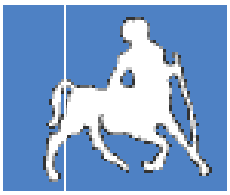


Διπλωματική Εργασία

<<Μελέτη του προτύπου video με την ονομασία "Scalable Video Coding">>

Επιμελητής:

Τσουμπλέκας Γεώργιος του Κωνσταντίνου

Επιβλέπων Καθηγητής

Ιωάννης Κατσαβουνίδης

(Αναπληρωτής Καθηγητής Πανεπιστημίου Θεσσαλίας)

Συνεπιβλέπων Καθηγητής

Γεώργιος Σταμούλης

(Καθηγητής Πανεπιστημίου Θεσσαλίας)

Βόλος, Ιούλιος 2010

Ευχαριστίες

Μετά από ένα ωραίο ταξίδι 6 χρόνων ,σαν προπτυχιακός φοιτητής στο Τμήμα Μηχανικών Η/Υ , Δικτύων και Τηλεπικοινωνιών, ολοκληρώνω τις προπτυχιακές σπουδές μου με την εκπόνηση της διπλωματικής μου εργασίας .

Ευχαριστώ τους επιβλέποντες καθηγητές μου, κ. Κατσαβουνίδη και κ. Σταμούλη για την άριστη συνεργασία που είχαμε σε όλη την διάρκεια της ενασχόλησης μου για την ολοκλήρωση της διπλωματικής εργασίας μου. Ιδιαίτερα ευχαριστώ το κ. Κατσαβουνίδη διότι, χωρίς τις συμβουλές και τις άπειρες ώρες συνεργασίας μαζί του, δε θα είχα καταφέρει να ολοκληρώσω με επιτυχία την διπλωματική εργασία. Επιπλέον θα ήθελα να ευχαριστώ και τα υπόλοιπα μέλη της ομάδας που δουλεύουν πάνω στον SVC (κ. Δημήτρη Ζαχαρή, κ. Μαρία Κοζύρη) για την μικρή και μεγάλη βοήθεια που μου πρόσφεραν όταν την χρειάστηκα .

Επιπροσθέτως, θα ήθελα να ευχαριστήσω ένα καλό φίλο, τον Κωνσταντή Νταλούκα, για την βοήθεια που μου πρόσφερε σε όλη την διάρκεια της φοιτητικής μου πορείας.

Επίσης ένα μεγάλο ευχαριστώ στην καλή φίλη Βασιλική Χριστοδούλου, που επιμελήθηκε την λεκτική και συντακτική ανάλυση της διπλωματικής μου.

Τέλος, ευχαριστώ θερμά την οικογένεια μου για την αμέριστη συμπαράσταση που μου πρόσφεραν όλα αυτά τα χρόνια, για να καταφέρω με επιτυχία να ολοκληρώσω με επιτυχία τις σπουδές μου.

Στην οικογένεια μου,
και στην Μαρία

Κεφάλαιο 1

1.1 Κωδικοποιητές και αποκωδικοποιητές video

Καθημερινά στην ζωή μας ,χρησιμοποιούμε πληθώρα κωδικοποιητών και αποκωδικοποιητών video. Τα κινητά τηλέφωνα 3ης γενιάς, η ψηφιακή τηλεόραση, το dvd player, το ipad χρησιμοποιούν πολλούς κωδικοποιητές και αποκωδικοποιητές video. Οι κωδικοποιητές video συμπιέζουν και οι αποκωδικοποιητές αποσυμπιέζουν την πληροφορία. Ένα video πολλών MB, μερικές φορές και πολλών GB για video υψηλής ανάλυσης, πρέπει να κωδικοποιηθεί σε ένα αρχείο λίγων KB για εξοικονόμηση χώρου, ή και για την μεταφορά του πάνω από το διαδίκτυο σε πραγματικό χρόνο. Ο παραλήπτης θα πρέπει να το αποκωδικοποιήσει και το video που θα προκύψει θα πρέπει να έχει παραπλήσια , αν όχι ίδια με το αρχικό ποιότητα. Επιπλέον η διαδικασία αυτή θα πρέπει να εκτελείται γρήγορα, ώστε να μη αντιλαμβάνεται ο παρατηρητής καθυστέρηση στην απεικόνιση των καρέ στην οθόνη. Ο ρυθμός αποκωδικοποίησης των frames εξαρτάται από την ανάλυση του video. Για ανάλυση 720x480, θα πρέπει ένας αποκωδικοποιητής πραγματικού χρόνου να αποκωδικοποιεί 30 καρέ το δευτερόλεπτο. Αντίστοιχα για ανάλυση 720x576, θα πρέπει να αποκωδικοποιεί 25 καρέ το δευτερόλεπτο. Τα πιο διαδεδομένα πρότυπα video είναι το MPGE-1, MPEG-2, MPEG_4, H.264.

1.2 Γρήγορη περιγραφή του προτύπου SVC

Η γρήγορη εξάπλωση του διαδικτύου, έξυπνων κινητών καθώς και των εφαρμογών video πάνω από το διαδίκτυο, οδήγησε πολλά standard video σε μια προσπάθεια, δημιουργία video που θα προσαρμόζονται, ανάλογα με τις απαιτήσεις του διαδικτύου, ή τιων συσκευών αναπαραγωγής video. Ωστόσο όλες οι εμπορικές εφαρμογές, δε είχαν την αναμενόμενη επιτυχία. Όμως το πρότυπο SVC πέτυχε, σε μεγάλο βαθμό όλα τα παραπάνω. Την δημιουργία ενός προσαρμοζόμενου κωδικοποιητή και αποκωδικοποιητή video .

Το πρότυπο SVC ,στην πραγματικότητα δε αποτελεί ένα εντελώς καινούργιο πρότυπο. Αλλά αποτελεί ένα νέο και πολλά υποσχόμενο χαρακτηριστικό του ήδη διαδεδομένου H.264. Το χαρακτηριστικό του SVC είναι ότι παρέχει την δυνατότητα κωδικοποίησης μιας εικόνας σε πολλά επίπεδα (Layers). Εισάγει την έννοια των επιπέδων και των ανεξάρτητων συνόλων. Ο διαχωρισμός των επιπέδων γίνεται με 3 τρόπους :

- Spatial Scalabity (χωρική κλιμάκωση).
- Temporal Scalability (χρονική κλιμάκωση).
- SNR Scalability (κλιμάκωση με βάση την ποιότητα).

Η εικόνα 1 (figure 1) δείχνει πως λειτούργει ο SVC

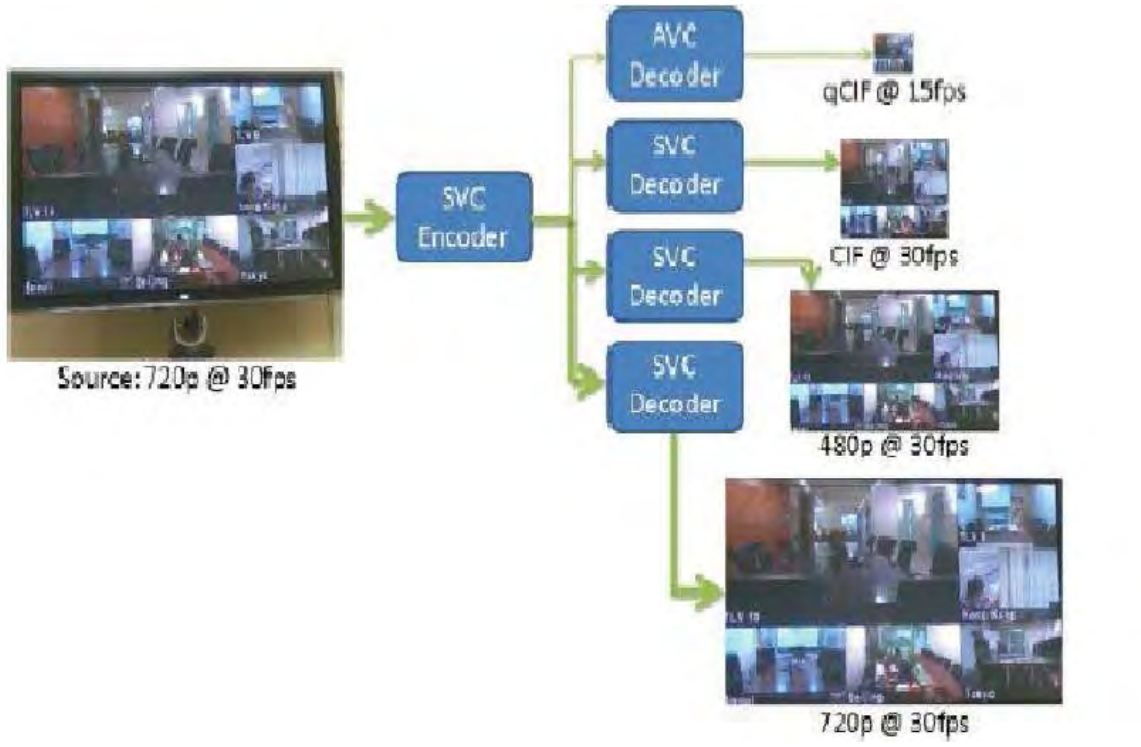


Figure 1

Αναλυτική περιγραφή του SVC στο κεφάλαιο 2

1.3 Σκοπός και στόχος της διπλωματικής

Στόχος της παρούσας διπλωματικής εργασίας ,ήταν να μελετηθεί το πρότυπο αυτό και να κατανοηθεί πως δουλεύει. Στην συνέχεια έπρεπε να βελτιστοποιήσουμε τον αποκωδικοποιητή, με στόχο την αποκωδικοποίηση ενός bitstream πολλών layer σε πραγματικό χρόνο, για ανάλυση D1(720x480) με ρυθμό κωδικοποίησης 30 καρέ. Αυτό τελικά έγινε πραγματικότητα με χρήση διανυσματικών εντολών και αρχιτεκτονικών αλλαγών. Αναλυτική περιγραφή θα παρουσιαστεί στα επόμενα κεφάλαια.

Κεφάλαιο 2

2.1Γενικές λεπτομέρειες για το πρότυπο SVC

Scalable Video coding είναι το όνομα που έδωσαν στο νέο extension του H.264 , που στο standard του H.264 περιγράφεται ως Annex G. Το πρότυπο SVC αναπτύχθηκε από το μια ομάδα μηχανικών που αποτελούν το group με την ονομασία " Joint Video Team". Το πρότυπο SVC είναι γραμμένο σε C++, και για να το κατεβάσει κάποιος, αρκεί να εγκαταστήσει έναν CVS client στον υπολογιστή του και να δώσει τις κατάλληλες παραμέτρους. Εκτός από τον encoder και τον decoder, ο προγραμματιστής θα λάβει και μια πληθώρα tools, αλλά και ένα πολύ αναλυτικό manual.

Τα εργαλεία αυτά είναι

- **DownConverStatic:**

  Το εργαλείο αυτό χρησιμοποιείται για να δημιουργήσει από ένα αρχικό stream video, ένα νέο video με διαφορετική ανάλυση, καθώς και με διαφορετικό ρυθμό απεικόνισης των καρέ.

- **BitStreamExtractor:**

  Όπως αναφέραμε στο κεφάλαιο 1, η λογική του SVC είναι η δημιουργία πολλών επιπέδων, σε ένα μόνο κωδικοποιημένο αρχείο. Το εργαλείο αυτό δίνει την δυνατότητα από το κωδικοποιημένο αρχείο να εξάγει ο προγραμματιστής, ένα υποσύνολο του κωδικοποιημένου αρχείου που αντιστοιχεί σε ένα επίπεδο ή συνδυασμός πολλών επιπέδων.

- **QualityLevelAssigner**

  Το παρόν εργαλείο δέχεται ως είσοδο το αρχικό κωδικοποιημένο SVC αρχείο. Ενσωματώνει την πληροφορία, όσο αναφορά την ποιότητα του κάθε επιπέδου, δηλαδή πληροφορίες για το QP. Τέλος παράγει ένα νέο αρχείο που περιέχει τις επιπρόσθετες πληροφορίες.

- **MCTFPreProcessorStatic**

  Το εργαλείο αυτό, χρησιμοποιείται για φιλτράρισμα και βελτίωση της ποιότητας της εικόνας πριν αυτή κωδικοποιηθεί.

- **PSNRStatic**

  Το PSNRStatic είναι το κατάλληλο εργαλείο για τον υπολογισμό του σηματοθορυβικού λόγου δυο εικόνων.

- **FixQPEncoderStatic**

  Το συγκεκριμένο εργαλείο παρέχει την δυνατότητα του ελέγχου και της προσαρμογής του bit-rate κατά την διαδικασία της κωδικοποίησης.

- **YUVCompareStatic**

  Συγκρίνει δυο εικόνες αν είναι ίδιες

- **SIPAnalyzer**

  Τέλος, το εργαλείο αυτό χρησιμοποιείται για να κάνει επιλεκτικό inter-layer prediction

2.2Αναλυτική περιγραφή

Στο κεφάλαιο 1 αναφέραμε ,συνοπτικά το βασικό χαρακτηριστικό του SVC. Δηλαδή, σύμφωνα με το πρότυπο, ο κωδικοποιητής δέχεται σαν είσοδο ένα video υψηλής ανάλυσης για παράδειγμα 720x480@30fps, και παράγει ένα κωδικοποιημένο αρχείο 264. Το αρχείο αυτό περιέχει περισσότερα από ένα επίπεδα, τα οποία όταν αποκωδικοποιηθούν θα μας δώσουν ένα video με τα ίδια χαρακτηριστικά με αυτά του αρχικού, δηλαδή ίδια ανάλυση, ίδιο frame rate αλλά και ίδια ποιότητα. Όμως από το ίδιο κωδικοποιημένο αρχείο, μπορούμε να παράγουμε video μικρότερης ανάλυσης, διαφορετικού bitrate, και διαφορετικής ποιότητας. Ο κωδικοποιητής, στην πραγματικότητα παράγει μια μικρή κωδικοποιημένη εικόνα, και ο αποκωδικοποιητής μέσα από διαδικασίες που θα περιγραφούν αναλυτικά σε επόμενα κεφάλαια, παράγει την αρχική εικόνα από την μικρή αποκωδικοποιημένη εικόνα που παίρνει ως είσοδο.

Στην εικόνα 1(figure1), φαίνεται η δομή ενός κωδικοποιητή SVC, τριών επιπέδων χωρικής κλιμάκωσης. Η κωδικοποίηση και κατ' επέκταση η αποκωδικοποίηση, όπως φαίνεται και από την εικόνα, με την χρήση του προτύπου SVC ,είναι μια πιο πολύπλοκη διαδικασία, από

την απλουστευμένη που παρουσιάστηκε στην προηγούμενη παράγραφο. Μέσα στις επόμενες παραγράφους θα ξεκαθαρίσει πλήρως ο τρόπος λειτουργίας του SVC. Στην εικόνα 1, κάθε επίπεδο έχει ξεχωριστή ανάλυση. Κάθε επίπεδο ακολουθεί την κλασική λογική του H.264. Δηλαδή χρησιμοποιεί τις βασικές αρχές για inter –prediction και intra-prediction.Το νέο χαρακτηριστικό που εισάγεται είναι η τεχνική του inter-layer prediction.Σύμφωνα με αυτή νέα τεχνική που εισάγεται στον SVC, επαναχρησιμοποιούνται πληροφορίες από το προηγούμενο επίπεδο για να δημιουργηθεί το ιεραρχικά ανώτερο επίπεδο.

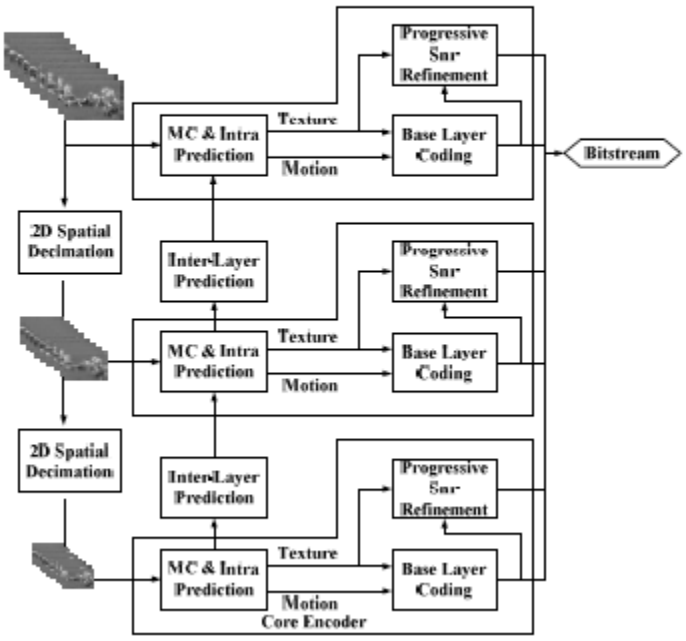


Figure 2

Τέτοιες πληροφορίες είναι για παράδειγμα τα motion vectors και τα residuals.

