0);
signal Gy1, Gy3       : std_logic_vector (pixel_width-1 downto 0);

begin
init_signal: component bit_buf PORT MAP (clk, clr, v_sync, delayed_V_sync);
valid_signal : component bit_buf port map (clk, clr, valid, valid2);

temp_signal2: component bit_buf port map (clk, clr, temp_s2, sign_en);

sub_1: component n_bits_sub port map ( Gx3 (pixel_width-1 downto 1), Gx1 (pixel_width-1 downto 1), Gx_temp, clk); -- ~0 delay
sub_2: component n_bits_sub port map ( Gy3 (pixel_width-1 downto 1), Gy1 (pixel_width-1 downto 1), Gy_temp, clk);

linebuf: for i in 1 to 2*No_cols generate
    linebuf_i: component byte_buf PORT MAP (clk, clr, valid, out_sign(i-1),out_sign(i));
end generate;

process (clr, clk, valid, col_cnt, line_cnt, h_sync, ser_data, out_sign, init)

```

```

begin
if clr = '1' then
    Gx3 <= (others => '0');
    Gx1 <= (others => '0');
    Gy3 <= (others => '0');
    Gy1 <= (others => '0');
else
    if rising_edge(clk) then
        init <= (delayed_V_sync OR v_sync) XOR v_sync;
        out_sign(0)<= ser_data;
        if valid2 = '1' then
            if col_cnt >conv_std_logic_vector (1,cols_width) and col_cnt <
conv_std_logic_vector (No_cols-2 , cols_width) then
                if line_cnt > conv_std_logic_vector (2,lines_width) and line_cnt <
conv_std_logic_vector (No_lines , lines_width) then
                    Gx1 <= out_sign(No_cols);
                    Gx3 <= out_sign(No_cols-2);
                    Gy3 <= ser_data;
                    Gy1 <= out_sign(2*No_cols-1);
                end if;
            end if;
        end if;
    end if;
end if;
end process;

process (clr, clk, col_cnt, line_cnt, valid, valid2, init, Gx_temp, Gy_temp)
begin
if clr = '1' then
    line_cnt <= (others => '0');
    col_cnt <= (others => '0');
else
    if rising_edge(clk) then
        valid <= h_sync and v_sync;
        if valid2= '1' then
            if col_cnt >conv_std_logic_vector (lines_cut+1,cols_width) and col_cnt <
conv_std_logic_vector (No_cols-lines_cut +2 , cols_width) then
                if line_cnt > conv_std_logic_vector (lines_cut+1,lines_width) and line_cnt <
conv_std_logic_vector (No_lines -lines_cut+2, lines_width) then
                    Gx <= Gx_temp;
                    Gy <= Gy_temp;
                end if;
            end if;
            if col_cnt >conv_std_logic_vector (lines_cut+1,cols_width) and col_cnt <
conv_std_logic_vector (No_cols-lines_cut+2 , cols_width) then
                if line_cnt > conv_std_logic_vector (lines_cut+1,lines_width) and line_cnt <
conv_std_logic_vector (No_lines -lines_cut+2, lines_width) then
                    temp_s2<= '1';
                end if;
            end if;
        end if;
    end if;
end if;

```

```

        end if;
    else
        temp_s2<= '0';
    end if;
else
    Gx <= (others => '0');
    Gy <= (others => '0');
end if;
if (h_sync and v_sync)='1' then
    if col_cnt = conv_std_logic_vector(0 ,cols_width) then
        line_cnt <= line_cnt +1;
    end if;
    if col_cnt = conv_std_logic_vector(No_cols-1 ,cols_width) then
        col_cnt <=(others => '0');
    else
        col_cnt <= col_cnt +1;
    end if;
end if;
if init = '1' then
    line_cnt <= (others => '0');
    col_cnt <= (others => '0');
end if;
end if;
end if;
end process;

end Architecture;
```

## BIT BUFFER

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
Entity bit_buf is
PORT (
clk, clr: in std_logic;
bit_in : in std_logic;
bit_out: out std_logic);
End bit_buf;
```

```

Architecture arch_buf of bit_buf is
begin
process(clk,clr) begin
if (clr='1') then
    bit_out <= '0';
else
    if rising_edge(clk) then
```

```

        bit_out <= bit_in;
    end if;
end if;
end process;
End Architecture;
```

## N\_BITS\_SUB

```

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

ENTITY n_bits_sub IS
generic ( n : integer := 8);
PORT(
A,B      : IN std_logic_vector (n-2 downto 0);
Sub      : OUT std_logic_vector (n-1 downto 0);
clk      : IN std_logic);
END ENTITY;
```

Architecture n\_bits of n\_bits\_sub is

```

signal As, Bs, temp : std_logic_vector (n-1 downto 0);
signal temp_sub :std_logic_vector (n-1 downto 0);
begin
temp <=abs(As-Bs);
As <= '0' & A;
Bs <= '0' & B;
temp_sub <= As- Bs;
process(clk)
begin
if(rising_edge(clk)) then
    sub (n-2 downto 0) <=temp(n-2 downto 0);
    sub (n-1) <= temp_sub (n-1);
end if;
end process;
end n_bits;
```