2.3 Inter-Layer Prediction

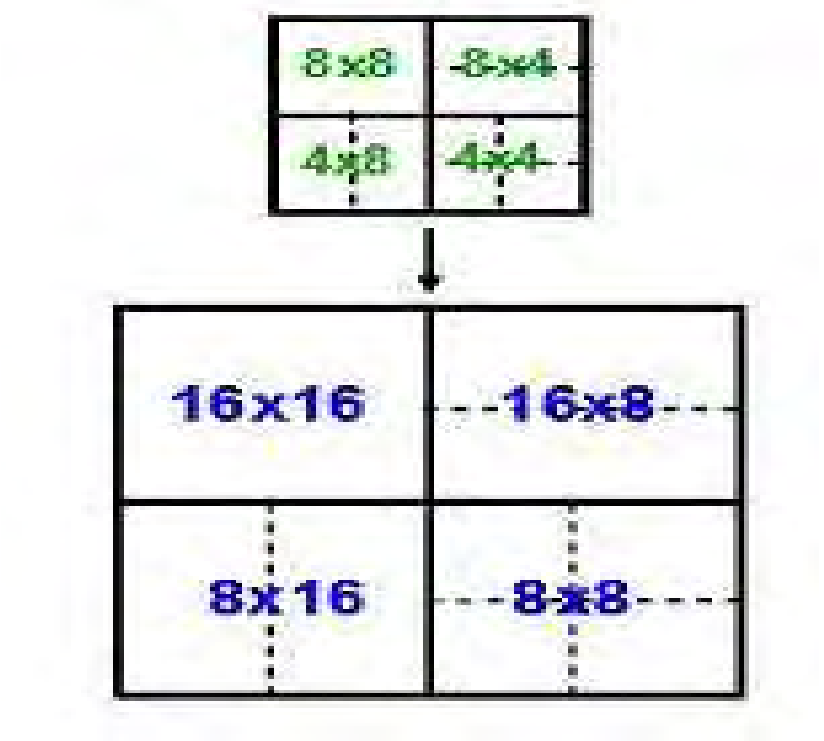
Όπως ήδη έχει αναφερθεί ο SVC εισάγει ένα νέο χαρακτηριστικό ,που ονομάζεται **Inter-Layer –Prediction**. Στις επόμενες παραγράφους γίνεται προσπάθεια να δοθεί μια ξεκάθαρη εικόνα για τον μηχανισμό αυτό. Ο κύριος στόχος του μηχανισμού αυτού είναι ή αξιοποίηση όλων των πληροφοριών από το base layer, έτσι ώστε να αποκωδικοποιηθεί με το καλύτερο δυνατό τρόπο μειώνοντας το σηματοθορυβικό λόγο του enhancement-layer. Στον SVC έχοντας ως στόχο την βελτιστοποίηση της κωδικοποίησης εισάγονται δύο αρχές του Inter-Layer-

Prediction:πρόβλεψη του τύπου του MB, συσχέτιση των παραμέτρων που χρησιμοποιούνται για το motion compensation και των υπολοίπων που χρησιμοποιούνται στο motion compensation. Ο μηχανισμός αυτός μπορεί να χρησιμοποιηθεί μόνο από το base layer για να παραχθεί το enhancement layer. Θα πρέπει η ανάλυση του base layer να είναι μικρότερη από αυτή του enhancement layer. Συνεχίζοντας θα περιγράψουμε τους 3 διαφορετικούς τύπους του μηχανισμού.

a) Inter Layer Motion Prediction:

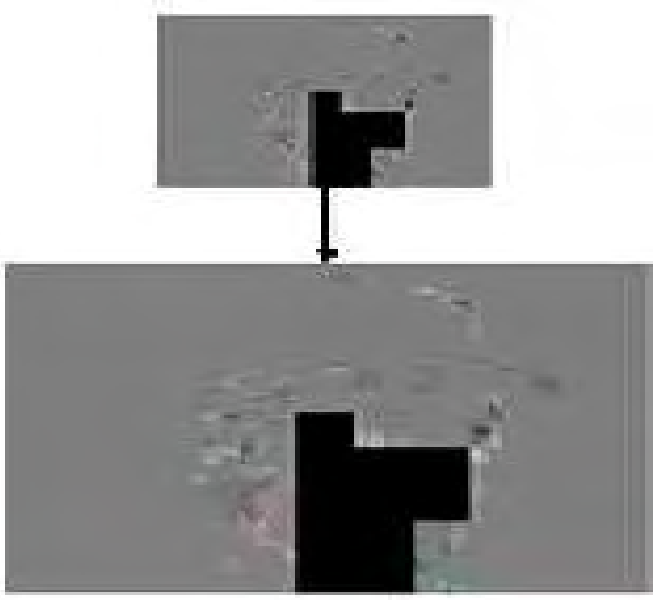
Για τα χωρικής κλιμάκωσης επίπεδα, και συγκεκριμένα για τα enhancement layers εισάγεται ένας νέος τύπος macroblock. Ο νέος τύπος αυτός διαχωρίζεται από τους υπόλοιπους, ορίζοντας μια νέα μεταβλητή που ονομάζεται base mode flag. Για αυτό τον συγκεκριμένο τύπο, η μόνη πληροφορία που μεταφέρεται από το ένα επίπεδο στο επόμενο είναι τα υπόλοιπα, που προκύπτουν κατά την διαδικασία του motion estimation. Όταν η μεταβλητή αυτή έχει την τιμή 1 το αντίστοιχο 8x8 block στο επίπεδο αναφοράς (base layer), σημειώνεται ως **intra coded macroblock** και το macroblock του enhancement layer αποκωδικοποιείται σαν **inter-layer intra-prediction,** που θα εξηγηθεί παρακάτω. Στην περίπτωση που ένα macroblock μαρκάρεται ως inter-coded στο base layer,τότε και στο enhancement layer to macroblock αυτό έχει τον ίδιο τύπο. Σε αυτή την περίπτωση όλες οι πληροφορίες που χρειάζεται το enhancement layer για να αποκωδικοποιηθεί, όπως τα motion vectors και τα υπόλοιπα προέρχονται από το base layer. Ο προσδιορισμός του macroblock στο enhancement layer, προέρχεται από την διαδικασία upsampling, του macroblock στο base layer. Στην πραγματικότητα από την διαδικασία upsampling ενός block μέσα στο macroblock. Αν το 8x8 block στο base layer δεν υποδιαιρείται σε μικρότερα blocks τότε αυτό εξακολουθεί να συμβαίνει και στο enhancement layer. Σε όλες τις άλλες περιπτώσεις κάθε sub-block MxN ενός 8x8 block αντιστοιχεί σε ένα sub-block (2*M)x(2*N) στο ανώτερο ιεραρχικά επίπεδο. Τα motion vectors από το base layer χρησιμοποιούνται στο enhancement layer, πολλαπλασιασμένα με το συντελεστή 2. Επιπλέον εκτός του τύπου macroblock που περιγράφεται παραπάνω, ο SVC δίνει την δυνατότητα να χρησιμοποιηθούν motion vectors των 8x8 blocks από το base layer στο ανώτερο ιεραρχικά επίπεδο τα οποία έχουν υποστεί κλιμάκωση ένα συγκεκριμένο συντελεστή. Επιπλέον μια νέα μεταβλητή εισάγεται, η **motion prediction flag**. Η μεταβλητή αυτή ελέγχει για κάθε τύπο block πχ 8x8,8x16,16x8, πότε θα χρησιμοποιηθεί η διαδικασία του motion vector predictor και πότε

όχι. Όταν η μεταβλητή αυτή έχει την τιμή ένα ,τότε η διαδικασία του motion estimation και αντίστοιχα του motion compensation, δε λαμβάνει χώρα στο enhancement layer αλλά χρησιμοποιούνται για την κωδικοποίηση και αποκωδικοποίηση τα motion vectors που έχουν υποστεί κλιμάκωση στο προηγούμενο επίπεδο. Αντίθετα όταν είναι 0 ακολουθείται η γνωστή διαδικασία που περιγράφεται στον H.264. Στην παρακάτω εικόνα παρουσιάζεται η αντιστοίχηση των blocks των base και enhancement layer.



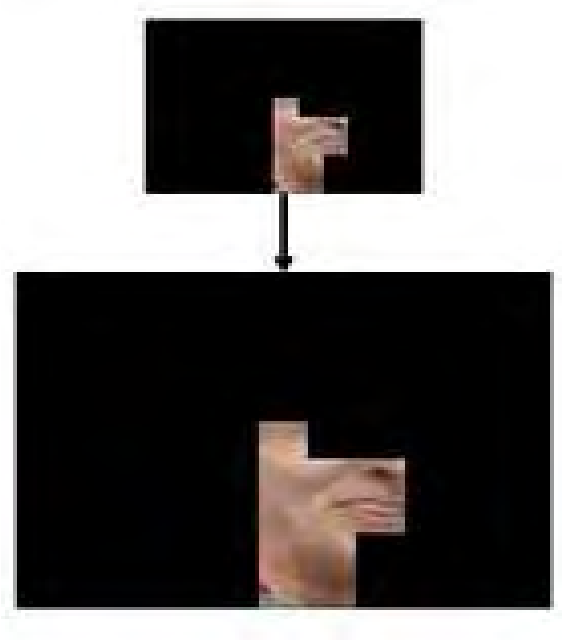
b) Inter Layer Residual Prediction

Η μέθοδος αυτή εφαρμόζεται σε όλα τα macroblock που έχουν κωδικοποιηθεί σαν inter. Ομοίως με πριν, εισάγεται μια νέα μεταβλητή που ελέγχει την διαδικασία αυτή. Η μεταβλητή αυτή ονομάζεται **base mode flag**. Η μεταβλητή αυτή χρησιμοποιείται για τα spatial enhancement layers. Όταν η μεταβλητή αυτή είναι 1, τα υπόλοιπα που περιέχονται στο 8x8 block στο base layer φιλτράρονται μέσα από την διαδικασία της διγραμμικής παρεμβολής. Αυτή η διαδικασία χρησιμοποιείται ως μέθοδος upsampling. Τα νέα υπόλοιπα που προκύπτουν χρησιμοποιούνται από το enhancement layer.Οι παρακάτω εικόνα απεικονίζει σχηματικά την διαδικασία residual upsampling



c) Inter Layer Intra Prediction

Η τελευταία περίπτωση αφορά τα intra-macroblocks. Στην περίπτωση που η μεταβλητή **base mode flag**, που περιγράψαμε πιο πάνω, είναι ίση με 1 και ο τύπος του macroblock είναι intra, τότε χρησιμοποιείται η παρούσα μέθοδος. Η μέθοδος αυτή είναι μια διαδικασία upsampling που χρησιμοποιεί ειδικά φίλτρα, όπως όλες οι μέθοδοι upsampling. Για τα στοιχεία που αναπαριστούν την φωτεινότητα (luma), χρησιμοποιούμε ένα ειδικό φίλτρο που ονομάζεται 4-tap filter. Για τα στοιχεία που αναπαριστούν το χρώμα (chroma), χρησιμοποιείται διγραμμική παρεμβολή. Η διαδικασία του φιλτραρίσματος που λαμβάνει χώρα στα όρια του macroblock χρησιμοποιεί τα pixels από γειτονικά intra macroblocks. Όταν τα γειτονικά macroblocks δε είναι intra τα απαιτούμενα pixels, παράγονται από ένα αλγόριθμο επέκτασης των ορίων του macroblock. Για να αποφθεχθούν ατέλειες στο καρέ, δηλαδή να προστεθεί θόρυβος, μετά την διαδικασία επέκτασης χρησιμοποιείται το deblocking filter. Η παρακάτω εικόνα απεικονίζει την διαδικασία intra-upsampling.



2.4Σύνδετικός κρίκος μεταξύ των επιπέδων

Μέχρι τώρα έχει περιγραφεί σε γενικές γραμμές πως λειτουργεί ο SVC, χωρίς να έχουν αναφερθεί οι περιορισμοί του βασικού layer και του εξαρτώμενου layer. Στην πραγματικότητα υπάρχουν εξαρτήσεις. Το βασικό layer, το οποίο το ονομάζουμε base layer και το εξαρτώμενο layer, το οποίο το ονομάζουμε enhancement layer, μπορούν να μελετηθούν σαν 2 σύνολα. Αυτά τα δύο σύνολα θα πρέπει να είναι ανεξάρτητα σύνολα. Αυτό σημαίνει ότι το base layer δε εξαρτάται από το enhancement layer, αλλά όχι το αντίστροφο. Το χαρακτηριστικό αυτό είναι ιδιαίτερα σημαντικό, διότι ακόμα και να χαθεί ή να καταστραφεί το enhancement layer ο αποκωδικοποιητής θα μπορέσει να αποκωδικοποιήσει τουλάχιστον το base layer.

2.5 Περιγραφή των 3 διαφορετικών τρόπων κλιμάκωσης

Ήδη στις προηγούμενες ενότητες έγινε αναφορά στον διαχωρισμό των επιπέδων και στους διαφορετικούς τρόπος κλιμάκωσης. Στην ενότητα αυτή θα περιγραφούν αναλυτικά οι τρεις διαφορετικοί τρόποι κλιμάκωσης. Οι τρείς τρόποι κλιμάκωσης όπως ήδη έχουν αναφερθεί είναι:

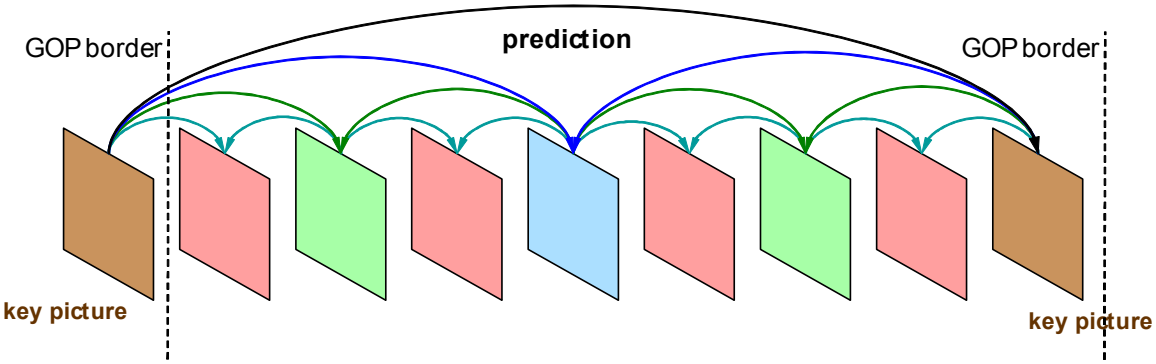
- Χρονική Κλιμάκωση (Temporal Scalability)
- Χωρική Κλιμάκωση (Spatial Scalability)
- Κλιμάκωση με βάση την ποιότητα (SNR Scalability)

2.5.1 Χρονική Κλιμάκωση

Η Χρονική κλιμάκωση, λαμβάνει χώρα όταν αλλάζει ο ρυθμός απεικόνισης των καρέ στην οθόνη. Για να πετύχουμε χρονική κλιμάκωση, αρκεί να "πετάξουμε κάποια καρέ (frame)". Κάθε frame μπορεί να έχει 3 τύπους .

- **I**: Περιέχει την περισσότερη συμπιεσμένη πληροφορία, αλλά δεν εξαρτάται από κανένα άλλο frame
- **P:** Τα frame αυτά εξαρτώνται μόνο από τα προηγούμενα frames,**I** και **P**
- **B:** Τα frame αυτού του τύπου εξαρτώνται από προηγούμενα frames, αλλά και από επόμενα frames, που έχουν τύπο **I** ή **P**.Δηλαδή 2 **B** frames είναι ανεξάρτητα μεταξύ τους .

Γίνεται προφανές ότι μπορεί να μην γίνει αποκωδικοποίηση των **B** frames, χωρίς να αλλοιωθεί η ποιότητα της εικόνας. Η λειτουργία αυτή εκσωματώνεται και στον H.264. Όμως ο SVC δε αγνοεί τα B frames, όταν θέλει να πετύχει χρονική κλιμάκωση. Η διαδικασία που ακολουθεί, είναι να κωδικοποιήσει τα ανεξάρτητα μεταξύ τους **B** frames σε ανώτερο ιεραρχικά επίπεδο. Ο αποκωδικοποιητής αποκωδικοποιεί και τα δύο επίπεδα (το base Layer και το enhancement layer). Τα καρέ που προκύπτουν τα αποθηκεύει σε μια ενδιάμεση προσωρινή μνήμη, και στο τέλος κάνει αναδιάταξη για να παραχθεί η σωστή εικόνα. Το πλήθος των διαφορετικών επιπέδων εξαρτάται από μια μεταβλητή που ονομάζεται GOP(Group Of Pictures). **Group Of Pictures** ονομάζεται το σύνολο των καρέ που βρίσκονται ανάμεσα στο **key-picture** (τα καρέ από τα οποία εξαρτώνται τα **B** frames) του τρέχοντος GOP, και του **key-picture** του προηγούμενου GOP. Στην επόμενη εικόνα παρουσιάζεται μια ιεραρχική δομή 8 εικόνων (καρέ).



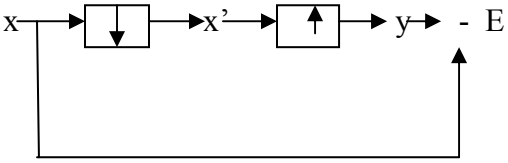
Στην παραπάνω εικόνα το μεσαίο καρέ(μπλε) για να προληφθεί χρησιμοποιεί ως αναφορά τα καρέ γύρω από αυτό. Εξαρτάται μόνο από το **key-picture**. Το καρέ αυτό μαζί με τα key-pictures αποτελούν την επομένη υψηλότερης χρονική ανάλυση .Οι εικόνες με το καφέ χρώμα αποτελούν το base layer. Οι εικόνες με το μπλε χρώμα αποτελούν το 1ο χωρικό επίπεδο. Οι εικόνες με το πράσινο χρώμα αποτελούν το 2ο επίπεδο χρονικής κλιμάκωσης και με το ροζ το 3ο. Όπως φαίνεται από την εικόνα είναι ανεξάρτητες μεταξύ τους και η πρόβλεψη τους γίνεται χρησιμοποιώντας ως καρέ αναφοράς τα καρέ του προηγούμενου επιπέδου χρονικής κλιμάκωσης. Είναι φανερό ότι αυτή η ιεραρχική κλιμάκωση έμφυτα περιέχει χρονική κλιμάκωση.

2.5.2 Χωρική Κλιμάκωση

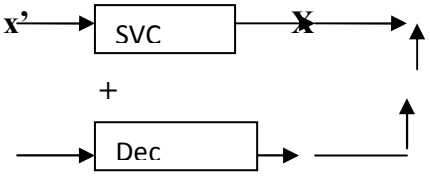
Χωρική κλιμάκωση σημαίνει, αλλαγή των διαστάσεων ενός καρέ video. Στην γενική περίπτωση η διαδικασία που ακολουθείται για να δημιουργηθεί ένα νέο καρέ που έχει κλιμακωθεί στο χώρο είναι ως εξής: Δεχόμαστε σαν είσοδο ένα καρέ βίντεο υψηλής ανάλυσης, έστω X. Το καρέ αυτό το επεξεργαζόμαστε με ειδικά φίλτρα, έτσι ώστε να μειωθούν οι διαστάσεις σε υποπολλαπλάσιο του 2 και δημιουργείται ένα νέο καρέ X'. Η διαδικασία αυτή ονομάζεται downsampling. Στην συνέχεια, το νέο καρέ video που έχει προκύψει, το επεξεργαζόμαστε ξανά με ειδικά φίλτρα για να ανακτήσουμε ένα καρέ με τις ίδιες διαστάσεις με το αρχικό, έστω Y. Στην επόμενη φάση, αφαιρούμε τα 2 καρέ X και Y, και δημιουργούμε ένα νέο καρέ που περιέχει σαν πληροφορία την διαφορά τους, έστω E. Τέλος το καρέ X' και το καρέ E κωδικοποιούνται και στέλνονται στο αποκωδικοποιητή. Ο αποκωδικοποιητής αποκωδικοποιεί τα δυο καρέ και τα προσθέτει .

Τα παρακάτω σχήματα δείχνουν την διαδικασία που περιγράψαμε.

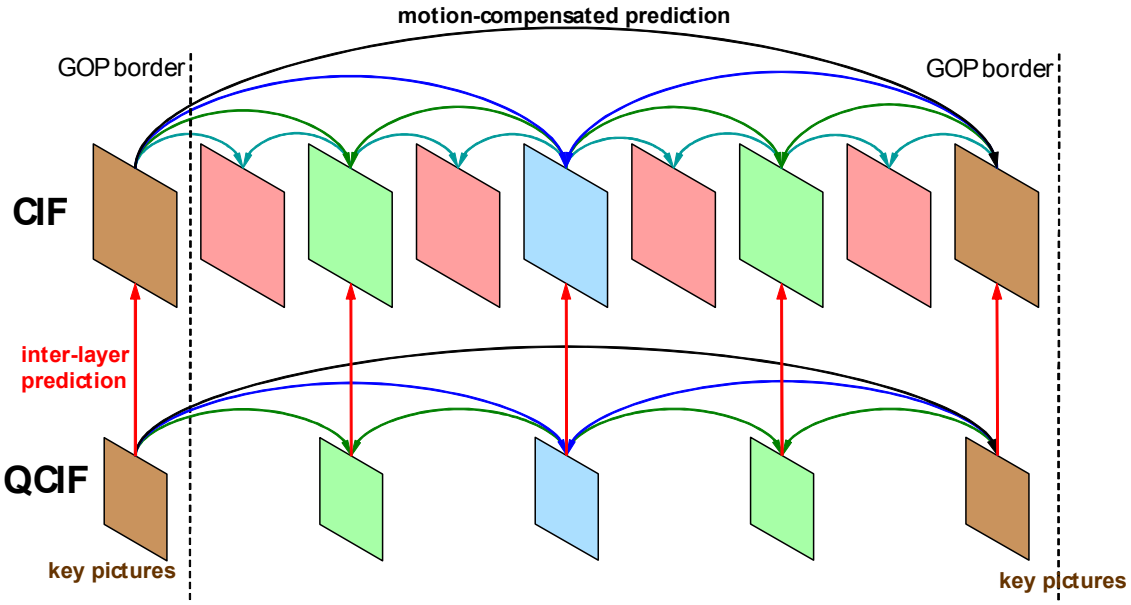
Σχήμα 1, Αποκωδικοποιητής



Σχήμα ,κωδικοποιητή



Στην επόμενη εικόνα φαίνεται μια ιεραρχική δομή 2 επιπέδων χωρικής κλιμάκωσης.