## Bin Selection

```

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```

Entity bin_sel is
  generic (n : integer := 8);
  Port (
    Gx, Gy : in std_logic_vector (n-1 downto 0);
    bin     : out std_logic_vector (3 downto 0);
    clk     : in std_logic
  );
end entity;

```

Architecture beh\_bin\_sel of bin\_sel is

```

signal bin_lim1, bin_lim2, bin_lim3, bin_lim4 : std_logic_vector (8 downto 0);
signal Bin1_Gx, Bin2_Gx, Bin3_Gx, Bin4_Gx : std_logic_vector (15 downto 0);
signal abs_Gx, abs_Gy : std_logic_vector (n-2 downto 0);
signal Gy_shifted : std_logic_vector (15 downto 0);
signal zeros : std_logic_vector (n-2 downto 0); -- mideniko dianisma gia sigkriseis
signal sign_x, sign_y : std_logic;

begin
zeros <= (others => '0');

-- kano ta oria ton bins akeraious *64
bin_lim1 <= "000010111"; -- tan(20)
bin_lim2 <= "000110101"; -- tan(40)
bin_lim3 <= "001101110"; -- tan(60)
bin_lim4 <= "101101011"; -- tan(80)

Bin1_Gx <= bin_lim1 * abs_Gx ;
Bin2_Gx <= bin_lim2 * abs_Gx ;
Bin3_Gx <= bin_lim3 * abs_Gx ;
Bin4_Gx <= bin_lim4 * abs_Gx ;

Gy_shifted <= "000" & abs_Gy & "000000";

process (clk, Gy_shifted, abs_Gx, abs_Gy)
begin
if rising_edge(clk) then
  abs_Gx <= Gx (n-2 downto 0);
  abs_Gy <= Gy (n-2 downto 0);
  sign_x <= Gx (n-1);
  sign_y <= Gy (n-1);

    if ((abs_Gy = zeros) and (abs_Gx /= zeros) and (sign_x='0')) then
      bin<= "0001";
    elsif ((abs_Gy = zeros) and (abs_Gx /= zeros) and (sign_x='1')) then
      bin<= "1001";
    elsif ((abs_Gy /= zeros) and (abs_Gx = zeros)) then

```

```

bin<= "0101";
elsif ((abs_Gy = zeros) and (abs_Gx = zeros)) then
bin<= "1010";
elsif ((abs_Gy /= zeros) and (abs_Gx /= zeros) and (sign_x = sign_y)) then
if Gy_shifted < bin1_Gx then
bin<="0001";
elsif ((bin1_Gx <= Gy_shifted) and (Gy_shifted < bin2_gx)) then
bin<="0010";
elsif ((bin2_Gx <= Gy_shifted) and (Gy_shifted < bin3_gx)) then
bin<="0011";
elsif ((bin3_Gx <= Gy_shifted) and (Gy_shifted < bin4_gx)) then
bin<="0100";
else bin<="0101";
end if;
elsif ((abs_Gx /= zeros) and (abs_Gy /= zeros) and (sign_x/=sign_y)) then
if Gy_shifted < bin1_Gx then
bin<="1001";
elsif (( bin1_Gx<= Gy_shifted) and (Gy_shifted < bin2_gx)) then
bin<="1000";
elsif (( bin2_Gx <= Gy_shifted) and (Gy_shifted < bin3_gx)) then
bin<="0111";
elsif (( bin3_Gx <= Gy_shifted) and (Gy_shifted < bin4_gx)) then
bin<="0110";
else bin<="0101";
end if;
end if;
end if;
end process;
end beh_bin_sel;

```

```

library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bin_sync is
generic (
    h_det_bl      : integer := 1;           -- platos detection window, se
blocks
    v_det_bl      : integer := 2;           -- ipsos detection window se blocks
    H_blk_size    : integer := 4;           -- platos eikonas se blocks
    V_blk_size    : integer := 3;           -- ipsos eikonas se blocks
    counter_8     : integer := 3;
    counter_60    : integer := 6;
    counter_80    : integer := 7;
    h_blk_bits   : integer := 10;          -- bit gia na aNaparastisoume to 624
    v_blk_bits   : integer := 9;           -- bit gia na aNaparastisoume to 464
    bits_rom_cnt: integer := 7;

```

```

bias          : integer := 20;
threshold     : integer := -2000;      -- katofli taxinomisis
No_cols       : integer := 32;   -- stiles
No_lines      : integer := 22;   -- grammes
lines_cut     : integer := 8;      -- perigramma
pixel_width   : integer := 8
);

port (  pixel    : in std_logic_vector (pixel_width-1 downto 0);
        clk, clr      : in std_logic;
        v_sync,h_sync : in std_logic;
                    en   : out std_logic;
                    bin  : out std_logic_vector(3 downto 0);
        frame_end    : out std_logic;
        Gx, Gy       : out std_logic_vector (pixel_width-1 downto 0)
);
end entity;
-----Architecture-----
architecture bin_beh of bin_sync is

-----Components-----
component byte_buf
    port (
clk, clr: in std_logic;
enable : in std_logic;
byte_in : in std_logic_vector (7 downto 0);
byte_out: out std_logic_vector (7 downto 0));
end component;

component bit_buf
    port (
clk, clr: in std_logic;
bit_in  : in std_logic;
bit_out: out std_logic);
end component;

component n_bits_sub
    port(
A,B  : IN std_logic_vector(pixel_width-2 downto 0);
sub   : OUT std_logic_vector(pixel_width-1 downto 0);
clk    : IN std_logic);
end component;

component line_buf
    port(
ser_data      : in std_logic_vector (pixel_width-1 downto 0);
clk, clr      : in std_logic;
sign_en : out std_logic;
);

```

```

        init      : buffer std_logic;
v_sync,h_sync : in std_logic;
        Gx, Gy   : out std_logic_vector (pixel_width-1 downto 0));
end component;

component bin_sel
    generic (n : integer := 8);
    Port (
        Gx, Gy      : in std_logic_vector (n-1 downto 0);
        bin         : out std_logic_vector (3 downto 0);
        clk         : in std_logic
        --;
        --enable     : in std_logic
    );
end component;
-----signals-----
signal bin_en1      : std_logic;
signal bin_en2      : std_logic;
signal Gytemp       : std_logic_vector (pixel_width-1 downto 0);
signal GXtemp       : std_logic_vector (pixel_width-1 downto 0);
signal Gytemp1      : std_logic_vector (pixel_width-1 downto 0);
signal GXtemp1      : std_logic_vector (pixel_width-1 downto 0);
begin
Gxy: component line_buf port map (pixel, clk, clr, bin_en1, frame_end, v_sync, h_sync, Gxtemp,
Gytemp);

bin_out: component bin_sel port map (Gxtemp, Gytemp, bin, clk);

process (clk, clr)
begin
if clr= '1' then
--      bin_en2 <= '0';
      en <= '0';
else
    if rising_edge(clk) then
        --bin_en2 <= bin_en1;
        en <= bin_en1;
        Gxtemp1 <= Gxtemp;
        Gytemp1 <= Gytemp;
        Gx <= Gxtemp1;
        Gy <= Gytemp1;
        end if;
    end if;
end process;
--ax<=Gxtemp;
--ay<=Gytemp;

```

```
end architecture;
```

## Histogram Generator

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity hist_gen is
generic (hor_blocks : integer := 32;
          ver_blocks      : integer := 24;
          h_bl_bits       : integer := 10; -- bit gia na aNaparastisoume to 624
          v_bl_bits       : integer := 9;   -- bit gia na aNaparastisoume to 464
          pixel_width     : integer := 8
);
port (clk      : in std_logic;
      clr      : in std_logic;
      bin      : in std_logic_vector (3 downto 0);
      Gx, Gy  : in std_logic_vector (pixel_width-1 downto 0);
      sl       : in std_logic;
      wr_init : in std_logic;

      --temp signals for memory
      reg0    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg1    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg2    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg3    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg4    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg5    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg6    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg7    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
      reg8    : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);

      mem_wr :in std_logic;
      addwr :in std_logic_vector (12 downto 0);
      addrd :in std_logic_vector (12 downto 0);
      memory_out : out std_logic_vector (125 downto 0);
      sl_in   : in std_logic_vector (1 downto 0)

);
end entity;
```

---

```
architecture hist_beh of hist_gen is
```

```

component ram
port(
    clk      : in std_logic;
    wr       : in std_logic;
    address_wr : in std_logic_vector (12 downto 0);
    address_rd : in std_logic_vector (12 downto 0);
    mem_wr   : in std_logic_vector (125 downto 0);
    mem_rd   : out std_logic_vector (125 downto 0)
);
end component;

type hist_calc is array (8 downto 0) of std_logic_vector (13 downto 0);
signal hist_calc1, hist_calc2 : hist_calc; -- sistoihies (2) apo regs gia bin count
signal hist_i1, hist_i2      : hist_calc;
signal hist_init1, hist_init2 : hist_calc; -- sistoihia2 apo regs gia bin count

signal sl_q      : std_logic;
signal
hist_i1r,
hist_i2r      : std_logic_vector(13 downto 0);      --
signal sum_abs : std_logic_vector (pixel_width -1 downto 0); -- athroisma metron |Gx|+|Gy|
signal sum     : std_logic_vector(13 downto 0)-- athroisma : "000000" & |Gx|+|Gy|
signal wr      : std_logic_vector (8 downto 0);      -- DEMUX: bin -> reg enable
signal
hist_o1,
hist_o2      : std_logic_vector (13 downto 0);      -- exodos reg1 kai reg2
signal add_val : std_logic_vector (13 downto 0);
signal wr1, wr2 : std_logic_vector (8 downto 0);      -- epilogi reg gia tous dio counter
signal ram_in  : std_logic_vector (125 downto 0);
signal ram_out : std_logic_vector (125 downto 0);