Τα κόκκινα βέλη παρουσιάζουν την χρήση του καινούριου χαρακτηριστικού που εισάγει ο SVC του "inter-layer-prediction". Το χαρακτηριστικό αυτό μπορεί να χρησιμοποιηθεί μόνο ανάμεσα στο base layer και στο enhancement layer. Όπως φαίνεται και από την εικόνα στο ανώτερο ιεραρχικά επίπεδο η πρόβλεψη των καρέ που δε είναι τύπου **I,** γίνεται χρησιμοποιώντας τις γνωστές μεθόδους του motion-compensated,από τον **H.264.**

2.5.3 Κλιμάκωση ποιότητας

Η κλιμάκωση ποιότητας μπορεί να θεωρηθεί σαν μια ειδική περίπτωση της χωρικής κλιμάκωσης, για καρέ βίντεο με την ίδια χωρική ανάλυση στο base layer και στο enhancement layer. Αυτή η περίπτωση υποστηρίζεται από το γενικό μηχανισμό λειτουργίας χωρικής κλιμάκωσης και αναφέρεται ως coarse-gain quality scalable coding(CGS). Ο ίδιος μηχανισμός που χρησιμοποιείται για inter-layer prediction στην χωρική ανάλυση χρησιμοποιείται και για την κλιμάκωση ποιότητα. Όμως δε χρησιμοποιούνται οι μηχανισμοί upsampling που περιγράφθηκαν παραπάνω αλλά καθώς η διαδικασία φιλτραρίσματος των κωδικοποιημένων ως **intra** καρέ αναφοράς του base layer(deblocking filter). Επιπλέον εκτός από την δημιουργία πολλών επιπέδων με διαφορετική ποιότητα, παρέχεται και η δυνατότητα επιλογής διαφορετικών ρυθμών μετάδοσης (bitrate) στα επίπεδα κλιμάκωσης. Όμως ο αριθμός των διαφορετικών ρυθμών μετάδοσης είναι μοναδικός για κάθε επίπεδο. Αυτή η δυνατότητα ονομάζεται MGS(Medium-Gain Scalability).

2.6 Διαφορές H.264 και SVC

Ο SVC περιέχει πολλά εργαλεία για την μείωση του θορύβου στα καρέ ενός video κατά την διαδικασία κωδικοποίησης, παρέχοντας κωδικοποιημένο video που η ποιότητα του πλησιάζει αυτή του single-layer-coding.Οι βασικότερες διαφορές είναι:

a) Η δυνατότητα της δημιουργίας πολλών χρονικά κλιμακωμένων επιπέδων, βελτιώνοντας την κωδικοποίηση, ενώ ταυτόχρονα αυξάνεται η αποτελεσματικότητα της κλιμάκωσης στο χώρο και της ποιότητας.

b) Εισαγωγή νέων μεθόδων για inter layer prediction, όσο αφορά τον υπολογισμό των motion vectors, και των υπολοίπων βελτιώνοντας την χωρική κλιμάκωση και την κλιμάκωση ποιότητας .

c) Η εισαγωγή της έννοιας **key-picture** και των ιεραρχικών επιπέδων που βασίζονται στην έννοια αυτή.

d) Single compensation loop decoding, για τα spatial και quality layers, παρέχοντας στον αποκωδικοποιητή πολυπλοκότητα, εφάμιλλη του single-layer-decoding.

Κεφάλαιο 3

3.1 Περιγραφή τεχνικών χαρακτηριστικών του υπολογιστικού συστήματος στο οποίο έγινε η βελτιστοποίηση του SVC decoder

Η βελτιστοποίηση του SVC πραγματοποιήθηκε σε υπολογιστικό σύστημα που διαθέτει αρχιτεκτονική γενικού σκοπού x86. Συγκεκριμένα η υλοποίηση έγινε σε αρχιτεκτονική intel 45 nm,συγκεκριμένα σε Intel mobile core 2 duo (μοντέλο T8100,64 bit), με επεξεργαστή συγχρονισμένο στα 2.1 GHz (figure 3). Η συνολική μνήμη του υπολογιστικού συστήματος είναι 2048 MB και συνολικής συχνότητας 667 MHz(figure 4). Επιπρόσθετα το συγκεκριμένο μηχάνημα διαθέτει L2-cache, συνολικής χωρητικότητας 3072 KB και τέλος διαθέτει 2 L1-cache μεγέθους 32KB για τα δεδομένα και 2 L1-cache ίδιου μεγέθους για τις εντολές (figure 5). Επίσης η συγκεκριμένη αρχιτεκτονική υποστηρίζει χρήση διανυσματικών πράξεων . Ο τύπος των διανυσματικών πράξεων που υποστηρίζει είναι MMX,SSE2,SSE3,SS4.1.

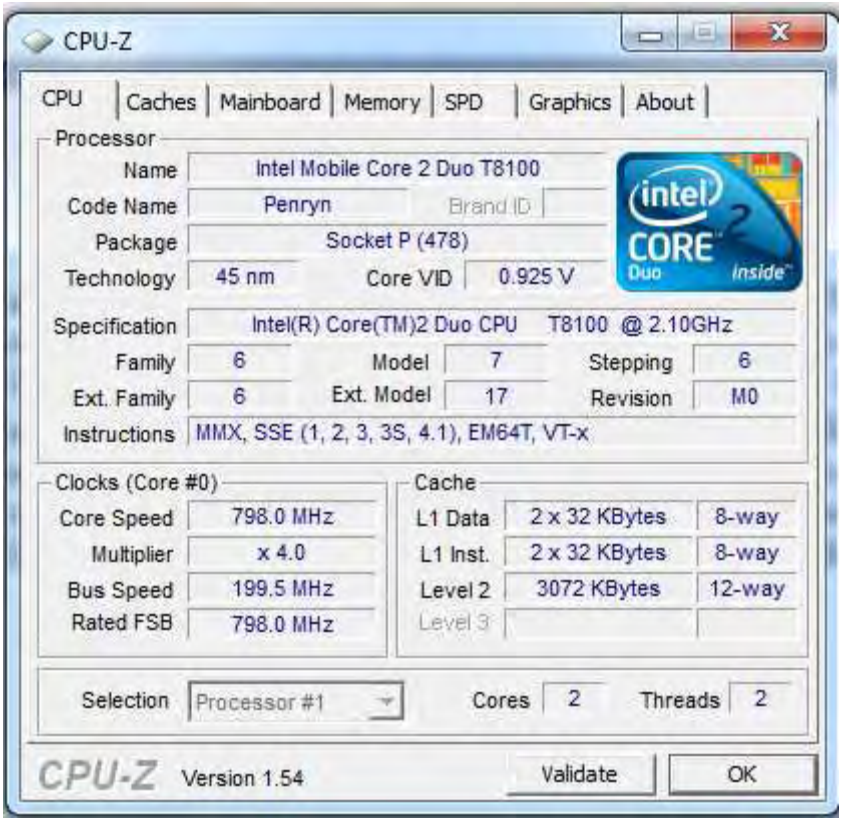


Figure 3

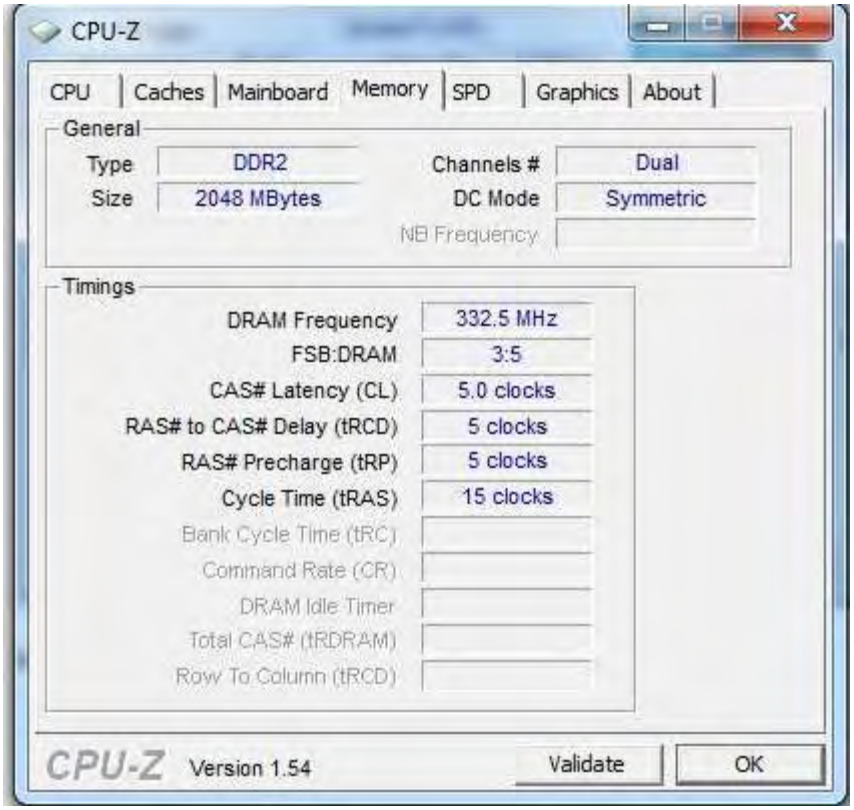

CPU-Z
CPU
Caches
Mainboard
Memory
SPD
Graphics
About
General
Type
DDR2
Channels #
Dual
Size
2048 MBytes
DC Mode
Symmetric
NB Frequency
Timings
DRAM Frequency
332.5 MHz
FSB:DRAM
3:5
CAS# Latency (CL)
5.0 clocks
RAS# to CAS# Delay (tRCD)
5 clocks
RAS# Precharge (tRP)
5 clocks
Cycle Time (tRAS)
15 clocks
Bank Cycle Time (tRC)
Command Rate (CR)
DRAM Idle Timer
Total CAS# (tRDRAM)
Row To Column (tRCD)
CPU-Z
Version 1.54
Validate
OK

Figure 4

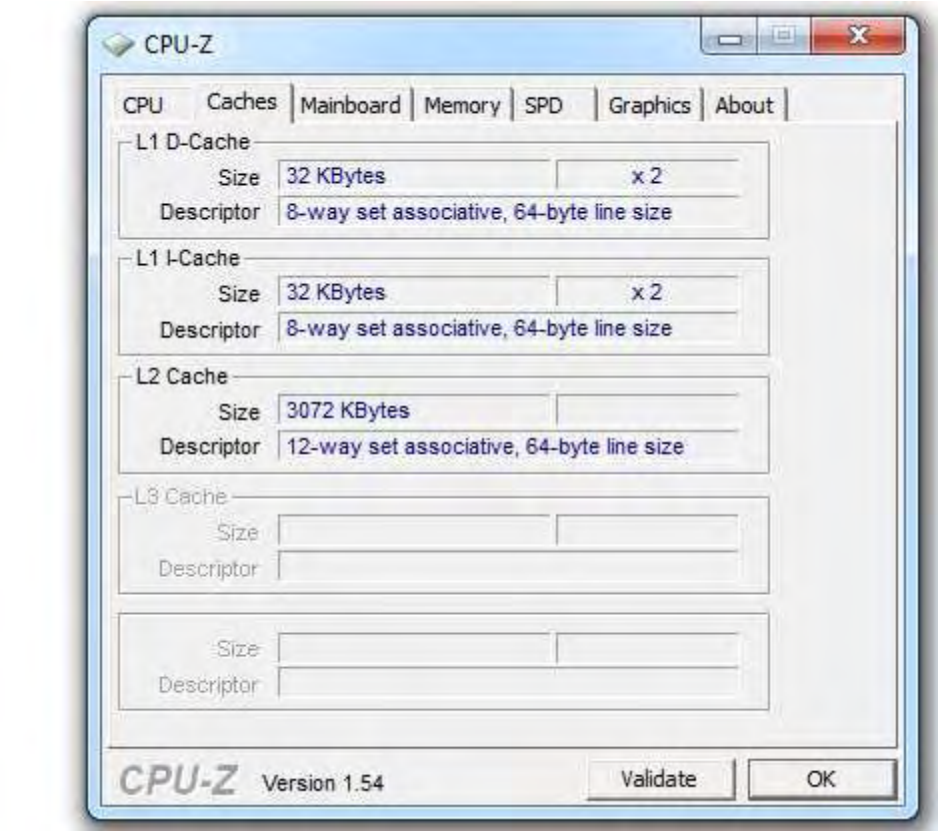


Figure 5

3.2 Περιγραφή εργαλείου vtune

Για τις περισσότερες δημοφιλείς αρχιτεκτονικές υπάρχουν πληθώρα εργαλείων τα οποία χρησιμοποιούνται για να δώσουν στο προγραμματιστή σαφή εικόνα για τα τμήματα του κώδικα που καταναλώνουν τους περισσότερους κύκλους μηχανής. Για τις αρχιτεκτονικές της Intel το ποιό δημοφιλές εργαλείο είναι το "vtune performance analyzer". Το εργαλείο αυτό χρησιμοποιήθηκε σε όλη την διάρκεια εκπόνησης της διπλωματικής εργασίας. Το vtune παρέχει μια ολοκληρωμένη ανάλυση της απόδοσης μιας εφαρμογής για αρχιτεκτονικές IA-32,Intel x64,Itanium processor families. Η ανάλυση μιας εφαρμογής γίνεται με την μέθοδο της δειγματοληψίας. Παρέχονται δυο τρόποι δειγματοληπτικού ελέγχου. Ο πρώτος τρόπος ονομάζεται time-based-sampling και ο δεύτερος τρόπος ονομάζεται event –based –sampling.

Η μέθοδος του time-based-sampling χρησιμοποιεί το ρόλοι του υπολογιστικού συστήματος και ανά συγκεκριμένες χρονικές περιόδους στέλνει interrupts στον επεξεργαστή και εκτελεί δειγματοληψία για το σύνολο των εντολών που εκτελούνται την συγκεκριμένη χρονική στιγμή.

Για παράδειγμα αν θέσουμε την περίοδο ίση με 1 ms, τότε κάθε 1 ms θα γίνεται η διαδικασία που περιγράψαμε πιο πάνω. Όμως η παραπάνω μέθοδος είναι η μόνη που υποστηρίζεται από όλους τους παλιότερους τύπους επεξεργαστών. Συγκεκριμένα για ορισμένα φορητά υπολογιστικά σύστημα τα οποία χρησιμοποιούν επεξεργαστή Intel Pentium 4, αφού οι επεξεργαστές αυτοί δε υποστηρίζουν την λειτουργία counter-interrupt overflow, που είναι απαραίτητη για την δεύτερη μέθοδο, που θα περιγραφεί αμέσως μετά. Επιπλέον για επεξεργαστές της σειράς M που έχουν ενεργοποιημένη την λειτουργία APICs. Για όλους τους άλλους τύπους επεξεργαστών είναι μέθοδος αυτή δε είναι η προκαθορισμένη.

Η μέθοδος του event-based-sampling, όπως ειπώθηκε είναι η προκαθορισμένη μέθοδος για τους περισσότερους σύγχρονους επεξεργαστές. Η συγκεκριμένη μέθοδος συγκεντρώνει δείγματα από την τρέχουσα ενεργή εντολή, μετά από ένα συγκεκριμένο αριθμό event που συμβαίνουν στον επεξεργαστή. Εκτός από τα time-events που είναι παρόμοια με την μέθοδο υπάρχει μια πληθώρα events που υποστηρίζει το vtune. Για παράδειγμα events , για λανθασμένες προβλέψεις διακλαδώσεων καθώς και events για cache misses. Το vtune παρέχει μια πληθώρα προκαθορισμένων ρυθμών για τα διάφορα event που χρησιμοποιούνται από το εργαλείο για να δώσουν μια πλήρη ανάλυση των σημείων που καθυστέρει μια εφαρμογή. Η μέθοδος του event-based-upsampling βασίζεται στην εμφάνιση των event που προκαλούνται σε ένα επεξεργαστή. Η συχνότητα με την οποία γίνεται η συλλογή δεδομένων, βασίζεται στο πόσο συχνά ένα event προκαλείται από την εφαρμογή που τρέχει. Η προκαθορισμένη επιλογή είναι τα clockticks events, με προκαθορισμένη τιμή 1ms. Αυτό σημαίνει ότι κάθε 1 ms η εφαρμογή προκαλεί interrupts και συλλέγονται οι απαραίτητες πληροφορίες.

Η διαφορές ανάμεσα στις 2 αυτές μεθόδους είναι το γεγονός, ότι η πρώτη μέθοδος αποτελεί ένα και μοναδικό event που λαμβάνει χώρα με συγκεκριμένη συχνότητα και εξαρτάται από το ρολόι του επεξεργαστή. Όμως η δεύτερη μέθοδος λειτουργεί σε διαφορετική λογική. Τα events του επεξεργαστή συμβαίνουν σε ένα συγκεκριμένο ρυθμό που καθορίζεται από την εκτέλεση της εφαρμογής. Αν θέλουμε να κάνουμε ανάλυση με βάση τα clockticks είναι ισοδύναμο με το να χρησιμοποιούμε ένα εξωτερικό ρολόι που προκαλεί διακοπές με συγκεκριμένο ρυθμό.

Ένα άλλο χαρακτηριστικό του vtune είναι το “ counter monitor”. Όταν ενεργοποιηθεί το χαρακτηριστικό αυτό παρουσιάζει τους πόρους του συστήματος που καταναλώνονται κατά την διάρκεια εκτέλεσης της εφαρμογής..

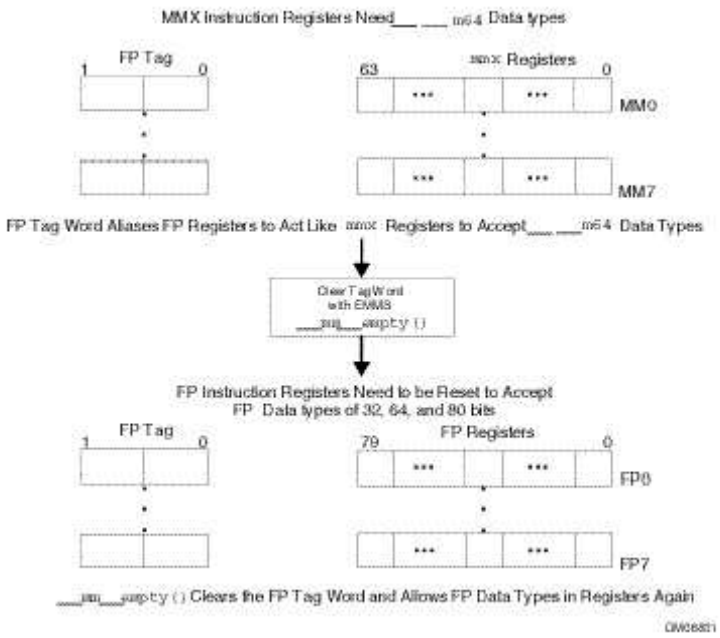
Τέλος ένα επιπλέον σημαντικό χαρακτηριστικό που παρέχει το vtune είναι το call graph. Το call graph profiling συλλέγει πληροφορίες για το διάγραμμα ροής μιας εφαρμογής. Παρέχει πληροφορίες για την συχνότητα κλήσης μιας συνάρτησης, για το πλήθος των συναρτήσεων που καλεί μια εφαρμογή καθώς επίσης και για το σύνολο των συναρτήσεων που καλεί η συγκεκριμένη συνάρτηση.