-----
begin
sum_abs <= ('0' & Gx(pixel_width -2 downto 0)) + ('0' & Gy(pixel_width -2 downto 0));
sum    <= add_val + ("000000" & sum_abs);
memoir : ram port map (clk, mem_wr, addwr, addrd, ram_out, ram_in);
memory_out<= ram_out;
-- control points
    reg0  <= ram_in(13 downto 0);
    reg1  <= ram_in(27 downto 14);
    reg2  <= ram_in(41 downto 28);
    reg3  <= ram_in(55 downto 42);
    reg4  <= ram_in(69 downto 56);
    reg5  <= ram_in(83 downto 70);
    reg6  <= ram_in(97 downto 84);
    reg7  <= ram_in(111 downto 98);

```

```

reg8    <= ram_in(125 downto 112);
-- 

---DEMUX bin -> enable counter register
wr <= "000000001" when bin= "0001" else
    "000000010" when bin= "0010" else
    "000000100" when bin= "0011" else
    "000001000" when bin= "0100" else
    "000010000" when bin= "0101" else
    "000100000" when bin= "0110" else
    "001000000" when bin= "0111" else
    "010000000" when bin= "1000" else
    "100000000";
-- 

wr_gen : for i in 0 to 8 generate
begin
    wr1(i) <= (wr_init and sl) or (not (sl) and wr(i)); -- energopoiountai ola gia (wr_init ---
                                                        -- and sl) gia arhikopoiithoun apo ram 'i '0'
    wr2(i) <= (wr_init and not (sl)) or (sl and wr(i));
end generate wr_gen;
---MUX exodos hist_gen ston mux

hist_o1 <= hist_calc1 (0) when bin= "0001" else
    hist_calc1 (1) when bin= "0010" else
    hist_calc1 (2) when bin= "0011" else
    hist_calc1 (3) when bin= "0100" else
    hist_calc1 (4) when bin= "0101" else
    hist_calc1 (5) when bin= "0110" else
    hist_calc1 (6) when bin= "0111" else
    hist_calc1 (7) when bin= "1000" else
    hist_calc1 (8);

hist_o2 <= hist_calc2 (0) when bin= "0001" else
    hist_calc2 (1) when bin= "0010" else
    hist_calc2 (2) when bin= "0011" else
    hist_calc2 (3) when bin= "0100" else
    hist_calc2 (4) when bin= "0101" else
    hist_calc2 (5) when bin= "0110" else
    hist_calc2 (6) when bin= "0111" else
    hist_calc2 (7) when bin= "1000" else
    hist_calc2 (8);

add_val <= hist_o1 when sl ='0' else
    hist_o2;
sel_gen1: for i in 0 to 8 generate
    mux_sel1: process (sl_in, hist_i1r, hist_i2r, ram_in)
begin
    case sl_in(0) is

```

```

when '0' =>
    hist_init1(i) <= ram_in (14*(i+1)-1 downto 14*i);
    hist_init2(i) <= (others=>'0');
when others =>
    hist_init1(i) <= (others=>'0');
    hist_init2(i) <= ram_in (14*(i+1)-1 downto 14*i);
end case;
end process mux_sel1;
end generate sel_gen1;

sl_gen : for i in 0 to 8 generate
begin
    mux_sl: process (sl_in, sum, ram_in,hist_init2, hist_init1)
    begin
        case sl_in(1) is
            when '0' =>
                hist_i1(i) <= hist_init1(i);
                hist_i2(i) <= sum;
            when others =>
                hist_i1(i) <= sum;
                hist_i2(i) <= hist_init2(i);
        end case;
    end process mux_sl;
end generate sl_gen;
-----
-- dff --
dff:process(clk,clr,sl)
begin
    if clr='1' then
        sl_q<='0';
    else
        if rising_edge(clk) then
            sl_q <= sl;
        end if;
    end if;
end process dff;

mux_gen : for i in 0 to 8 generate
begin
    mux: process (sl_q, hist_calc1, hist_calc2)
    begin
        case sl_q is
            when '0' => ram_out (14*(i+1)-1 downto 14*i) <= hist_calc1(i);
            when others => ram_out (14*(i+1)-1 downto 14*i) <= hist_calc2(i);
        end case;
    end process mux;
end generate mux_gen;
reg_gen : for i in 0 to 8 generate

```

```

begin
reg: process (clk, clr, wr1, wr2, hist_i1, hist_i2 )
begin
    if clr='1' then
        hist_calc1(i) <= (others=>'0');
        hist_calc2(i) <= (others=>'0');
    else
        if rising_edge(clk) then
            if wr1(i)='1' then
                hist_calc1(i) <= hist_i1(i);
            else
                hist_calc1(i) <= hist_calc1(i);
            end if;
            if wr2(i)='1' then
                hist_calc2(i) <= hist_i2(i);
            else
                hist_calc2(i) <= hist_calc2(i);
            end if;
        end if;
    end if;
end process reg;
end generate reg_gen;
end architecture;

```

## RAM read