3.3Διανυσματικές εντολές

Στη παρούσα ενότητα θα περιγραφτούν οι διανυσματικές εντολές που χρησιμοποιούνται από τις αρχιτεκτονικές της Intel.Υπάρχουν 2 τύποι διανυσματικών εντολών .Οι πρώτος τύπος επιτρέπει πράξεις με αριθμούς μήκους 64 bits και ο δεύτερος τύπος πράξεις με 128 bits. Ο πρώτος τύπος δε υποστηρίζεται από αρχιτεκτονικές Intel x64.

3.3.1 MMX instructions

Το σύνολο εντολών MMX, εισάγει ένα νέο τύπο δεδομένων τον **__m64 type** .Στο νέο σύνολο, ορίζονται 57 διανυσματικές πράξεις. Το σύνολο των καταχωρητών που μπορούν να χρησιμοποιηθούν για MMX εντολές είναι οχτώ. Τα ονόματα αυτών ξεκινούν με το αναγνωριστικό mm .Δηλαδή mm0,mm1,mm2,mm3,mm4,mm5,mm6,mm7. Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι οι καταχωρητές αυτοί δε έχουν σχεδιαστεί εκ νέου στις αρχιτεκτονικές Intel, αλλά για τις διανυσματικές εντολές χρησιμοποιούνται οι ίδιοι καταχωρητές για εκτέλεση πράξεων με δεκαδικούς. Για αυτό το σκοπό χρησιμοποιείται ένα flag για να αλλάξει την κατάσταση των καταχωρητών για να χρησιμοποιηθούν σε οποιαδήποτε χρήση. Αν στην εφαρμογή που έχει υλοποιηθεί δε εκτελούνται πράξεις με δεκαδικούς μετά τις διανυσματικές εντολές, δε χρειάζεται να αλλάξει η κατάσταση των καταχωρητών. Η μετάβαση από την μια κατάσταση στην άλλη γίνεται με την χρήση των EMMS εντολών. Σε περίπτωση που εκτελεστούν δεκαδικές πράξεις μετά από MMX instructions χωρίς να “αδειάσει” η κατάσταση των καταχωρητών τότε υπάρχει ο κίνδυνος, να οδηγηθεί η εφαρμογή σε ανεπιθύμητα αποτελέσματα. Στην εικόνα που ακολουθεί μπορεί να γίνει κατανοητό πως χρησιμοποιούνται οι καταχωρητές για τους τύπους float για εκτέλεση διανυσματικών πράξεων μήκους 64 –bits.



3.3.2 SSE Instructions

Επιπλέον στις καινούργιες αρχιτεκτονικές εισάγεται ένας νέος τύπος δεδομένων για διανυσματικές πράξεις που έχει το αναγνωριστικό **__m128**. Ο τύπος αυτός υποστηρίζει πράξεις με αριθμούς μήκους 128 bits. Σε αντίθεση με τους καταχωρητές MMX υποστηρίζεται από όλες τις αρχιτεκτονικές Intel. Το σύνολο των εντολών που ορίζονται σε αυτό το νέο τύπο διανυσματικών πράξεων αριθμείται στις 70. Για την εκτέλεση διανυσματικών εντολών SSE, σχεδιάστηκαν 8 νέοι καταχωρητές. Οι καταχωρητές αυτοί έχουν τα αναγνωριστικά xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7. Εκτός από αυτούς τους 8 αυτούς καταχωρητές για τα λειτουργικά συστήματα 64 bits χρησιμοποιούνται και 8 επιπλέον καταχωρητές που ξεκινούν από τον xmm8 έως xmm15. Στο νέο αυτό σύνολο καταχωρητών κάθε καταχωρητής μπορεί να χρησιμοποιηθεί με κάθε ένα από τους ακόλουθους τρόπους:

- 4 x32 bit απλής ακρίβειας αριθμών τύπου floating point
- 2 x64 bit διπλής ακρίβειας αριθμών τύπου floating point
- 4-x2 bit αριθμών τύπου integer
- 8 x16 bit αριθμών τύπου integer
- 16x8 bit αριθμών τύπου integer

Επιπρόσθετα οι καταχωρητές υποστηρίζουν διαφορετικές εντολές για προσημασμένους και μη –προσημασμένους αριθμούς. Όπως ήδη αναφέρθηκε οι SSE εντολές υποστηρίζουν πράξεις με δεκαδικούς αριθμούς. Η εμφάνιση των καταχωρητών αυτών έλυσε ένα σημαντικό πρόβλημα που είχαν οι MMX καταχωρητές. Δηλαδή η επαναχρησιμοποίηση των ίδιων καταχωρητών για δεκαδικές πράξεις από τις εντολές MMX που δεν έδινε την δυνατότητα παράλληλης επεξεργασίας δεκαδικών και διανυσματικών πράξεων. Το πρόβλημα αυτό λύθηκε χρησιμοποιώντας νέους καταχωρητές για διανυσματικές πράξεις με την χρησιμοποίηση SSE εντολών . Το χαρακτηριστικό αυτό σε συνάρτηση με το γεγονός ότι μπορούν ταυτόχρονα να εκτελεστούν περισσότερες εντολές σε σχέση με τους MMX καταχωρητές, έχουν κάνει την χρήση των καταχωρητών πιο συχνή. Όμως θα πρέπει να ο προγραμματιστής να λάβει υπόψη ότι τα δεδομένα θα πρέπει να είναι ευθυγραμμισμένα σε μνήμη πολλαπλάσιο του 16. Σε καινούργιες αρχιτεκτονικές i7, i5, η απόδοση μπορεί να μειωθεί αισθητά αν χρησιμοποιούνται δεδομένα από μή ευθυγραμμισμένες περιοχές μνήμης αλλά η εφαρμογή ολοκληρώνει την ζωή της ομαλά. Όμως σε παλιότερες αρχιτεκτονικές, όπως Quad Core, Core 2 Duo, ή χρήση μη ευθυγραμμισμένων διευθύνσεων μνήμης οδηγεί σε σφάλμα στην μνήμη και διακοπή εκτέλεσης της εφαρμογής.

Κεφάλαιο 4

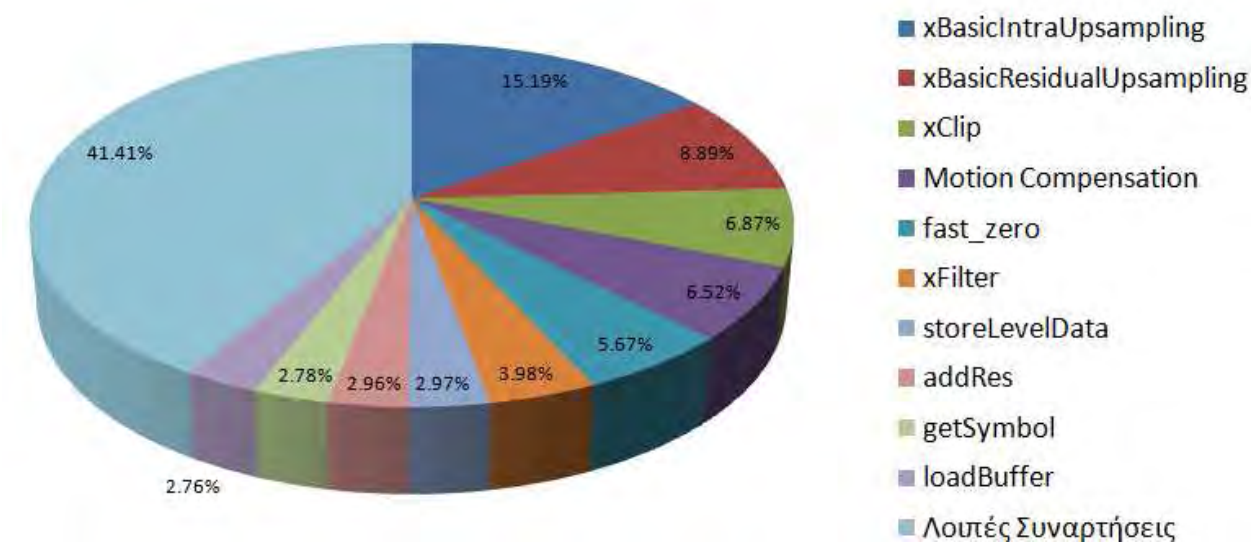
Στο κεφάλαιο αυτό θα γίνει εκτενής αναφορά στο τρόπο με τον οποίο παραλληλοποιήθηκαν τα δεδομένα για να εκτελεστούν διανυσματικές πράξεις.

4.1 Περιγραφή των benchmark που χρησιμοποιήθηκαν και των πόρων που κατανάλωνε ο αρχικός κώδικας

Στην εργασία αυτή χρησιμοποιήθηκε ένα αρχείο yuv ανάλυσης 720x480, 300 καρέ. Από αυτό το αρχείο προέκυψε ένα SVC bitstream, που χρησιμοποιήθηκε από τον αποκωδικοποιητή. Ο συνολικός ρυθμός μετάδοσης των δεδομένων για το υψηλότερο ιεραρχικά layer ήταν 3.992 Mbits/sec.Συνολικά χρησιμοποιήθηκαν 13 layers, 3 επίπεδα για τις διαφορετικές χωρικές αναλύσεις και κάθε ένα από αυτά είχε 4 επίπεδα διαφορετικής χρονικής κλιμάκωσης ,με επιπλέον 1 επίπεδο για την χωρική ανάλυση 720x480. Στο παρακάτω πίνακα παρουσιάζονται τα χαρακτηριστικά του κάθε επιπέδου.

Layer	Resolution	Frame Rate	BitRate	MinBitRate
0	192x128	1.8750	80.40	80.40
1	192x128	3.7500	108.10	108.10
2	192x128	7.500	142.20	142.20
3	192x128	15.000	173.90	173.90
4	368x240	1.8750	335.90	335.90
5	368x240	3.7500	493.10	493.10
6	368x240	7.500	693.10	693.10
7	368x240	15.00	895.50	895.50
8	720x480	1.8750	1186.30	1186.30
9	720x480	3.750	1725.00	1725.00
10	720x480	7.500	2481.00	2481.00
11	720x480	15.000	3384.00	3384.00
12	720x480	30.00	3992.00	3992.00

Επιπρόσθετα ο συνολικός χρόνος που χρειαζόταν για να ολοκληρωθεί με επιτυχία η αποκωδικοποίηση του παραπάνω bitstream ,κυμαινόταν από 29 μέχρι 30 δευτερόλεπτα, που αντιστοιχεί σε 60726000000 μέχρι 62820000000 κύκλους μηχανής. Τέλος, εύκολα προκύπτει από τα παραπάνω ότι ο αρχικός ρυθμός αποκωδικοποίησης κυμαινόταν από10.34 μέχρι 10 fps. Στο διάγραμμα που ακολουθεί διακρίνεται το ποσοστό που δαπανά η εφαρμογή σε κάθε συνάρτηση.



Όπως εύκολα μπορεί να διακρίνει κάποιος το μεγαλύτερο μέρος καταναλώνεται στην διαδικασία του inter layer prediction,που η περιγραφή του μηχανισμού δόθηκε στο κεφάλαιο 2.

4.2 Βελτιστοποίηση των συναρτήσεων για την διαδικασία Inter-Layer-Prediction

Στην ενότητα αυτή θα γίνει εκτενής περιγραφή των μεθόδων που χρησιμοποιήθηκαν για την βελτιστοποίηση του μηχανισμού Inter-Layer-Prediction.

4.2.1 xBasicIntraUpSampling

Η συνάρτηση αυτή καλείται για να γίνει Inter-Layer Intra Prediction, όπως αυτή περιγράφτηκε στο κεφάλαιο 2. Όπως έχει ήδη αναφερθεί, η διαδικασία αυτή λαμβάνει χώρα για την δημιουργία του ανώτερου ιεραρχικά χωρικού επιπέδου από το base layer. Στην συνέχεια θα δοθεί εκτενής περιγραφή για τον τρόπο με τον οποίο υπολογίζονται τα pixels του Luma. Για τα pixels που αναπαριστούν το Luma χρησιμοποιείται ένα φίλτρο που εμπεριέχει 4 τιμές, οι τιμές

του οποίου έχουν άθροισμα 32. Η διαδικασία του upsampling δημιουργεί νέα pixels με ακρίβεια 1/16 pix.Στη πραγματικότητα η διαδικασία είναι αυτή διακριτή συνέλιξη. Η διακριτή συνέλιξη δίνεται από το τύπο$h(n) * g(n) = \sum_{-00}^{+00} h(k) * g(n-k)$. Στην συγκεκριμένη περίπτωση το h(k) περιέχει τις τιμές του φίλτρου και g(n-k) αντιστοιχεί στις τιμές pixels που χρησιμοποιούνται για την συγκεκριμένη μέθοδο. Η μέθοδος upsampling αρχικά υπολογίζει τις τιμές των pixel οριζόντια και στην συνέχεια κάθετες δημιουργώντας το νέο καρέ. Στην πραγματικότητα για κάθε pixel υπολογίζονται 2 νέα για το enhancement layer. Η μέθοδος αποτελείται από 2 επαναλήψεις, 2 βρόγχων. Η πρώτη επανάληψη αποτελείται από δυο βρόγχους επαναλήψεων και υπολογίζει τις τιμές για τα οριζόντια. Ο πρώτος βρόγχος έχει κάτω όριο το 0 και πάνω όριο το πλάτος του καρέ του base layer. Ο δεύτερος βρόγχος έχει κάτω όριο το 0 και πάνω όριο το μήκος του τρέχοντος επιπέδου. Ανάλογα με την θέση των pixels που χρησιμοποιούνται για το upsampling χρησιμοποιούνται και διαφορετικά φίλτρα, έτσι ώστε να ικανοποιείται ο περιορισμός για ακρίβεια ενός 1/16.Ο πίνακας που έπεται δείχνει τα φίλτρα που χρησιμοποιούνται.

0	32	0	0
-1	32	2	-1
-2	31	4	-1
-3	30	6	-1
-3	28	8	-1
-4	26	11	-1
-4	24	14	-2
-3	22	16	-3
-3	19	19	-3
-3	16	22	-3
-2	14	24	-4
-1	11	26	-4
-1	8	28	-3

-1	6	30	-3
-1	4	31	-2
-1	2	32	1

Εκτός από τις 2 περιπτώσεις στα όρια κάθε γραμμής που δημιουργείται ένα pixel μόνο αριστερά και αντίστοιχα ένα μόνο δεξιά, για όλες τις άλλες θέσεις δημιουργούνται 2 νέα pixels ανάμεσα σε 2 pixels και τα προηγούμενα pixels διαγράφονται. Στο επόμενο σχήμα με το κύκλο αναπαριστώνται τα pixels του προηγούμενου επιπέδου και με το σύμβολο x τα pixels που δημιουργούνται

Σχήμα 1

x ○ x x ○ x x ○ x x ○ x

Εκτός από τις οριακές περιπτώσεις που υπολογίζεται ένα νέο pixel, σε όλες τις άλλες περιπτώσεις χρησιμοποιούνται τα ίδια pixels για να υπολογιστούν τα νέα pixels στις άρτιες και περιττές θέσεις. Για παράδειγμα για το pixel στην θέση 1 θα χρησιμοποιηθούν τα pixels m,m+1,m+2,m+3,του base layer. Ομοίως και για το pixel στην θέση 2 . Η θέση των pixels που θα χρησιμοποιηθούν ορίζεται από μια μεταβλητή που ονομάζεται offset. Στην πραγματικότητα εισάγει την έννοια του αιτιατού συστήματος . Αιτιατό σύστημα είναι το σύστημα του οποίου η έξοδος δε εξαρτάται από μελλοντικές τιμές. Επιπλέον, επειδή ο αλγόριθμός βασίζεται στην ακρίβεια του 1/16, τα φίλτρα που χρησιμοποιούνται για τις άρτιες και περιττές θέσεις έχουν διαφορά φάσης 1/2. Στο συγκεκριμένο αλγόριθμο όπως αναφέραμε γίνεται πρώτα το οριζόντιο upsampling και ύστερα το κάθετο upsampling. Επιπροσθέτως, ελέγχεται αν τα pixels που χρησιμοποιούνται από το base layer βρίσκονται μέσα σε συγκεκριμένα όρια που ορίζονται από το offset, όπως αναφέραμε, και αν δε ισχύει γίνεται η κατάλληλη προσαρμογή. Παράλληλα, για όλες τις πράξεις χρησιμοποιούνται πράξεις με int τόσο για τα ενδιάμεσα αποτελέσματα όσο και για τα τελικά. Στο τέλος της διαδικασίας γίνεται μετατροπή σε short. Ενώ μέσα στους βρόγχους γίνεται ο έλεγχος αν η διαδικασία αφορά τα luma pixels. Η πρώτη αλλαγή που έγινε ήταν η χρησιμοποίηση short αριθμών, και μόνο για όποια ενδιάμεσα αποτελέσματα χρειαζόταν χρησιμοποιήθηκαν πράξεις ακεραίων. Στη συνέχεια δημιουργήθηκαν δύο επιπλέον βρόγχοι. Ο

ένας περιέχει διανυσματικές εντολές και εκτελείται μόνο στην περίπτωση που όλα τα pixels βρίσκονται μέσα στα όρια ,Αλλιώς εκτελείται ο αρχικό κώδικας. Ο έλεγχος για το αν η μέθοδος πρέπει να γίνει για τα luma pixels εκτελείται πλέον έξω από τους βρόγχους των επαναλήψεων. Τέλος, τροποποιήθηκε ο τρόπος με τον οποίο γίνεται η διαδικασία upsampling. Πλέον γίνεται πρώτα το vertical upsampling και ύστερα το horizontal. Οι λόγοι αυτής της αλλαγής θα γίνουν κατανοητοί στην συνέχεια. Για το vertical upsampling ,χρησιμοποιήθηκε η μέθοδος που είναι γνωστή ως "collapse", δηλαδή για κάθε γραμμή που περιέχει τα pixels του base layer, πολλαπλασιάζονται όλες οι στήλες με την ίδια σταθερά, που αντιστοιχεί σε μια τιμή του αντιστοίχου φίλτρου. Στην συνέχεια θα παρουσιαστεί πως δημιουργούνται οι καταχωρητές με τις κατάλληλες τιμές για κάθε φίλτρο. Αρχικά δημιουργούμε 2 πίνακες τύπου short, διαστάσεων 16x8, που ο ένας περιέχει τις 2 πρώτες τιμές του φίλτρου, που δημιουργεί τις κατάλληλες τιμές για τον πολλαπλασιασμό των 2 πρώτων γραμμών, και ο άλλος πίνακας τις επόμενες τιμές που δημιουργούν τις κατάλληλες σταθερές για τον πολλαπλασιασμό της 3ης και 4ης γραμμής. 'Εστω οι τιμές του φίλτρου a,b,c,d ,τότε προκύπτουν οι εξής πίνακες

Πίνακας A

a	a	a	a	a	a	a	a

Ο πίνακας A προκύπτει ως εξής. Φορτώνουμε σε ένα καταχωρητή xmm1 τις ακόλουθες 4 τιμές

0x0000FFFF, 0x0000FFFF00x0000FFFF,x0000FFFF. Επομένως προκύπτει ο xmm1 να περιέχει στην απεικόνιση 8x16 bits τις ακόλουθες τιμές.

xmm1:

0xFFFF	0	0xFFFF	0	0	0xFFFF	0	0xFFFF

Ο καταχωρητής xmm0 περιέχει τις ακόλουθες τιμές

xmm0:

a	b	a	b	a	b	a	b

Στην συνέχεια γίνεται χρήση της εντολής pand με ορίσματα τους καταχωρητές xmm0, xmm1. Το αποτέλεσμα που προκύπτει υφίσταται δεξιά ολίσθηση κατά 2 bits, με χρήση της εντολής psrldq και αποθηκεύεται σε νέο καταχωρητή. Τα αποτελέσματα των 2 αυτών πράξεων οδηγούνται σε ένα λογικό or με χρήση της εντολής por. Τελικά μέσα από αυτή την σειρά πράξεων προκύπτει ένας καταχωρητής που το περιεχόμενο των 16x8 bits περιέχει το πίνακα A. Στην συνέχεια πρέπει να δημιουργήσουμε ένα καταχωρητή με περιεχόμενο την τιμή b επαναλαμβανόμενη 8 φορές. Η διαδικασία μοιάζει με την προηγούμενη με την διαφορά ότι στο καταχωρητή xmm1 φορτώνονται οι τιμές 0xFFFF0000, 0xFFFF0000, 0xFFFF0000, 0xFFFF0000. Τελός μετά την εκτέλεση της εντολής pand ακολουθεί η εντολή pslldq. Μετά την εκτέλεση των εντολών αυτών προκύπτει ο καταχωρητής xmm1 με τιμές ,αυτές του παρακάτω πίνακα

b	b	b	b	b	b	b	b

Επαναλβάνοντας την ίδια διαδικασία και για το δεύτερο πίνακα με τα φίλτρα, προκύπτουν και οι καταχωρητές xmm2 και xmm3 με τιμές ίδιες με αυτές των πινάκων που ακολουθούν.

c	c	c	c	c	c	c	c

d	d	d	d	d	d	d	d

Χρησιμοποιώντας την εντολή pmullw, πολλαπλασιάζουμε κάθε γραμμή με τα κατάλληλα φίλτρα. Στο τέλος προσθέτουμε τα τελικά αποτελέσματα ανά 2 χρησιμοποιώντας την εντολή paddw. Στο τέλος τα αποτελέσματα αυτά αποθηκεύονται σε ένα προσωρινό καταχωρητή τύπου short για να χρησιμοποιηθούν, από το οριζόντιο upsampling. Στο οριζόντιο upsampling η είσοδος είναι η έξοδος της προηγούμενης διαδικασίας και τα ενδιάμεσα αποτελέσματα είναι τύπου integer, ενώ τα τελικά είναι τύπου short. Επειδή τα ενδιάμεσα αποτελέσματα είναι τύπου int θα πρέπει να χρησιμοποιήσουμε εντολές που δέχονται ως είσοδο αριθμούς τύπου short και παράγουν integer. Μια τέτοια υλοποίηση είναι εύκολο να γίνει για το οριζόντιο αλλά δύσκολο για το κάθετο .Αυτός είναι ο λόγος που έγινε πρώτα το κάθετο upsampling. Στην συγκεκριμένη

περίπτωση χρησιμοποιούμε την εντολή pmaddwd. Η εντολή αυτή πολλαπλασιάζει τα περιεχόμενα 2 καταχωρητών ,και παράγει ενδιάμεσα αποτελέσματα τα οποία προσθέτει ανά 2 και προκύπτουν 4 x32 αριθμοί ως έξοδος.

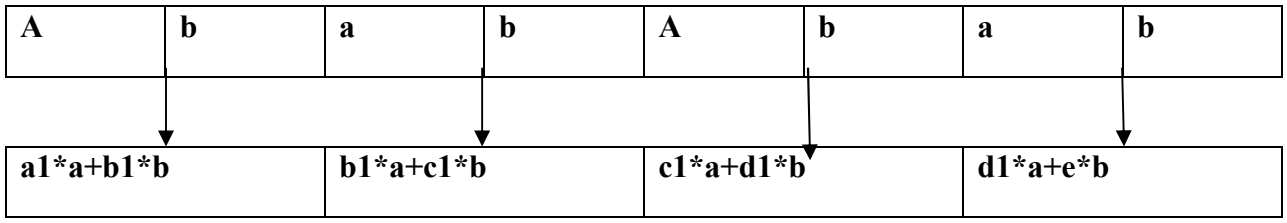
Έστω **a1 b1 c1 d1 e f g h i j k l m n o p** τα pixels που θα χρησιμοποιηθούν ως pixels αναφοράς. Ομοίως με πριν χρησιμοποιούμε τους 2 πίνακες που περιέχουν τις τιμές των φίλτρων. Τις τιμές των φίλτρων τις φορτώνουμε σε δυο καταχωρητές xmm0, xmm1. Επομένως στο ένα καταχωρητή έχουμε τις τιμές του πρώτου πίνακα **a b a b a b a b a b** και στο δεύτερο καταχωρητή τις τιμές **c d c d c d c d.** Στην συνέχεια θα πρέπει να φορτώσουμε τα κατάλληλα pixels στους καταχωρητές για να εκτελέσουμε την εντολή pmaddwd. Για την εκτέλεση της εντολής αυτής με δεύτερο όρισμα τον καταχωρητή xmm0, θα πρέπει να δημιουργήσουμε ένα καταχωρητή ο οποίος να έχει περιεχόμενο το παρακάτω πίνακα

a1	**b1**	**b1**	**c1**	**c1**	**d1**	**d1**	**e**

Επομένως δημιουργούμε ένα νέο καταχωρητή που περιέχει αναμεμιγμένες τις τιμές των πρώτων pixels, και αποτελεί την πρώτη είσοδο της pmaddwd.

Η παρακάτω εικόνα απεικονίζει την είσοδο και την έξοδο της παραπάνω εντολής

a1	**b1**	**b1**	**c1**	**c1**	**d1**	**d1**	**e**



Στην συνέχεια θα περιγραφτεί πως προκύπτει η πρώτη είσοδος. Αρχικά φορτώνουμε τα πρώτα 8 pixels σε ένα καταχωρητή έστω xmm2. Στην συνέχεια χρησιμοποιούμε την εντολή **PUNPCKLWD.** Η χρήση της εντολής αυτής έχει το παρακάτω αποτέλεσμα.:

a1	**0**	**b1**	**0**	**c1**	**0**	**d1**	**0**

Σε ένα νέο καταχωρητή αποθηκεύουμε τα 8 pixels του καταχωρητή ολισθημένες κατά 2 bits. Επομένως τα περιεχόμενα του νέου καταχωρητή είναι:

b1	c1	d1	e	f	g	h	0

Το αποτέλεσμα της εντολής αυτής χρησιμοποιείται ως είσοδο της εντολής **PUNPCKLWD .**

Επομένως τα νέα περιεχόμενα του καταχωρητή είναι :

b1	0	c1	0	d1	0	e	0

Ακολουθεί ολίσθηση των περιεχομένων του καταχωρητή κατά 2 θέσεις αριστερά ,με χρήση της εντολής pslldq. Τέλος τα περιεχόμενα των δυο καταχωρητών συγχωνεύονται σε έναν νέο με την χρήση της εντολής por. Για τα pixels που χρησιμοποιούν το πίνακα με τις τιμές για τα φίλτρα ,**c d c d c d c d** η διαδικασία που ακολουθούμε είναι η εξής. Έχουμε ήδη δημιουργήσει ένα καταχωρητή με περιεχόμενα

a1	**b1**	**b1**	**c1**	**c1**	**d1**	**d1**	**e**

Χρησιμοποιώντας την εντολή για δεξιά λογική ολίσθηση (`psrldq`) ολισθαίνουμε τα περιεχόμενα κατά 8 θέσεις .Άρα προκύπτει ένας νέος πίνακας

c1	d1	d1	e	0	0	0	0

Σε ένα νέο καταχωρητή αποθηκεύουμε τα 8 pixels του καταχωρητή ολισθημένες κατά 2 bits. Επομένως τα περιεχόμενα του νέου καταχωρητή

Το αποτέλεσμα της εντολής αυτής χρησιμοποιείται ως είσοδο της εντολής **PUNPCKLWD .**

Επομένως τα νέα περιεχόμενα του καταχωρητή είναι :

Ακολουθεί ολίσθηση των περιεχομένων του καταχωρητή κατά 2 θέσεις αριστερά, με χρήση της εντολής pslldq. Τέλος τα περιεχόμενα των δυο καταχωρητών συγχωνεύονται σε έναν νέο με την χρήση της εντολής por. Για τα pixels που χρησιμοποιούν το πίνακα με τις τιμές για τα φίλτρα, **c d c d c d c d** η διαδικασία που ακολουθούμε είναι η εξής .Έχουμε ήδη δημιουργήσει ένα καταχωρητή με περιεχόμενα

Χρησιμοποιώντας την εντολή για δεξιά λογική ολίσθηση (`psrldq`) ολισθαίνουμε τα περιεχόμενα κατά 8 θέσεις. Άρα προκύπτει ένας νέος πίνακας

Η ίδια διαδικασία που ακολουθήθηκε για τα χαμηλότερα pixels, δηλαδή αυτά στις θέσεις 0 -4, που πολλαπλασιάζονται με το φίλτρο 1, μπορεί να ακολουθηθεί και για τα υψηλότερης θέσης pixels που χρησιμοποιούν το ίδιο φίλτρο. Έτσι για το φίλτρο 1 προκύπτει ότι για τα pixels στις θέσεις 5-9 πρέπει ο καταχωρητής να έχει τις τιμές του επόμενου πίνακα

e	f	f	g	g	h	h	i

Ολισθαίνοντας το περιεχόμενο κατά 4 θέσεις αριστερά προκύπτουν οι 4 υψηλότερης σημαντικότητας τιμές που απαιτούνται για το φίλτρο 2. Ήδη έχουν δημιουργηθεί και οι 4 χαμηλότερης σημαντικότητας τιμές του φίλτρου 2 και με χρήση της εντολής por προκύπτει ο καταχωρητής που χρειάζεται για το φίλτρο 2.

Τα περιεχόμενα του νέου καταχωρητή είναι

c1	d1	d1	e	e	f	f	g

Για το φίλτρο 2 υπολείπεται να υπολογιστούν τα υψηλότερα pixels. Η διαδικασία που ακολουθείται είναι παρόμια με αυτή που περιγράφτηκε ήδη, και προκύπτει τα περιεχόμενα ενός νέου καταχωρητή

e	f	f	g	g	h	h	i

Στην συνέχεια τα αποτελέσματα των 2 pmaddwd, χρησιμοποιούνται ως είσοδο στην εντολή paddw. Όπως αναφέρθηκε ήδη στην αρχή η παραπάνω διαδικασία, χρησιμοποιείται για τον υπολογισμό είτε των περιττών είτε των άρτιων pixel στο enhancement layer. Για την συγχώνευση των άρτιων και περιττών σε ένα καταχωρητή ,χρησιμοποιήθηκε η εντολή **PUNPCKLWD.** Επειδή όμως μετά την εντολή pmaddwd, ενδέχεται τα αποτελέσματα να μη είναι στο εύρος που ορίζει το πρότυπο C για τους signed short, πρέπει να γίνουν μια σειρά από άλλες πράξεις. Αρχικά ολίσθηση του τελικού αποτελέσματος κατά 10 bits αφού πρώτα γίνει στρογγυλοποίηση, προσθέτοντας την τιμή 512. Οι παραπάνω διανυσματικές πράξεις εκτελούνται χρησιμοποιώντας τις εξής εντολές : **paddd, psrad**. Στην επόμενη φάση θα πρέπει

το εύρος των αριθμών κάθε αποτελέσματος να περιορίζεται από 0 -255. Για να επιτευχθεί αυτό αρχικά χρησιμοποιείται η εντολή **PACKSSDW,** για να γίνει η συγχώνευση των αποτελεσμάτων σε 8x16 bits. Στην συνέχεια χρησιμοποιείται η εντολή **PACKUSWB**, για να περιοριστεί το εύρος των αριθμών από 0 -255. Ύστερα από την ολοκλήρωση αυτών των πράξεων γίνεται η συγχώνευση των καταχωρητών ,και γράφονται πίσω στην μνήμη.

Στην παρούσα παράγραφο θα αναλυθεί η διαδικασία που ακολουθήθηκε για την παραλληλοποίηση της διαδικασίας upsampling των pixels που αντιστοιχεί στο Chroma. Αρχικά έγιναν ορισμένες αρχιτεκτονικές αλλαγές, οι οποίες είναι ίδιες με αυτές για τα pixel που αντιπροσωπεύουν το Luma. Επιπρόσθετα οι πράξεις γίνονται με μη προσημασμένους αριθμούς αριθμούς, μήκους 16 –bits. Για την διαδικασία που περιγράφεται σε αυτήν την παράγραφο χρησιμοποιείται η μέθοδος της διγραμμικής παρεμβολής, όπου και εδώ εισάγεται η έννοια του 1/16. Επιπλέον τα φίλτρα που χρησιμοποιούνται είναι πολλαπλάσια του 2, επομένως μπορούμε να διαιρέσουμε όλα τα φίλτρα με το 2 χωρίς φυσικά να αλλοιωθεί το αποτέλεσμα. Ο επόμενος πίνακας παρουσιάζει όλα τα φίλτρα

16	0
15	1
14	2
13	3
12	4
11	5
10	6
9	7
8	8
7	9
6	10
5	11
4	12
3	13
2	14
1	15

Όπως και στο upsampling για τα pixels της φωτεινότητας και εδώ γίνεται πρώτα το vertical upsampling. Η μέθοδος που ακολουθείται είναι η ίδια. Δηλαδή πολλαπλασιασμός μιας ολόκληρης γραμμής με μια συγκεκριμένη τιμή. Σε αυτή την φάση δε ήταν απαραίτητο η αλλαγή

της σειράς εκτέλεσης του upsampling αφού τα δεδομένα δε ξεπερνούν τα 16 bits, παρόλα αυτά χρησιμοποιήθηκε και αυτή η αρχιτεκτονική αλλαγή. Επιπλέον αν υποθέσουμε ότι χρησιμοποιείται το φίλτρο με τιμές x,y, η πρώτη γραμμή θα πρέπει να πολλαπλασιαστεί με την τιμή x και η δεύτερη γραμμή με την τιμή y. Για την δημιουργία των τιμών αυτών, χρησιμοποιούνται οι ίδιες μάσκες που χρησιμοποιήθηκαν και στην αντίστοιχη διαδικασία για το Luma,καθώς και οι ίδιες πράξεις. Επιπλέον και οι πράξεις είναι ίδιες, με την μόνη διαφορά ότι η τελική πρόσθεση γίνεται με την εντολή **paddusb.** Στην συνέχεια θα περιγραφεί η διαδικασία του horizontal upsampling. Όπως και για τα pixel της φωτεινότητας χρησιμοποιείται η εντολή **pmaddwd.** Η διαδικασία είναι πιο απλή από την αντίστοιχη για την φωτεινότητα .Αν υποθέσουμε ότι το φίλτρο έχει τις τιμές x,y και τα αντίστοιχα pixels τις τιμές **a b c d e f g h i j k l m n o p .** Θα πρέπει να δημιουργηθεί ένας καταχωρητής με τιμές που φαίνονται στον ακόλουθο πίνακα

a	b	b	c	c	d	d	e

Η διαδικασία που ακολουθείται έχει ως εξής: αρχικά φορτώνουμε τα 8 πρώτα pixels σε ένα καταχωρητή .Στην συνέχεια εκτελείται η διανυσματική εντολή **PUNPCKLWD.** Έτσι προκύπτει ένας νέος καταχωρητής με περιεχόμενα :

a	0	b	0	c	0	d	0

Το καταχωρητή που εμπεριέχει τα αρχικά pixels τον ολισθαίνουμε κατά 2 θέσεις δεξιά χρησιμοποιώντας την εντολή **psrldq.** Μετά την ολίσθηση εκτελείται η διανυσματική εντολή **PUNPCKLWD,** και στην συνέχεια υφίσταται αριστερή ολίσθηση κατά 2 bits με χρήση της εντολής **pslldq.** Τέλος οι δυο νέοι καταχωρητές που προκύπτουν οδηγούν την εντολή por και προκύπτει ο τελικός καταχωρητής που θα χρησιμοποιηθεί για το horizontal upsampling. Παρόμοια διαδικασία ακολουθείται για pixels στις θέσεις 5 -9. Γι' αυτές τις θέσεις οι τιμές του καταχωρητή είναι :

e	f	f	g	g	h	h	j

Για οριζόντιο upsampling πριν την εκτέλεση των τελικών πράξεων μπορεί να γίνει συγχώνευση των περιττών και άρτιων pixels του enhancement layer, αφού το μήκος των δεδομένων κυμαίνεται έπο 0 -16 bits. Στην συνέχεια εκτελούνται οι τελικές διανυσματικές πράξεις για τον

υπολογισμό των τελικών τιμών. Αρχικά προστίθεται η τιμή 128 (εντολή **paddusw**), για την στρογγυλοποίηση και στην συνέχεια ολίσθηση κατά 8 bits(εντολή **psraw**). Τέλος, εκτελείται η διαδικασία περιορισμού του εύρους των αριθμών από 0 έως 255. Ο περιορισμός γίνεται με χρήση της εντολής **PACKUSWB.**

Έπειτα από όλες τις παραπάνω αλλαγές οι συνολικοί κύκλοι μηχανής που καταναλώνονται σε αυτή την συνάρτηση είναι **403200000.** Η μη βελτιστοποιημένη συνάρτηση για να ολοκληρώσει με επιτυχία την εκτέλεσης της απαιτούσε **9561300000.** Η συνολική επιτάχυνση υπολογίζεται στα 23.71x. Δηλαδή βελτιστοποιήθηκε σε ποσοστό 95.79% .

4.2.2 xBasicResidualUpsampling

Η συνάρτηση αυτή χρησιμοποιείται για την μέθοδο Inter Layer Residual Prediction,η οποία περιγράφτηκε στο κεφάλαιο 2. Η συνάρτηση αυτή για να κάνει upsampling τα υπόλοιπα χρησιμοποιεί διγραμμική παρεμβολή .Ο αλγόριθμος που ακολουθείτε δε διαφέρει με την αυτόν που περιγράφτηκε στην προηγούμενη ενότητα, για την μέθοδο xBasicIntraUpSampling. Όμως υπάρχει μια διαφορά με τον προηγούμενο αλγόριθμο. Οι τιμές των φίλτρων που χρησιμοποιούνται κατά την διαδικασία του upsampling αλλάζουν τιμή στα όρια ενός block 4x4. Για παράδειγμα, έστω ότι εκτελείται το πρώτο κομμάτι του αλγορίθμου για το οριζόντιο upsampling. Για το τρέχων επίπεδο, πρέπει να βρούμε τα pixels που αντιστοιχούν στο επίπεδο αναφοράς για την γραμμή i που εκτελείται ο αλγόριθμος. Αν τα pixels αναφοράς που πρέπει να χρησιμοποιηθούν για την διγραμμική παρεμβολή έχουν την τιμή 3 και την τιμή 4, τότε προφανώς ανήκουν σε διαφορετικό block. Σε αυτήν την περίπτωση αντιγράφεται είτε το αριστερό είτε το δεξιά pixel αναφοράς, πολλαπλασιασμένο με 16. Το pixel που θα αντιγραφεί εξαρτάται από την θέση του pixel του τρέχοντος που πρέπει να υπολογιστεί. Αν για παράδειγμα πρέπει να υπολογιστεί στην θέση i που είναι περιττή , η απόσταση του είναι μικρότερη από το αριστερά ,άρα αντιγράφεται το αριστερά. Αν πρέπει να υπολογιστεί στην θέση j που είναι άρτια, η απόσταση του είναι πιο κοντά στο αριστερό pixel, άρα αντιγράφεται αυτό. Στην συνέχεια θα περιγραφούν οι αλλαγές που έγιναν για να γίνουν διανυσματικές πράξεις. Οι αρχιτεκτονικές αλλαγές οι οποίες έγιναν, είναι ίδιες με αυτές της προηγούμενης μεθόδου για το Chroma. Όμως σε αυτήν την περίπτωση, η εκτέλεση πρώτα του κάθετο upsampling, είναι απαραίτητη γιατί ως είσοδο έχουμε προσημασμένους αριθμούς, με μήκος 16 bits. Το εύρος των αριθμών αυτών

κυμαίνεται από [-255 έως 255]. Τα φίλτρα που χρησιμοποιούνται είναι ίδια με αυτά του Chroma στην προηγούμενη μέθοδο, για τις μη οριακές περιοχές. Για τις οριακές περιοχές, δηλαδή εκεί που αλλάζει block χρησιμοποιείται το φίλτρο [16 0] ή το φίλτρο [0 16], εξαρτάται ποιά τιμή του τρέχοντος layer πρέπει να υπολογιστεί. Για το vertical upsampling χρησιμοποιούμε ένα **if –else** για να ελέγξουμε αν τα υπόλοιπα των γραμμών που θέλουμε να υπολογίσουμε ανήκουν σε διαφορετικά blocks .Έστω ότι ένα φίλτρο με τιμές [a,b]. Θα πρέπει να δημιουργήσουμε ένα καταχωρητή με περιεχόμενα **a a a a a a a a** για την γραμμή i και ένα καταχωρητή με περιεχόμενα **b b b b b b b b** για την γραμμή i+1. Ο τρόπος που δημιουργούνται είναι ίδιος με αυτόν για το χρώμα. Στην συνέχεια εκτελούνται οι ίδιες πράξεις που εκτελούνται για το chroma, με τη διαφορά ότι η τελική πρόσθεση γίνεται με χρήση της εντολής **paddw.** Η διαφοροποίηση αυτή γίνεται διότι πλέον χρησιμοποιούμε προσημασμένους αριθμούς, μήκους 16 bit. Για το horizontal upsampling η διαδικασία που ακολουθείται για να φορτώσουμε στου καταχωρητές τις κατάλληλες τιμές είναι ίδια με την αντίστοιχη του chroma που περιγράφτηκε για την υλοποίηση της μεθόδου της προηγούμενης ενότητας. Όμως ορίζεται νέος πίνακας με τις τιμές των φίλτρων που πρέπει να φορτωθούν σε ένα καταχωρητή για να γίνει η διγραμμική παρεμβολή με χρήση διανυσματικών πράξεων. Ο πίνακας είναι ο παρακάτω :

16	0	16	0	16	0	1	0
15	1	15	1	15	1	1	0
14	2	14	2	14	2	1	0
13	3	13	3	13	3	1	0
12	4	12	4	12	4	1	0
11	5	11	5	11	5	1	0
10	6	10	6	10	6	1	0
9	7	9	7	9	7	1	0
8	8	8	8	8	8	0	1
7	9	7	9	7	9	0	1

6	10	6	10	6	10	0	1
5	11	5	11	5	11	0	1
4	12	4	12	4	12	0	1
3	13	3	13	3	13	0	1
2	14	2	14	2	14	0	1
1	15	1	15	1	15	0	1

Σε αυτήν παράγραφο θα περιγράψουμε τις πράξεις που γίνονται για να τον υπολογισμό των τελικών τιμών των υπολοίπων. Ο καταχωρητής που περιέχει τις κατάλληλες τιμές που απαιτούνται για το upsampling και ο καταχωρητής με τις τιμές του φίλτρου, αποτελούν την είσοδο της **pmaddwd.** Ο καταχωρητής που προκύπτει ως έξοδο ,προστίθεται με ένα καταχωρητή, για να γίνει η στρογγυλοποίηση. Επειδή το φίλτρο δεν έχει παντού τις ίδιες τιμές ,όπως αναφέραμε ο καταχωρητής αυτός έχει περιεχόμενο 4x32 **[128 128 128 8]**. Στην συνέχεια ολισθαίνουμε κατά αριστερά το αποτέλεσμα του καταχωρητή κατά 4 bits, με στόχο να "πετάξουμε" την τελευταία τιμή. Το περιεχόμενο του νέου καταχωρητή το διαιρούμε με 128 (εντολή **psrad**). Η τελευταία τιμή αποτελεί την τιμή στα όρια . Η τιμή αυτή πρέπει να διαιρεθεί με 16 .Επομένως ολισθαίνουμε το περιεχόμενο του αρχικού καταχωρητή κατά 12 θέσεις έτσι ώστε η τιμή στην θέση [4] της 4x32 απεικόνιση να έρθει στην θέση 1.Διαιρούμε την τιμή αυτή με 16 (εντολή **psrad**). Στην συνέχεια την τιμή που προέκυψε την επαναφέρουμε στην αρχική της θέση και συγχωνεύουμε τα αποτελέσματα των 2 καταχωρητών (ο ένας περιέχει τις τιμές ολισθημένες κατά 8 και ο άλλος ολισθημένες κατά 4), με χρήση της εντολής **por.**

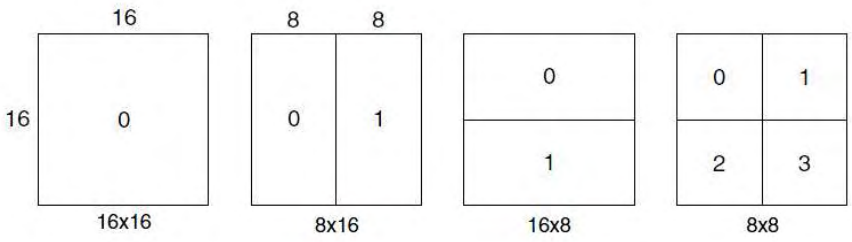
Μετά την βελτιστοποίηση της παραπάνω συνάρτησης με χρήση διανυσματικών εντολών, οι συνολικοί κύκλοι που καταναλώνονται μέσα σε αυτή είναι **338100000.** Στον αρχικό κώδικα η μη βελτιστοποιημένη συνάρτηση απαιτούσε **5594400000.** Η συνολική επιτάχυνση της συγκεκριμένης συνάρτησης είναι 16.54x. Δηλαδή η συγκεκριμένη συνάρτηση βελτιστοποιήθηκε κατά 94 %.

4.2.3 Συνολική βελτιστοποίηση

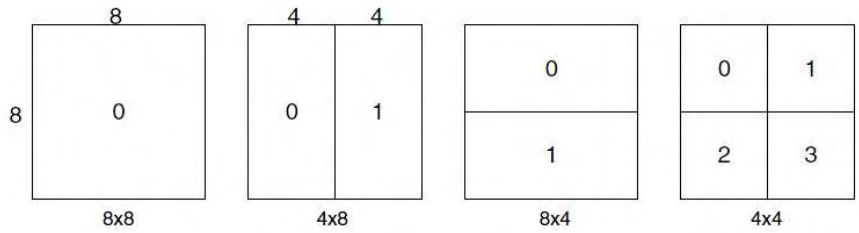
Το τμήμα του κώδικα που εκτελεί το μηχανισμό του Inter-Layer Prediction, όπως αναφέραμε αποτελείται από τις συναρτήσεις xBasicIntraUpSampling, xBasicResidualUpsampling. Παράλληλα, στο μηχανισμό αυτό χρησιμοποιείται και η συνάρτηση xClip, που περιορίζει το εύρος ενός αριθμού σε συγκεκριμένα όρια. Η συνάρτηση αυτή χρησιμοποιείται μόνο για το υπολογισμό των υπολοίπων καθως και των pixels στην αρχή και τέλος καθε γραμμής και στήλης. Επιπλέον στην αρχική υλοποιήση χρησιμοποιούταν για να περιορίσει το εύρος των pixels απο 0 -255. Η υλοποίηση αυτή πλέον γίνεται με εντολές SIMD. Ο compiler την συνάρτηση αυτή την έχει κάνει inline. Συνοψίζοντας συνολικά όλος ο μηχανισμός για να ολοκληρώσει με επιτυχία την εκτέλεσή του δαπανούσε **19481700000** κύκλου μηχανής . Με την χρήση διανυσματικών εντολών οι συνολικοί κύκλοι μηχανής μειώνονται στους **741300000.** Η συνολική επιτάχυνση είναι **26.2x.** Το οποίο αντιστοιχεί σε ποσοστό **96.2 %.**

4.3 Motion Compensation για την φωτεινότητα

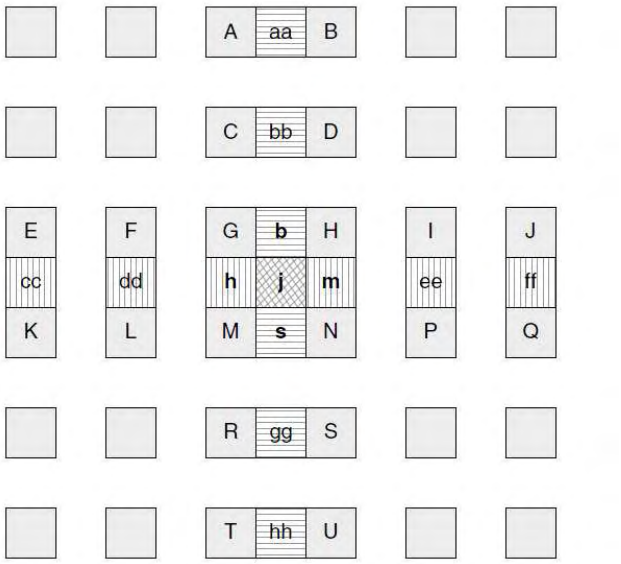
Στη ενότητα αυτή θα περιγραφεί η διαδικασία βελτιστοποίησης του motion compensation. Motion compensation είναι η διαδικασία, κατά την οποία μια εικόνα (ένα καρέ βίντεο) αποκωδικοποιείται χρησιμοποιώντας ως αναφορά προηγούμενα ή καρέ που έπονται. Δηλαδή, το καρέ που αποκωδικοποιείται με την τεχνική του motion compensation δε περιέχει κωδικοποιημένες πληροφορίες που αφορούν την ακριβή τιμή των pixels, αλλά περιέχει πληροφορία που περιέχει το index για το καρέ αναφοράς που πρέπει να χρησιμοποιηθεί για να κατασκευαστεί σωστά το τρέχων καρέ, καθώς και τα υπόλοιπα που προέκυψαν κατά την διαδικασία του motion estimation,τα οποία προστίθενται στο τέλος της διαδικασίας. Η πληροφορία που αναφέρει ποίο frame πρέπει να χρησιμοποιηθεί σαν frame αναφοράς, χαρακτηρίζεται σαν motion vector .Στη γενική περίπτωση ένα motion vector χρησιμοποιείται για όλο το macroblock, όμως σε ένα macroblock μπορούν να υπάρχουν περισσότερα από ένα motion vectors. Το macroblock χωρίζεται σε μικρότερα blocks κάθε ένα από τα οποία έχει ένα motion vector. Η παρακάτω εικόνα δείχνει τους τρόπους υποδιαίρεσης ενός macroblock.



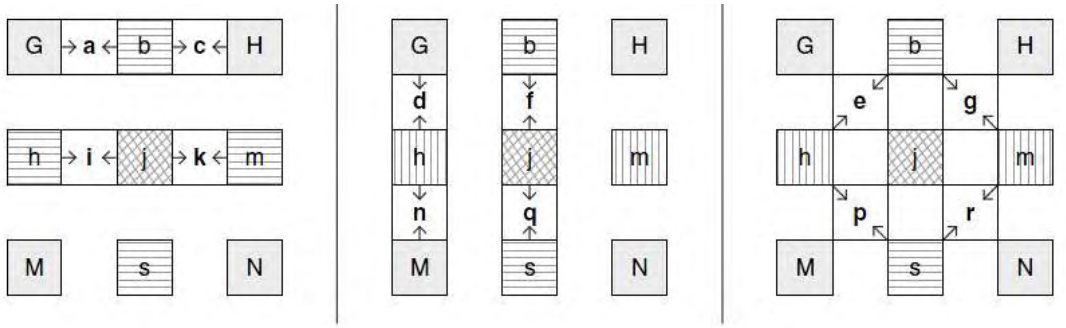
Επιπλέον κάθε ένα από τα παραπάνω blocks μπορεί να υποδιαιρεθεί σε sub blocks κάθε ένα από τα οποία περιέχει ένα motion vector. Οι διαφορετικοί τρόποι υποδιαίρεσης ενός block φαίνονται στην επόμενη εικόνα



Η διαφορά ανάμεσα σε δύο καρέ, δηλαδή το motion vector, έχει ακρίβεια 1/4 για τα pixels που αντιπροσωπεύουν την φωτεινότητα και έχει ακρίβεια 1/8 για τα pixels που αντιπροσωπεύουν το χρώμα. Τα pixels για το chroma και το luma που βρίσκονται στις μη ακέραιες θέσεις, δε υπάρχουν στα καρέ αναφοράς, για το λόγο αυτό θα πρέπει να υπολογιστούν. Στην συνέχεια παρατίθεται η εικόνα για την παρεμβολή ακρίβειας ½, και θα εξηγηθεί στην συνέχεια.



Τα pixel που αντιστοιχούν στις θέσεις ½ και βρίσκονται ανάμεσα σε ακέραια pixels, υπολογίζονται πρώτα. Κάθε pixel που αντιστοιχεί στην θέση ½ και βρίσκεται ανάμεσα σε ακέραια pixels υπολογίζεται χρησιμοποιώντας ένα φίλτρο 6 τιμών, με τιμές τις ακόλουθες [1/32,-5/32,20/32,20/32,-5/32,1/32]. Για παράδειγμα για το pixel b που βρίσκεται στην θέση ½ οριζόντια και ανάμεσα από τα ακέραια pixels G και H ,υπολογίζεται ως εξής b=round(E-5F+20G+20H-5I+J)/32. Για το pixel h που βρίσκεται στην θέση ½ κάθετα ακολουθείται η ίδια διαδικασία. Όταν υπολογιστούν όλα τα ½ pixels που βρίσκονται ανάμεσα σε ακέραια pixels ,πρέπει να υπολογιστούν και τα υπολειπόμενα ½ pixels ,αυτά δηλαδή που βρίσκονται ανάμεσα σε ½ pixels .Για παράδειγμα το pixel j παράγεται χρησιμοποιώντας τα pixels [cc dd h m ee ff]. Τέλος αφού υπολογιστούν όλα τα ½ pixels, στην συνέχεια υπολογίζονται τα pixels στη ¼ θέση. Τα ¼ pixels τοποθετούνται ανάμεσα σε ακέραια pixels ή ανάμεσα σε ½ pixels . Ο υπολογισμός των pixels αυτών υπολογίζεται με χρήση γραμμικής παρεμβολής. Οι παρακάτω εικόνες παρουσιάζουν τον υπολογισμό του ¼ pixels, χρησιμοποιώντας οριζόντιας γραμμικής παρεμβολής ,κάθετης και διαγώνιας .



Μέχρι τώρα έγινε αναφορά για τον τρόπο που υπολογίζονται τα pixels για το luma. Για τα pixels του chroma η διαδικασία είναι πολύ πιο απλή. Η μέθοδος που χρησιμοποιείται είναι διγραμμική παρεμβολή. Οι τιμές των φίλτρων εξαρτώνται από την θέση του pixel και είναι υπόλοιπα της θέσης του pixel αναφοράς με το 8.

Μια σημαντική σημείωση για το luma είναι ότι υπάρχει εναλλακτικός τρόπος υπολογισμού των ½ pixels, που χρησιμοποιείται και από τον αρχικό κώδικα. Για παράδειγμα για να υπολογίσουμε το pixel b, ο απλός τρόπος είναι να πολλαπλασιάσουμε τα pixels που χρειάζονται με τις κατάλληλες τιμές του φίλτρου. Όμως ένας πιο αποτελεσματικός τρόπος, όσο αναφορά την απόδοση, είναι η ανάλυση της παραπάνω σχέσης και να προκύψει η ακόλουθη σχέση (ως αναφορά έχουμε το pixel b),b=4(G+H-F-I)+16(G+H)-F-I+E+J. Σε θέματα απόδοσης είναι πιο γρήγορη η παραπάνω υλοποίηση αφού πλέον οι πολλαπλασιασμοί μπορούν να αντικατασταθούν με ολισθήσεις.

4.3.1 Βελτιστοποίηση συνάρτησης για ½ οριζόντιο interpolation με χρήση ακέραιων pixel (xPredDy0Dx2)

Η συνάρτηση αυτή δέχεται ως είσοδο μια περιοχή, όπως αυτή ορίζεται από τον τρόπο που διαιρείται το macroblock σε μικρότερα blocks. Όπως ήδη αναφέρθηκε και στην αρχή της ενότητας 4.3, ανάλογα με τον τύπο του macroblock η οριζόντια παρεμβολή θα εφαρμοστεί σε blocks μήκους 16, 8, 4. Για την χρήση διανυσματικών εντολών θα πρέπει αρχικά θα πρέπει να αντιγραφούν τα απαραίτητα pixels σε μια εσωτερική μνήμη, η οποία πρέπει να είναι ευθυγραμμισμένη (Κεφάλαιο 3 Διανυσματικές εντολές). Τα pixel που χρειάζονται για το ½ interpolation αρχικά βρίσκονται σε ακέραιες θέσεις στο καρέ αναφοράς και είναι ίση με το γινόμενο uiSizeY*(uiSizeX+5), όπου uiSizeY το πλάτος του block αναφοράς στο αντίστοιχο καρέ και uiSizeX το μήκος του αντίστοιχου block αναφοράς. Γίνεται εύκολα αντιληπτό ότι για το οριζόντιο interpolation χρειαζόμαστε 5 επιπλέον pixels. Δε χρειάζεται να γίνει καμία επιπλέον

ενεργεία για τα 5 επιπλέον pixel, αν αυτά βρίσκονται εκτός από το εύρος του καρέ, διότι ήδη έχει γίνει επέκταση σε επίπεδο frame αλλά και macroblock. Στην εισαγωγή της ενότητας 4.3 έχει περιγραφεί πως γίνεται το ½ interpolation για να υπολογιστούν τα pixels στις θέσεις ½, που βρίσκονται ανάμεσα σε ακέραια pixels. Στην συνέχεια φορτώνουμε σε ένα καταχωρητή τα κατάλληλα pixels. Επειδή η είσοδος έχει μήκος 16 bit, για να φορτώσουμε τα δεδομένα διακρίνουμε 3 περιπτώσεις .

- uiSizex =16 .Πρέπει να φορτώσουμε 21 bits. Άρα πρέπει να χρησιμοποιήσουμε τρείς εντολές movdqa για να φορτώσουμε τα κατάλληλα δεδομένα .
- uiSize=8. Πρέπει να φορτώσουμε 13 bits .΄Άρα πρέπει να χρησιμοποιήσουμε δύο εντολές movdqa για να φορτώσουμε τα κατάλληλα δεδομένα.
- uiSize=4 Πρέπει να φορτώσουμε 9 bits .Άρα πρέπει να χρησιμοποιήσουμε 1 εντολή movdqa και μια εντολή movq.

Στην συνέχεια πρέπει οι καταχωρητές που θα χρησιμοποιηθούν για το interpolation να φορτώσουν τα κατάλληλα δεδομένα .