```

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

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

entity read_mem is
generic (
    h_det_bl      : integer := 3;          -- platos detection window, se blocks
    v_det_bl      : integer := 5;          -- ipsos detection window se blocks
    H_bl_size     : integer := 8;          -- platos eikonas se blocks
    V_bl_size     : integer := 12;         -- ipsos eikonas se blocks
    counter_8     : integer := 3;
    counter_60    : integer := 6;

```

```

        counter_80      : integer := 7;
        h_bl_bits       : integer := 10; -- bit gia na aNaparastisoume to 624
        v_bl_bits       : integer := 9;           -- bit gia na aNaparastisoume to 464
        bits_rom_cnt: integer := 7;
        bias            : integer := 20;
        threshold       : integer :=-2000;       -- katofli taxinomisis
        No_cols         : integer := 96; -- stiles
        No_lines        : integer := 64; -- grammes
        lines_cut       : integer := 8;           -- perigramma
        pixel_width     : integer := 8
    );

```

port ( clk : in std\_logic;
 start\_stop : in std\_logic;
 rd\_memory : in std\_logic;
 mem\_clas : out std\_logic\_vector (12 downto 0)
 );
end entity;

architecture read\_mem\_beh of read\_mem is

```

signal Y_bl          : std_logic_vector (3 downto 0);
signal X_bl          : std_logic_vector (2 downto 0);
signal h_step        : std_logic_vector (5 downto 0);
signal v_step        : std_logic_vector (5 downto 0);
signal shift         : std_logic;
signal ver_block     : std_logic_vector (5 downto 0);
signal hor_block     : std_logic_vector (6 downto 0);
-----
```

begin

```

class_mem_pointer: process (clk, X_bl, Y_bl, h_step, v_step, shift, start_stop)
begin
if start_stop = '1' then
    X_bl <= (others => '0');
    Y_bl <= (others => '0');
    h_step <= (others => '0');
else
    if rising_edge (clk) then
        if rd_memory = '1' then
            if X_bl = conv_std_logic_vector(h_det_bl - 1,3) then
                X_bl <= (others => '0');
                if Y_bl= conv_std_logic_vector(v_det_bl -1,4) then
                    Y_bl <= (others => '0');
                    if h_step = conv_std_logic_vector(H_bl_size - h_det_bl, 6) then
                        if v_step = conv_std_logic_vector(V_bl_size - v_det_bl , 6) then
                            v_step<= (others => '0');
                        else

```

```

        v_step <= v_step + 1;
    end if;
        h_step <= (others => '0');
        else
            h_step <= h_step + 1;
    end if;
    else
        Y_bl <= Y_bl +1;

    end if;
else
    X_bl <= X_bl +1 ;
end if;
end if;
end if;
end process;

mem_clas<= ((v_step) * conv_std_logic_vector(H_bl_size, 7))+ Y_bl
*conv_std_logic_vector(H_bl_size, 7) +(h_step + X_bl);
end architecture;

```

## Classifier

```

library ieee ;
use ieee.std_logic_1164.all;
--use ieee.numeric_std.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity classifier is
generic (
    h_det_bl      : integer := 2;          -- platos detection window, se blocks
    v_det_bl      : integer := 2;          -- ipsos detection window se blocks
    H_bl_size     : integer := 10;         -- platos eikonas se blocks
    V_bl_size     : integer := 6;           -- ipsos eikonas se blocks
    counter_8      : integer := 3;
    counter_60     : integer := 6;
    counter_80     : integer := 7;
    h_bl_bits     : integer := 10;          -- bit gia na aNaparastisoume to 624
    v_bl_bits     : integer := 9;           -- bit gia na aNaparastisoume to 464
    bits_rom_cnt: integer := 7;
    bias          : integer := 20;
    threshold     : integer := -1000;       -- katofli taxinomisis
    No_cols       : integer := 96;          -- stiles
    No_lines      : integer := 64;          -- grammes
    lines_cut     : integer := 8;           -- perigramma

```

```

        pixel_width      : integer := 8
    );

port(
    clk           : in std_logic;
    init          : in std_logic;
    addr_rd       : in std_logic_vector (2 downto 0);
    draw_x        : out std_logic_vector (9 downto 0);
    draw_y        : out std_logic_vector(8 downto 0);
    BOXES         : out std_logic_vector (2 downto 0);
    valid_q       : in std_logic;

    add_det_x_o   : out std_logic_vector (9 downto 0);
    add_det_y_o   : out std_logic_vector (8 downto 0);
    reg0_o        : out signed (13 downto 0);
    rom0_o        : out signed (11 downto 0);
    add_wr_o      : out std_logic;

    hist          : in signed (125 downto 0)
);
end entity;

architecture arch_class of classifier is

component rom_mem0 IS
PORT(address  : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock      : IN STD_LOGIC ;
      q          : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem1 IS
PORT(address  : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock      : IN STD_LOGIC ;
      q          : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem2 IS
PORT(address  : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock      : IN STD_LOGIC ;
      q          : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem3 IS
PORT(address  : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock      : IN STD_LOGIC ;
      q          : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

```