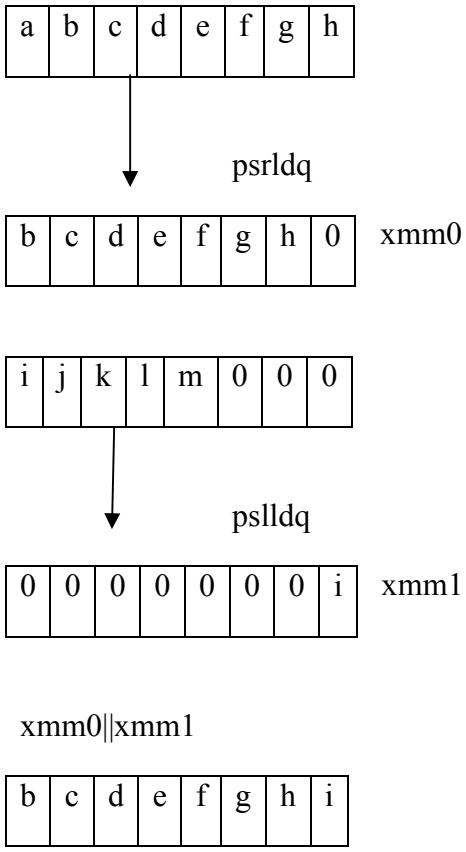
Αν για παράδειγμα έχουμε να φορτώσουμε τα pixels [**a b c d e f g h i j k l m**], που αντιστοιχούν στις θέσεις -2 έως στην θέση 10. Για το υπολογισμό του pixel στην θέση 0 θα πρέπει να χρησιμοποιήσουμε τα pixel [**a,b,c,d,e,f**]. Για τον υπολογισμό του pixel στην θέση 1 πρέπει να χρησιμοποιήσουμε τα pixel [**b,c,d,e,f,g**]. Ομοίως για τον υπολογισμό του pixel στην θέση 3 θα πρέπει να χρησιμοποιηθούν τα pixel **[c,d,e,f,g,h] .** Είναι προφανές ότι για θέση i+1 χρειαζόμαστε τα pixel της θέσης i, ολισθημένα κατά μια θέση αριστερά και προσθέτοντας στο τέλος ένα μόνο νέο pixel. Στη παρούσα φάση θα παρουσιαστεί πως θα δημιουργηθούν τα κατάλληλα περιεχόμενα, δηλαδή οι καταχωρητές που περιέχουν για την θέση i τα pixel -2, -1, 0, 1, 2, 3.

Υποθέτουμε ότι uiSize=8 .Η διαδικασία είναι παρόμοια για όλες τις περιπτώσεις.

Για τον καταχωρητή που περιέχει όλα τα pixel που απέχουν -2 από την θέση i , αρκεί να φορτώσουμε τα περιεχόμενα της προσωρινής μνήμης σε ένα καταχωρητή .

Για τον καταχωρητή που περιέχει τα pixel που απέχουν -1 από την θέση i, η διαδικασία που ακολουθείται είναι εξής. Έχοντας φορτώσει τα δεδομένα ,ολισθαίνουμε το πρώτο

καταχωρητή που εμπεριέχει τα pixels από -2 εως 6 κατά 2 θέσεις δεξιά χρησιμοποιώντας την εντολή psrldq. Στην συνέχεια ο δεύτερος καταχωρητής που περιέχει τις τιμές από 7 εώς 11, ολισθαίνει κατά 14 θέσεις αριστερά χρησιμοποιώντας την εντολή pslldq. Τέλος τα περιεχόμενα των δύο προσωρινών καταχωρητών συγχωνεύονται χρησιμοποιώντας την εντολή por. Τα παρακάτω σχήματα δείχνουν την παραπάνω διαδικασία.



Για τον καταχωρητή που περιέχει όλα τα pixels που απέχουν 0 η διαδικασία που ακολουθείται είναι η ίδια. Αλλάζει μόνο ο αριθμός των bits που πρέπει να ολισθήσουν. Δεξιά κατά 4 και αριστερά κατά 12. Στην συγκεκριμένη περίπτωση δε χρειάζεται να δοθούν σχήματα, γιατί είναι περιττό αφού έχει αναφερθεί ήδη στην αρχή και έχει ομοιότητες με το προηγούμενο.

Για τον καταχωρητή που περιέχει όλα τα pixels που απέχουν +1, η διαδικασία που ακολουθείται είναι ομοίως η ίδια, και εδώ αλλάζει το πλήθος των bits που θα πρέπει να ολισθήσουν .Σε αυτή την περίπτωση ολισθαίνετε κατά 6 bits δεξιά ο πρώτος καταχωρητής και κατά 10 αριστερά ο

δεύτερος καταχωρητής . Η διαδικασία ομοίως επαναλαμβάνεται και για τα pixel που απέχουν +2 και +3.

Για τα pixel που απέχουν +2 ο πρώτος καθώς και ο δεύτερος καταχωρητής ολισθαίνουν κατά 8 ,δεξιά και αριστερά αντίστοιχα.

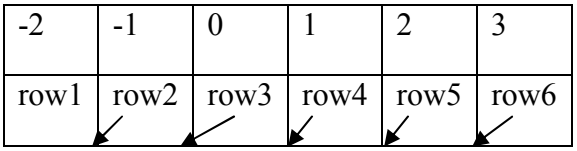
Για τα pixel που απέχουν +3 ο πρώτος καταχωρητής ολισθαίνει κατά 10 και ο δεύτερος κατά 6 .

Αφού δημιουργηθούν οι καταχωρητές με τα κατάλληλα περιεχόμενα συνεχίζει η διαδικασία με την καρδία του interpolation. Για ευκολία θα χρησιμοποιούνται τα ονόματα xmm0,xmm3,xmm4 ,xmm5,xmm6,xmm7,για τις θέσεις -2 -1 0 1 2 3 αντίστοιχα. Η πρώτη πράξη που λαμβάνει χώρα είναι η πρόσθεση του xmm4 και του xmm5 με χρήση της εντολής paddw . Στη συνέχεια το ενδιάμεσο αποτέλεσμα πολλαπλασιάζεται με 4 χρησιμοποιώντας την εντολή psllw. Αποθηκεύοντας το ως ενδιάμεσο αποτέλεσμα [1]. Στην συνέχεια γίνεται η πρόσθεση του καταχωρητή xmm6 με τον καταχωρητή xmm3 χρησιμοποιώντας την εντολή paddw. Το αποτέλεσμα αποθηκεύεται ως ενδιάμεσο [2]. Συνεχίζοντας την διαδικασία αφαιρούνται τα δύο προσωρινά αποτελέσματα [1] [2] χρησιμοποιώντας την εντολή psubw και αποθηκεύεται στην [2]. Το αποτέλεσμα πολλαπλασιάζεται με 4, χρησιμοποιώντας την εντολή psllw. Το αποτέλεσμα αποθηκεύεται στο [1] . Στην συνέχεια γίνεται η πρόσθεση του νέου αποτελέσματος που είναι αποθηκευμένο στον καταχωρητή [1] με αυτού του καταχωρητή [2], με την χρήση της εντολής paddw και το αποτέλεσμα αποθηκεύετε σαν προσωρινό αποτέλεσμα στον καταχωρητή [1]. Τέλος στον προσωρινό καταχωρητή [1] προσθέτουμε τον καταχωρητή xmm0, χρησιμοποιώντας την εντολή paddw και στο αποτέλεσμα αυτό προστίθεται ο xmm7 και αποθηκεύεται εκ νέου στον ίδιο καταχωρητή. Τέλος προστίθεται στον καταχωρητή αυτόν η σταθερά 16 χρησιμοποιώντας την εντολή paddw , η οποία έχει φορτωθεί από ένα πίνακα τύπου short 8 θέσεων. Τέλος γίνεται διαίρεση του αποτελέσματος με το 32 χρησιμοποιώντας αριθμητική ολίσθηση(psraw). Όμως θα πρέπει να περιοριστεί το εύρος των τιμών από 0 -255 . Ο τρόπος για να περιοριστεί το εύρος των τιμών είναι η χρήση της εντολής packuswb. Τέλος με την χρήση της εντολής movdqa ή movq , ανάλογα με το μήκος γράφονται τα pixel εκ νέου στην προσωρινή μνήμη. Και στην συνέχεια με memcpy γράφονται στο block του τρέχοντος καρέ

Στον αρχικό κώδικα η συνάρτηση αυτή δαπανούσε **96600000** κύκλους μηχανής. Μετά τις παραπάνω βελτιστοποιήσεις απαιτεί μόνο **37800000.** Άρα η συνολική επιτάχυνση της συνάρτησης αντιστοιχεί σε **2.55x**, και το ποσοστό βελτίωσης είναι περίπου 61%

4.3.2 Βελτιστοποίηση συνάρτησης για ½ κάθετο interpolation με χρήση ακέραιων pixel(xPredDx0Dy2)

Στην ενότητα αυτή θα δοθεί περιγραφή για τον τρόπο υπολογισμού του ½ pixel όταν για τον υπολογισμό του χρησιμοποιούνται ακέραια pixels. Η είσοδος της συνάρτησης αυτής είναι τύπου 16 bits. Για να είναι εφικτή η χρήση διανυσματικών πράξεων θα πρέπει να αντιγράψουμε τα pixel που χρειαζόμαστε για την παρεμβολή σε μια προσωρινή μνήμη. Τα pixels που απαιτούνται είναι το γινόμενο (uiSizeY+5)*uiSizeX. Αρά χρειαζόμαστε 5 επιπλέον pixels . Η διαδικασία που ακολουθείται είναι πολύ απλή και στηρίζεται στο γεγονός ότι όλα τα pixel και των 6 γραμμών θα πολλαπλασιαστούν με τις ίδιες τιμές για να υπολογιστούν τα ένα pixel σε συγκεκριμένη γραμμή j, όπου j ο αριθμός της γραμμής. Επομένως είναι εύκολο να υπολογίσουμε όλα τα pixels μιας γραμμής χρησιμοποιώντας τον αλγόριθμο της παρεμβολής για το κάθετο interpolation . Οι πράξεις που χρησιμοποιούνται είναι ίδιες με αυτές του interpolation. Για να φορτώσουμε τα δεδομένα, χρησιμοποιούμε τις εντολές movdqa ή movq,ανάλογα με το μήκος του block αναφοράς . Επιπλέον ο αλλαγμένος αλγόριθμος αλλάζει δομή και χρησιμοποιεί μόνο ένα εξωτερικό for-loop για να προσπελαστούν όλες οι γραμμές . Επιπροσθέτως, σε κάθε επανάληψη δε χρειάζεται να φορτώνουμε εκ νέου 5 καινούργιες γραμμές, αφού σε σχέση με την προηγούμενη επανάληψη μόνο η τελευταία στήλη αλλάζει . Για παράδειγμα, αν θέλουμε να υπολογίσουμε το pixel σε μια τυχαία θέση i τότε θα πρέπει να φορτώσουμε τις γραμμές i-2,i-1,i+0,i+1,i+2,i+3. Για την θέση i+1 θα πρέπει να φορτώσουμε τις τιμές i-1,i+0,i+1,i+2,i+3,i+4. Έτσι, αρκεί να αντιγράψουμε τα προηγούμενα δεδομένα σε μια γραμμή πάνω από αυτή που ήδη χρησιμοποιούνταν και να φορτώσουμε στο τελευταίο καταχωρητή την νέα τιμή .Το παρακάτω σχήμα δείχνει την λογική αύτη .



row2	row3	row4	row5	row6	row7

Η αποθήκευση γίνεται όπως στην προηγούμενη περίπτωση για το οριζόντιο ½. Στο αρχικό κώδικα η συνάρτηση για να ολοκληρώσει την ζωή της με επιτυχία απαιτούσε συνολικά **2919000000** κύκλους μηχανής .Η βελτιστοποιημένη έκδοση της συνάρτησης απαιτεί **75600000 .** Άρα η συνολική επιτάχυνση είναι 3.85x .Σε ποσοστό επί του αρχικού κώδικα είναι 73%.

4.3.3 Βελτιστοποίηση της συνάρτησης για τον υπολογισμό του ½ pixel interpolation με χρήση τόσο ακέραιων pixels όσο και ½ pixels (xPredDx2Dy2)

Στην ενότητα αυτή θα περιγραφεί η βελτιστοποίησης της συνάντησης που υπολογίζει τα ½ pixels που βρίσκονται ανάμεσα σε ½ pixels καθώς και σε ακέραια pixels . Όπως και στις προηγούμενες περιπτώσεις και σε αυτή την η είσοδος των δεδομένων είναι μήκους 16 bit .Σε αυτή την περίπτωση η διαδικασία διαφέρει σε σχέση με τις άλλες 2. Αρχικά το πλήθος των pixels που απαιτούνται για να ολοκληρωθεί η εκτέλεση της διαδικασίας είναι (uiSizeY+5)*(uiSizeX+5). Επιπλέον τα περιεχόμενα του block που χρειάζονται αντιγράφονται σε μια τοπική μνήμη ,που περιέχει όλα τα pixels ξεκινώντας από την γραμμή i-2 και την στήλη j-2. Η τοπική αυτή προσωρινή μνήμη περνάει ως είσοδο σε μια συνάρτηση που υπολογίζει τα ½ pixels που βρίσκονται ανάμεσα σε ακέραια pixels. Η διαδικασία που ακολουθείται είναι ακριβώς ίδια με αυτή της ενότητας 4.3.1, όμως στο τέλος δε γίνεται περιορισμός του εύρους τιμών από 0 -255 διότι αυτά είναι προσωρινά αποτελέσματα. Τα αποτελέσματα αυτά αποθηκεύονται σε μια τοπική αποθήκη [1]. Στην συνέχεια για την ολοκλήρωση της διαδικασίας θα χρησιμοποιηθεί pmaddw διότι η εντολή αυτή παράγει από 8x16 bits 4x32 bits ,που είναι απαραίτητο σε αυτή την περίπτωση . Όπως έχει αναφερθεί στην αρχή της ενότητας 4.3 τα φίλτρα που χρησιμοποιούνται είναι [1,-5,20,20,-5,1]. Προφανώς δε χρειάζεται να πολλαπλασιάσουμε με την τιμή 1. Σε προηγούμενη ενότητα έχει περιγραφεί η εντολή pmaddw και πλέον είναι προφανές ότι ένας καταχωρητής δε μπορεί να έχει και τις 4 τιμές, δηλαδή το [-5,20,-5,20] για την χρησιμοποίηση της εντολής pmaddw. Αλλά θα οριστούν δύο νέοι πίνακες τύπου short 8 θέσεων με τις εξής τιμές [**20 20 20 20 20 20 20 20**], **[-5 -5 -5 -5 -5 -5 -5 -5] [2].** Με χρήση της εντολής movdqa φορτώνονται οι σταθερές αυτές στους κατάλληλους καταχωρητές. Η προσωρινή μνήμη [1] περιέχει στη θέση 0 τα pixels που απέχουν -2 οριζόντια

από την θέση j, στην θέση 16 τα pixels που απέχουν -1 οριζόντια από την θέση j , στην θέση 32 τα pixels που απέχουν 0 απο την οριζόντια θέση j, στην θέση 48 τα pixels που απέχουν +1 από την οριζόντια θέση j ,στην θέση 60 τα pixels που απέχουν +2 από την οριζόντια θέση j και τέλος στην θέση 80 τα pixels που απέχουν +3 από την οριζόντια θέση. Τα συνολικά pixels που χρειάζεται για να ολοκληρωθεί η διαδικασία είναι (uiSizeY)*(uiSizeX+5) τα οποία προφανώς έχουν αποθηκευτεί στην προσωρινή μνήμη [1]. Στην συνέχεια πρέπει να φορτώσουμε τις κατάλληλες τιμές στους καταχωρητές χρησιμοποιώντας τις εντολές movdqa. Για ευκολία θα χρησιμοποιηθούν τα εξής ονόματα xmm4 για όλα τα pixels που απέχουν -2 , xmm2 για αυτά που απέχουν -1 ,xmm0 για αυτά που απέχουν +0, xmm1 για αυτά που απέχουν +1, xmm3 για αυτά που απέχουν +2 και τέλος xmm5 για αυτά που απέχουν +3. Αρχικά πρέπει να δημιουργήσουμε τις τιμές που θα πολλαπλασιαστούν με το φίλτρο **[20 20 20 20 20 20 20 20] [1].** Για να δημιουργήσουμε τον καταχωρητή που εμπεριέχει τις κατάλληλες τιμές αρκεί να χρησιμοποιήσουμε την εντολή **punpcklwd** με ορίσματα xmm0, xmm1 για να φορτώσουμε τα pixels από 0 -4 και την εντολή **punpckhwd** για να φορτώσουμε τα pixels από 4-8 . Έστω ότι ο καταχωρητής xmm0 έχει τις τιμές [a b c d e f g h] και ο καταχωρητής xmm1 [b c d e f g h i]. Χρησιμοποιώντας την εντολή **punpcklwd** προκύπτει ο καταχωρητής με τα εξής περιεχόμενα

a	b	b	c	c	d	d	e

Ο παραπάνω καταχωρητής μαζί με το φίλτρο [1], οδηγούνται ως είσοδος στην εντολή pmaddw και αποθηκεύονται τα αποτελέσματα σε ένα προσωρινό καταχωρητή [3]. Συνεχίζοντας θα πρέπει να δημιουργηθεί ο κατάλληλος καταχωρητής που θα είναι η πρώτη είσοδος που μαζί με το φίλτρο [2] οδηγούν την εντολή pmaddw. Η λογική είναι η ίδια. Χρησιμοποιούμε τις εντολές **punpcklwd punpckhwd.** Το αποτέλεσμα της pmaddw αποθηκεύεται σε ένα προσωρινό καταχωρητή [4]. Έπειτα, οι δύο προσωρινοί καταχωρητές [3] [4] προστίθενται με χρήση της εντολής paddd και το αποτέλεσμα αποθηκεύεται στον [3] . Στην συνέχεια ανάλογα αν χρειαζόμαστε τα pixels 0-4 ή 5-8 του καταχωρητή xmm4 χρησιμοποιούμε την εντολή **punpcklwd ή punpckhwd .**Το αποτέλεσμα της εντολής αυτής αποθηκεύεται στον [4]. Συνεχίζοντας προσθέτουμε τους [3] και [4] και το αποτέλεσμα αποθηκεύεται στον [3] . Η ίδια διαδικασία που ακολουθήθηκε για τον xmm4 ακολουθείται και για τον xmm5. Τέλος προστίθεται ο [3] και [4] . Στο τελικό αποτέλεσμα που έχει αποθηκευτεί στον [3] προσθέτουμε

την σταθερά 512 ,που έχει φορτωθεί σε έναν καταχωρητή από ένα πίνακα τύπου int 4 θέσεων ,και το αποτέλεσμα αποθηκεύεται στον [3]. Το αποτέλεσμα αυτό διαιρείται με 1024 χρησιμοποιώντας την εντολή psraw .Έπειτα περιορίζουμε το εύρος από [0-255]. Χρησιμοποιώντας τις εντολές packssdw και packuswb.

Στο τέλος της διαδικασίας αν το uiSizeX =4 γράφουμε τα δεδομένα στο τρέχων block ,χρησιμοποιώντας την movq . Σε κάθε άλλη περίπτωση ενώνουμε τα δεδομένα στις θέσεις 0 -4 και 4-8 . Η διαδικασία είναι απλή, επειδή οι πράξεις έγιναν με 32x4 ,όταν τα αποτελέσματα "πακεταρίστηκαν" σε 16x8 στα υψηλότερης σημαντικότητας bits έχουν 0. Επομένως αν στο δεύτερο αποτέλεσμα γίνει ολίσθηση κατά 8 bits και στην συνέχεια χρησιμοποιηθεί η εντολή por έχουμε ένα καταχωρητή με τα αποτελέσματα από 0-8. Στο τέλος γράφουμε τα αποτελέσματα πίσω στην προσωρινή μνήμη χρησιμοποιώντας την movdqa και στην εξωτερική χρησιμοποιώντας memcpy.