```

component rom_mem4 IS
PORT(address : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock       : IN STD_LOGIC ;
      q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem5 IS
PORT(address : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock       : IN STD_LOGIC ;
      q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem6 IS
PORT(address : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock       : IN STD_LOGIC ;
      q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem7 IS
PORT(address : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock       : IN STD_LOGIC ;
      q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component rom_mem8 IS
PORT(address : IN STD_LOGIC_VECTOR (bits_rom_cnt -1 DOWNTO 0);
      clock       : IN STD_LOGIC ;
      q           : OUT STD_LOGIC_VECTOR (11 DOWNTO 0));
END component;

component ram_x is
port( clk       : in std_logic;
      wr        : in std_logic;
      address_wr : in std_logic_vector (2 downto 0);
      address_rd : in std_logic_vector (2 downto 0);
      mem_wr    : in std_logic_vector (9 downto 0);
      mem_rd    : out std_logic_vector (9 downto 0)
);
end component;

component ram_y is
port( clk       : in std_logic;
      wr        : in std_logic;
      address_wr : in std_logic_vector (2 downto 0);
      address_rd : in std_logic_vector (2 downto 0);
      mem_wr    : in std_logic_vector (8 downto 0);
      mem_rd    : out std_logic_vector (8 downto 0)
);

```

```

end component;
-----signals-----
signal wr, wr_q, draw_box : std_logic;

signal add_wr : std_logic_vector (2 downto 0);
signal rom_counter : std_logic_vector (6 downto 0);
signal
mult_res0,
mult_res1,mult_res2,mult_res3,mult_res4,
mult_res5,mult_res6,mult_res7,mult_res8, mult_res10 : signed (25 downto 0);
signal
mult_res0t,mult_res1t,mult_res2t,mult_res3t
,mult_res4t,mult_res5t,mult_res6t,mult_res7t,
mult_res8t : signed (25 downto 0);

signal
reg0,reg1,reg2,reg3,reg4,reg5,reg6,reg7,reg8 : signed (13 downto 0);
--
signal
sum10,sum11,sum12,sum13,sum14 : signed (15 downto 0);
signal
sum20, sum21, sum22 : signed (15 downto 0);
signal
sum31, sum30 : signed (15 downto 0);
signal
rom0_in, rom1_in, rom2_in,
rom3_in, rom4_in, rom5_in,
rom6_in, rom7_in, rom8_in : std_logic_vector (11 downto 0);
signal add_val, temp_sum, res : signed (15 downto 0);
signal final_val : signed (15 downto 0);
signal det_x, det_x_q : std_logic_vector (6 downto 0);
signal det_y, det_y_q : std_logic_vector (5 downto 0);
signal add_det_x : std_logic_vector (9 downto 0);
signal add_det_y : std_logic_vector (8 downto 0);

signal
rom_counter_q,
rom_counter_2q,
rom_counter_3q,
rom_counter_4q,
rom_counter_5q,
rom_counter_6q
--,
--rom_counter_7q
: std_logic_vector (6 downto 0);

-----signals
signal valid_2q, valid : std_logic;

```

```

signal rom_counter_7q           : std_logic_vector (6 downto 0);
signal to_thresh               : signed (15 downto 0);
-----
begin
ROM0: rom_mem0 PORT MAP (rom_counter, clk, rom0_in);
ROM1: rom_mem1 PORT MAP (rom_counter, clk, rom1_in);
ROM2: rom_mem2 PORT MAP (rom_counter, clk, rom2_in);
ROM3: rom_mem3 PORT MAP (rom_counter, clk, rom3_in);
ROM4: rom_mem4 PORT MAP (rom_counter, clk, rom4_in);
ROM5: rom_mem5 PORT MAP (rom_counter, clk, rom5_in);
ROM6: rom_mem6 PORT MAP (rom_counter, clk, rom6_in);
ROM7: rom_mem7 PORT MAP (rom_counter, clk, rom7_in);
ROM8: rom_mem8 PORT MAP (rom_counter, clk, rom8_in);
ramX: ram_x  PORT MAP (clk, wr, add_wr, addr_rd, add_det_x, draw_x);
ramY: ram_y  PORT MAP (clk, wr, add_wr, addr_rd, add_det_y, draw_y);

add_det_x <= (det_x_q & "000") + 8;
add_det_y <= (det_y_q & "000") + 8;

add_det_x_o<= add_det_x;
add_det_y_o<= add_det_y;

BOXES <= add_wr;
-----
process (clk, init, valid_q, hist, rom_counter)
begin
if init = '1' then
    reg0  <= (others=>'0');---thelei clk?
    reg1  <= (others=>'0');
    reg2  <= (others=>'0');
    reg3  <= (others=>'0');
    reg4  <= (others=>'0');
    reg5  <= (others=>'0');
    reg6  <= (others=>'0');
    reg7  <= (others=>'0');
    reg8  <= (others=>'0');
else
    if rising_edge(clk) then
        if valid_q= '1' then
            reg0 <= hist(13 downto 0); ---thelei clk?
            reg1 <= hist(27 downto 14);
            reg2 <= hist(41 downto 28);
            reg3 <= hist(55 downto 42);
            reg4 <= hist(69 downto 56);
            reg5 <= hist(83 downto 70);
            reg6 <= hist(97 downto 84);

```

```

        reg7 <= hist(111 downto 98);
        reg8 <= hist(125 downto 112);
        rom_counter_q <= rom_counter;
    end if;
end if;
end process;

temp_sum <= (others=>'0') when rom_counter_6q = conv_std_logic_vector (0, 7) else
    res;
res <= final_val + add_val;

final_proc: process (clk, temp_sum, init, valid, rom_counter_5q )
begin
if init = '1' then
    final_val <= (others => '0');
else
    if rising_edge(clk) then
        if valid = '1' then
            final_val <= temp_sum;
        else
            final_val <= final_val;
        end if;
        rom_counter_6q <= rom_counter_5q;
    end if;
end if;
end process;

write_ram: process (clk, wr, init, valid_q, det_x, det_y )
begin
if init = '1' then
    wr_q <= '0';
    valid_2q <= '0';
else
    if rising_edge(clk) then
        wr_q <= wr;
        valid<= valid_2q ;
        valid_2q <= valid_q;
        det_y_q <= det_y;
        det_x_q <= det_x;
    end if;
end if;
end process;

coefs: process (clk, init, valid)
begin
if init = '1' then
    det_x <= (others=>'0');

```

```

det_y <= (others=> '0');
else
    if rising_edge(clk) then
        if valid = '1' then
            if rom_counter = conv_std_logic_vector ((h_det_bl * v_det_bl)-1, 7) then
                if det_x = conv_std_logic_vector (h_bl_size - h_det_bl , 10) then
                    det_y <= det_y + 1;
                    det_x <= (others=> '0');
                else
                    det_y <= det_y;
                    det_x <= det_x + 1;
                end if;
            else
                det_x <= det_x ;
                det_y <= det_y;
            end if;
        end if;
    end if;
end if;
end process;

thres: process (init, clk, wr, add_wr, rom_counter_6q, rom_counter_7q, final_val, draw_box)
begin

if init = '1' then
    add_wr <= (others => '0');
    wr <= '0';
else
    if rising_edge(clk) then
        rom_counter_7q <= rom_counter_6q;
        if rom_counter_7q = conv_std_logic_vector((h_det_bl * v_det_bl)-1, 7) then
            to_thresh <= res;
            draw_box <= '1';
        else
            draw_box <= '0';
        end if;
        if draw_box = '1' then
            if to_thresh < conv_signed(threshold,16) then
                if add_wr = "111" then
                    wr <= '0';
                    add_wr <= add_wr;
                else
                    add_wr <= add_wr +1;
                    wr <= '1';
                end if;
            end if;
        else
            wr <= '0';
        end if;
    end if;
end process;

```

```

            add_wr <= add_wr;
        end if;
    end if;
end if;
end process;

address_cnt: process (clk, init, valid, rom_counter)
begin
if init = '1' then
    rom_counter <= (others=> '0');
else
    if rising_edge(clk) then
        if valid_q = '1' then
            if rom_counter = conv_std_logic_vector ((h_det_bl * v_det_bl)-1, 7) then
                rom_counter <= (others=> '0');
            else
                rom_counter <= rom_counter +1;
            end if;
        end if;
    end if;
end if;
end process;

reg0_o <= reg0 ;
rom0_o <= signed(rom0_in);

mult_res0 <= reg0 * signed(rom0_in);
mult_res1 <= reg1 * signed(rom1_in);
mult_res2 <= reg2 * signed(rom2_in);
mult_res3 <= reg3 * signed(rom3_in);
mult_res4 <= reg4 * signed(rom4_in);
mult_res5 <= reg5 * signed(rom5_in);
mult_res6 <= reg6 * signed(rom6_in);
mult_res7 <= reg7 * signed(rom7_in);
mult_res8 <= reg8 * signed(rom8_in);

mults: process (clk, init, reg0, reg1, reg2, reg3, reg4, reg5, reg6, reg7, reg8, valid_2q,
                 rom0_in, rom1_in, rom2_in, rom3_in, rom4_in, rom5_in, rom6_in, rom7_in,
                 rom8_in, res)
begin
if init = '1' then
    mult_res0t <= (others=> '0');
    mult_res1t <= (others=> '0');
    mult_res2t <= (others=> '0');
    mult_res3t <= (others=> '0');
    mult_res4t <= (others=> '0');
    mult_res5t <= (others=> '0');
    mult_res6t <= (others=> '0');

```

```

mult_res7t <= (others=> '0');
mult_res8t <= (others=> '0');

else
    if rising_edge (clk) then
        if valid_2q = '1' then
            mult_res0t <= mult_res0;
            mult_res1t <= mult_res1;
            mult_res2t <= mult_res2;
            mult_res3t <= mult_res3;
            mult_res4t <= mult_res4;
            mult_res5t <= mult_res5;
            mult_res6t <= mult_res6;
            mult_res7t <= mult_res7;
            mult_res8t <= mult_res8;
        end if;
    end if;
end if;
end process;

sums: process (clk, init,sum10, sum11, sum12, sum13, sum14, sum20, sum21, sum22, sum30,
sum31,
rom_counter_q, rom_counter_2q, rom_counter_3q, rom_counter_4q,
mult_res0, mult_res1, mult_res2, mult_res3, mult_res4, mult_res5, mult_res6, mult_res7,
mult_res8 )
begin
if init = '1' then
    sum10 <= (others=> '0');
    sum11 <= (others=> '0');
    sum12 <= (others=> '0');
    sum13 <= (others=> '0');
    sum14 <= (others=> '0');
    sum20 <= (others=> '0');
    sum21 <= (others=> '0');
    sum22 <= (others=> '0');
    sum31 <= (others=> '0');
    sum30 <= (others=> '0');
else
    if rising_edge (clk) then
        if valid = '1' then
            sum10 <= mult_res0t (25 downto 10)+ mult_res4t(25 downto 10);
            sum11 <= mult_res1t (25 downto 10)+ mult_res5t(25 downto 10);
            sum12 <= mult_res2t (25 downto 10)+ mult_res6t(25 downto 10);
            sum13 <= mult_res3t (25 downto 10)+ mult_res7t(25 downto 10);---
            sum14 <= mult_res8t (25 downto 10);
            rom_counter_2q<= rom_counter_q;
-----  

            sum20 <= sum10 +sum11;
            sum21 <= sum12 +sum13;
-----
```

```

        sum22 <= sum14;
        rom_counter_3q <= rom_counter_2q;

        sum30 <= sum20 + sum21;
        sum31 <= sum22;
        rom_counter_4q <= rom_counter_3q;
-----
```

```

        add_val <= sum30 + sum31;
        rom_counter_5q <= rom_counter_4q;
    end if;
end if;
end process;

end architecture;
```

## ROM

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY rom_mem0 IS
    PORT
    (
        address      : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
        clock        : IN STD_LOGIC ;
        q            : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
    );
END rom_mem0;
ARCHITECTURE SYN OF rom_mem0 IS

    SIGNAL sub_wire0      : STD_LOGIC_VECTOR (11 DOWNTO 0);
    COMPONENT altsyncram
    GENERIC (
        clock_enable_input_a      : STRING;
        clock_enable_output_a     : STRING;
        init_file                : STRING;
        intended_device_family    : STRING;
        lpm_hint                 : STRING;
```

```

lpm_type : STRING;
numwords_a : NATURAL;
operation_mode : STRING;
outdata_aclr_a : STRING;
outdata_reg_a : STRING;
widthad_a : NATURAL;
width_a : NATURAL;
width_bytlena_a : NATURAL
);
PORT (
    clock0 : IN STD_LOGIC ;
    address_a : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    q_a : OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
);
END COMPONENT;

BEGIN
    q <= sub_wire0(11 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "rom_mem0.mif",
        intended_device_family => "Cyclone II",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 128,
        operation_mode => "ROM",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "CLOCK0",
        widthad_a => 7,
        width_a => 12,
        width_bytlena_a => 1
    )
    PORT MAP (
        clock0 => clock,
        address_a => address,
        q_a => sub_wire0
    );

```

```
END SYN;

-- =====
-- CNX file retrieval info
-- =====
-- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
-- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
-- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
-- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
-- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: Clken NUMERIC "0"
-- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
-- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
-- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
-- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
-- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
-- Retrieval info: PRIVATE: MIFfilename STRING "data_file_here"
-- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "128"
-- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
-- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
-- Retrieval info: PRIVATE: RegOutput NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
-- Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
-- Retrieval info: PRIVATE: WidthAddr NUMERIC "7"
-- Retrieval info: PRIVATE: WidthData NUMERIC "10"
-- Retrieval info: PRIVATE: rden NUMERIC "0"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: INIT_FILE STRING "data_file_here"
-- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
-- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "128"
-- Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
```

```
-- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
-- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
-- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "7"
-- Retrieval info: CONSTANT: WIDTH_A NUMERIC "10"
-- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
-- Retrieval info: USED_PORT: address 0 0 7 0 INPUT NODEFVAL address[6..0]
-- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL clock
-- Retrieval info: USED_PORT: q 0 0 10 0 OUTPUT NODEFVAL q[9..0]
-- Retrieval info: CONNECT: @address_a 0 0 7 0 address 0 0 7 0
-- Retrieval info: CONNECT: q 0 0 10 0 @q_a 0 0 10 0
-- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
-- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0.cmp FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0.bsf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0_inst.vhd FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0_waveforms.html FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL rom_mem0_wave*.jpg FALSE
-- Retrieval info: LIB_FILE: altera_mf
```