Στον αρχικό κώδικα η συνάρτηση **xPredDx2Dy2** δαπανούσε **81900000** κύκλους μηχανής. Η βελτιστοποιημένη συνάρτηση καταναλώνει **35700000**. Η συνολική βελτιστοποίηση είναι **2.29** x. Σε ποσοστό είναι 56.5 %. Στην βελτιστοποίηση αυτή δε έχει υπολογιστεί η συνάρτηση που υπολογίζει τα ½ pixels με βάση την θέση των ακέραιων pixels, γιατί καλείται και από άλλα σημεία.

4.3.4 Βελτιστοποίηση συνάρτησης για ¼ οριζόντιο interpolation(xPredDy0Dx13)

Στη ενότητα αυτή θα αναλυθεί η διαδικασία βελτιστοποίησης του ¼ οοριζόντιου interpolation. Όπως ήδη αναφέρθηκε για την παρούσα διαδικασία χρησιμοποιείται γραμμική παρεμβολή. Όπως και στις προηγούμενες περιπτώσεις η είσοδος είναι τύπου short . Για την περίπτωση αυτή χρειάζονται uiSizeY*(uiSizeX+5) pixels. Τα απαραίτητα pixels για την διαδικασία αντιγράφονται σε μια προσωρινή μνήμη. Για το ¼ οριζόντιο interpolation τα pixels που απαιτούνται προκύπτουν ως εξής: Το πρώτο όρισμα δημιουργείται από το οριζόντιο ½ horizontal interpolation όπως αυτό περιγράφεται στην ενότητα 4.3.1.Το δεύτερο όρισμα προκύπτει ανάλογα με την τιμή του υπολοίπου της διαίρεσης της οριζόντιας συντεταγμένης του motion vector με το 4. Αν το dx ==0 τότε πρέπει να δημιουργηθεί ένας καταχωρητής που περιέχει τα pixels που απέχουν 0 από το pixel αναφοράς. Αν dx ==3 τότε πρέπει να δημιουργηθεί ένας καταχωρητής που περιέχει τα pixels που απέχουν +1 από το pixels αναφοράς .Για την διαδικασία της γραμμικής παρεμβολής προσθέτουμε τα 2 ορίσματα, με την εντολή

paddw. Στην συνέχεια προσθέτουμε την σταθερά 1 ,την οποία φορτώνουμε με χρήση της εντολής movdqa . Τέλος διαιρούμε το αποτέλεσμα με 2 ,χρησιμοποιώντας την εντολή psraw.

Η μη βελτιστοποιημένη συνάρτηση καταναλώνει **207900000** κύκλους μηχανής**.** Η βελτιστοποιημένη έκδοση της συνάρτησης καταναλώνει **81900000** κύκλους μηχανής. Η συνολική επιτάχυνση 2.54x, που αντιστοιχεί σε ποσοστό **61 %** .

4.3.5 Βελτιστοποίηση συνάρτησης για ¼ κάθετο interpolation(xPredDx0Dy13)

Στην ενότητα αυτή θα περιγραφεί η διαδικασία του κάθετου ¼ interpolation .Τα pixels που χρειάζονται για την μέθοδο αυτό είναι το πολλαπλάσιο του (uSizeY+5)*uiSizeX και αντιγράφονται σε μια προσωρινή μνήμη που πρέπει να είναι ευθυγραμμισμένη . Όπως και για το οριζόντιο ¼ interpolation και το κάθετο χρησιμοποιεί γραμμική παρεμβολή. Τα ορίσματα που χρειάζονται για την γραμμική παρεμβολή προκύπτουν ως εξής. Το πρώτο όρισμα προκύπτει από την μέθοδο «½ vertical interpolation», όπως αυτή περιγράφεται στην ενότητα 4.3.2, το δεύτερο δίνεται από το όρισμα dy που είναι το υπόλοιπο της κάθετης συντεταγμένης με το 4. Συγκεκριμένα η θέση της γραμμής που πρέπει να χρησιμοποιηθεί για το interpolation στο αρχικό κώδικα δίνεται από την σχέση (iDy>>1) * iSrcStride. Επειδή όμως για το interpolation στην βελτιστοποιημένη συνάρτηση χρησιμοποιείται μια προσωρινή μνήμη η οποία αντιγράφει τα δεδομένα απο την γραμμή που απέχει από το pixel αναφοράς -2, πρέπει να να υπολογίσουμε το dy με νέα σχέση. Συγκεκριμένα για uiSizeX=4, υπολογίζεται απο την σχέση **iDy = (iDy>>1) * 4+8**. Για την τιμη uiSizeX=8 ,υπολογίζεται απο την σχέση **iDy = (iDy>>1) * 8+16.** Για την τιμή uiSizeX =16, υπολογίζεται απο την σχέση **iDy = (iDy>>1) * 16+32.** Έχοντας υπολογίζει το iDy σε κάθε επανάληψη φορτώνουμε την κατάλληλες τιμές που είναι απαραίτητες για το ¼ interpolation .Για παράδειγμα για uiSizex=4 η γραμμή που πρέπει να φορτωθεί σε κάθε επανάληψη δίνεται απο την σχεση *(scratch_mem+ iDy+(y<<2)). Οι πράξεις που ακολουθούν εφόσον έχουμε φορτώσει στους καταχωρητές τις κατάλληλες τιμές είναι οι ίδιες με αυτές της μεθόδου στην ενότητα 4.3.4 . Η μη βελτιστοποιημένη συνάρτηση δαπανούσε **571200000** κύκλους μηχανής . Μετά την βελτιστοποίηση οι κύκλοι μηχανής μειώθηκαν σε **163800000 .** Άρα η συνολική επιτάχυνση είναι **3.49 x**, δηλαδή επιτεύχθηκε βελτιστοποίηση σε ποσοστό περίπου 71.8%.

4.3.6 Βελτιστοποίηση συνάρτησης για ¼ οριζόντιο interpolation με χρήση ½ pixels που υπολογίζονται από ακέραια και ½ pixels(xPredDx2Dy13)

Στη ενότητα αυτή θα περιγραφεί η διαδικασία ¼ interpolation με χρήση ½ pixels που βρίσκονται ανάμεσα σε ακέραια pixels καθώς και σε pixels ½. Τα pixel, ο τύπος των οποίων είναι short, που απαιτούνται να αντιγραφούν σε μια προσωρινή μνήμη για να είναι εφικτή η εκτέλεση διανυσματικών πράξεων είναι το γινόμενο του (uiSizeY+5)*(uiSizeX+5). Ομοίως με τις 2 προηγούμενες ενότητες και εδώ για το ¼ interpolation χρησιμοποιείται γραμμική παρεμβολή. Τα ορίσματα της γραμμικής παρεμβολής προκύπτουν ως εξής. Αρχικά το πρώτο όρισμα προκύπτει από την ίδια ακριβώς διαδικασία της ενότητας 4.4.3, δηλαδή ο υπολογισμός ½

pixels που βρίσκονται ανάμεσα σε ½ pixels αλλά και σε ακέραια pixels. Στην συνέχεια το δεύτερο όρισμα εξαρτάται από το idy. Αν το idy έχει τιμή 1, ως δεύτερο όρισμα ,χρησιμοποιείται ένας καταχωρητής που περιέχει όλα τα pixels που απέχουν 0 από το pixel αναφοράς. Σε οποιαδήποτε άλλη περίπτωση φορτώνονται τα pixels της γραμμής που απέχουν+1 από το pixels αναφοράς . Η φόρτωση γίνεται με την χρήση της εντολής movdqa ή movq. Στην συνέχεια, οι πράξεις που ακολουθούν είναι ίδιες με αυτές για το ¼ οριζόντιο interpolation .

Στο αρχικό κώδικα η συνάρτηση αυτή απαιτούσε για την εκτέλεση της **212100000** κύκλους μηχανής. Μετά την βελτιστοποίηση οι κύκλοι μηχανής μειώθηκαν στους **115500000.** Η συνολική επιτάχυνση είναι **1.8 x** ,δηλαδή 46 % επί του αρχικού κώδικα

4.3.7 Βελτιστοποίηση της συνάρτησης xPredDx2

Η συνάρτηση αυτή καλείται από την συνάρτηση xPredDx2Dy13 και την συνάρηση xPredDx2Dy2. Η περιγραφή της έχει δοθεί στις αντίστοιχές ενότητες. Η συνολική βελτιστοποίηση που επιτευχθεί στην συνάρτηση αυτή είναι 2.9 x, που αντιστοιχεί σε ποσοστό 69 %.

4.3. 8 Βελτιστοποίηση συνάρτησης για ¼ κάθετο interpolation με χρήση ½ pixels που υπολογίζονται από ακέραια και ½ pixels (xPredDy2Dx13)

Η είσοδος που δέχεται η συνάρτηση αυτή είναι μια περιοχή (block) που περιέχει ένα αριθμό pixels, τα οποία αναπαριστούνται με 16 bits. Τα συνολικά pixels που πρέπει να αντιγραφούν στην προσωρινή ευθυγραμμισμένη μνήμη είναι το γινόμενο (uiSizeY+5)*(uiSizeX+6). Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι το πρώτο pixel που αντιγράφεται στην θέση [0,0] βρίσκεται στην θέση [-2 -2] στο καρέ αναφοράς . Στην πρώτη φάση του αλγορίθμου εκτελείται ½ κάθετο interpolation όπως αυτό περιγράφτηκε στην ενότητα 4.3.2, χωρίς να γίνει περιορισμός στο εύρος των τιμών από 0 -255. Άρα κάθε γραμμή δημιουργούνται pixels στις θέσεις ½ τα οποία αποθηκεύονται σε προσωρινή μνήμη. Στην συνέχεια τα νέα pixels στις θέσεις ½ υφίστανται οριζόντιο interpolation. Ο τρόπος που θέλουμε τα pixels να είναι μέσα στους 5 καταχωρητές που θα χρησιμοποιηθούν φαίνεται στους παρακάτω πίνακες.

a	b	c	d	e	f
b	c	d	e	f	g
c	d	e	f	g	h
d	e	f	g	h	i

e	f	g	h	i	j

Στην ενότητα 4.3.1 έχει αναφερθεί πως θα δημιουργηθούν καταχωρητές με τα συγκεκριμένα περιεχόμενα . Για ευκολία θα θεωρήσουμε το καταχωρητή xmm0 που θα απέχει -2 από το pixel αναφοράς, xmm3 που απέχει -1 από το pixel αναφοράς ,xmm4 που απέχει 0 ,xmm5 που απέχει +1, xmm6 που απέχει +2, xmm7 που απέχει +3 από το pixel αναφοράς. Το οριζόντιο interpolation θα γίνει με χρήση της εντολής pmaddw. Το φίλτρο που χρησιμοποιείται είναι [1 -5 20 20 -5 1]. Προφανώς για τις τιμές 1 δε χρειάζεται να χρησιμοποιήσουμε pmaddw. Για τις υπόλοιπες τιμές του φίλτρου δημιουργούμε δυο νέους καταχωρητές στους οποίους φορτώνουμε τις σταθερές [20 20 20 20 20 20 20 20] [1],[-5 -5 -5 -5 -5 -5 -5 -5] [2]. Έπειτα πρέπει να δημιουργήσουμε τους καταχωρητές που θα χρησιμοποιηθούν ως είσοδο στην pmaddw .Για το φίλτρο [1], που θα αποτελέσει την δεύτερη είσοδο της pmaddw, χρησιμοποιούμε την εντολή **punpcklwd ή punpckhwd** (ανάλογα ποια pixels θέλουμε) με είσοδο τους καταχωρητές xmm4 , xmm5. Για το φίλτρο [2], οι καταχωρητές που πρέπει να συνδυαστούν με χρήση της εντολής **punpcklwd ή punpckhwd** είναι οι xmm3 και xmm6. Στην συνέχεια τα αποτελέσματα από τις εντολές pmaddw, προστίθενται με χρήση της εντολής paddw και προκύπτει το προσωρινό αποτέλεσμα [3]. Ανάλογα με ποια pixels απαιτούνται χρησιμοποιούμε την εντολή **punpcklwd ή punpckhwd,** για να πάρουμε τα κατάλληλα pixels από τους καταχωρητές xmm0,xmm7. Προστίθενται με την χρήση της εντολής paddw με [3] και το αποτέλεσμα αποθηκεύτηκε εκ νέου στο [3]. Με την χρήση της εντολής paddw προστίθεται η σταθερά 512 στο [3] .Τέλος διαιρείται με το 1024, με χρήση της εντολής psraw και γίνεται περιορισμός του εύρους των τιμών από 0 -255 με χρήση των εντολών **packssdw και packuswb** . Η διαδικασία του υπολογισμού του ½ pixel ολοκληρώνεται. Στην συνέχεια θα πρέπει να υπολογιστεί και το ¼ interpolation .Αρχικά αν το μήκος του block αναφοράς είναι μεγαλύτερο του 4 θα πρέπει να γίνει συγχώνευση των τελικών αποτελεσμάτων για τα χαμηλότερα και υψηλότερα pixels. Η διαδικασία είναι απλή, επειδή οι πράξεις έγιναν με 32x4, όταν τα αποτελέσματα "πακεταρίστηκαν" σε 16x8 στα υψηλότερης σημαντικότητας bits έχουν 0. Επομένως αν στο δεύτερο αποτέλεσμα γίνει ολίσθηση κατά 8 bits και στην συνέχεια χρησιμοποιηθεί η εντολή por έχουμε ένα καταχωρητή με τα αποτελέσματα από 0-8. Τώρα έχει δημιουργηθεί το ένα όρισμα για γραμμική παρεμβολή .Το άλλο όρισμα εξαρτάται από την τιμή του idx . Αν το idx ισούται με 2, τότε ως δεύτερο όρισμα χρησιμοποιείται ο καταχωρητής xmm4. Σε κάθε άλλη περίπτωση

χρησιμοποιείται ο xmm5. Η μη βελτιστοποιημένη συνάρτηση χρειαζόταν για την ολοκλήρωση της **470400000** κύκλους μηχανής**.** Η βελτιστοποιημένη συνάρτηση χρειάζεται μόνο **151200000.** Επομένως είναι 3.1 φορές πιο γρήγορη και σε ποσοστό περίπου 69 %.

4.3.9 Βελτιστοποίηση της συνάρτησης που υπολογίζει την τιμή ενός pixel όταν η θέση του είναι διαφορετική από τις παραπάνω περιπτώσεις ,διαγώνιο interpolation ¼ (xPredElse)

Στην ενότητα αυτή θα περιγράψουμε την τελευταία συνάρτηση που εκτελείται όταν δε πληρούνται οι προϋποθέσεις για να εκτελεστεί μια από τις συναρτήσεις που περιγράφηκαν στις προηγούμενες ενότητες. Στην συγκεκριμένη περίπτωση το πλήθος των pixel που απαιτούνται να αντιγραφούν στην ευθυγραμμισμένη προσωρινή μνήμη είναι (uiSizeX+6)*(uiSizeY+6). Στη παρούσα φάση γίνεται μια διαδικασία κάθετου ½ interpolation και ½ οριζόντιου interpolation και στο τέλος μια γραμμική παρεμβολή για το ¼ interpolation.

Για το οριζόντιο interpolation η γραμμή του block από την οποία θα ξεκινήσει το interpolation εξαρτάται από idy. Αν το idy είναι ίσο με 1 τότε το οριζόντιο interpolation ξεκινά από την γραμμή 0 αλλιώς από την γραμμή 1. Η διαδικασία είναι ίδια με αυτή που περιγράφηκε στην ενότητα 4.3.1, αλλά ανάλογα με την τιμή του idy φορτώνουμε από την προσωρινή μνήμη τα κατάλληλα pixels, έχοντας υπόψη ότι πρέπει να είναι μετατοπισμένη η θέση μνήμης κατά 2, διότι η προσωρινή μνήμη έχει αποθηκευμένα τα pixels του block ξεκινώντας από την θέση [-2 -2] ενώ στην συγκεκριμένη περίπτωση απαιτείται η μετατόπιση έτσι ώστε να χρησιμοποιηθούν τα pixels από την θέση [-2 0]. Το αποτέλεσμα αποθηκεύεται στον προσωρινό καταχωρητή [1].

Για το κάθετο interpolation τα pixels κάθε γραμμής που θα χρησιμοποιηθούν εξαρτώνται από το idx. Αν το idx ισούται με 1, για το interpolation χρησιμοποιούνται όλα τα pixels της γραμμής, ξεκινώντας από την θέση 0. Αλλιώς χρησιμοποιούνται τα pixels ξεκινώντας από την θέση 1. Η μέθοδος που ακολουθείται είναι ίδια με το κάθετο ½. Όμως επειδή η προσωρινή μνήμη αποθήκευσης περιέχει τα δεδομένα του block αναφοράς ξεκινώντας από την θέση [-2 -2] θα πρέπει να χρησιμοποιήσουμε διανυσματικές εντολές έτσι ώστε να μετατοπιστούν τα δεδομένα στην θέση [0 -2] ή [1 -2] που είναι η σωστή θέση για το κάθετο interpolation. Για μήκος του block ίσο με 4 αφού φορτωθούν όλες οι τιμές στους καταχωρητές με χρήση της εντολής movdqa (παραπομπή ενότητα 4.3.2) ,στην συνέχεια αν το idx είναι ίσο με 1 τότε αρκεί μια δεξιά ολίσθηση κατά 4 όλων των καταχωρητών, αλλιώς αν το idx δεν είναι 1 τότε αρκεί μια δεξιά ολίσθηση κατά 6 (εντολή psrldq) . Για μήκος του block μεγαλύτερο του 4 πχ 8, τότε

έχοντας φορτώσει τα pixels από 0 -8 και τα pixels 8 -16 για όλες τις γραμμές τότε για idx ίσο με 1 απαιτείται να ολισθήσουμε τον πρώτο καταχωρητή κατά 4 bits δεξιά και το δεύτερο καταχωρητή κατά 12 bits αριστερά pslldq,η συγχώνευση γίνεται με την εντολή por. Στην συνέχεια για idx διάφορο του 1 θα πρέπει να ολισθήσουν τα pixels του πρώτου καταχωρητή κατά 6 θέσεις δεξιά και του δεύτερου κατά 10 θέσεις αριστερά και η συγχώνευση γίνεται με την εντολή por. Για μήκος 16 η διαδικασία είναι παρόμοια, όμως θα πρέπει να φορτώσουμε και τα pixels από 16 έως 22. Στο τέλος της παραπάνω διαδικασίας οι καταχωρητές έχουν τα κατάλληλα pixels και εκτελείται ½ κάθετο interpolation όπως αυτό περιγράφεται στην ενότητα 4.3.2. Η φόρτωση των καταχωρητών για κάθε επανάληψη γίνεται με την ίδια μέθοδο με αυτή της ενότητας 4.3.2. Το αποτέλεσμα του κάθετου ½ interpolation αποθηκεύεται σε ένα προσωρινό καταχωρητή [2]. Τελειώνοντας τα αποτελέσματα 1 και 2 τα παρεμβάλουμε γραμμικά όπως ορίζει το ¼ interpolation και τα αποτελέσματα αποθηκεύονται από την προσωρινή μνήμη στο τρέχων block. Πριν την βελτιστοποίηση της συνάρτησης, αυτή δαπανούσε συνολικά **1241100000** κύκλους μηχανής**.** Μετά τις αλλαγές ο συνολικός χρόνος που δαπανά είναι **514500000** κύκλους μηχανής**,** είναι η συγκεκριμένη συνάρτηση 2.41x πιο γρήγορη .Σε ποσοστό 59 % .

4.3.10 Συνολική βελτιστοποίηση για το Luma

Στον ακόλουθο πίνακα συνοψίζεται για κάθε συνάρτηση ο χρόνος σε κύκλους μηχανής που απαιτείται για την μη βελτιστοποιημένη έκδοση και για την βελτιστοποιημένη.

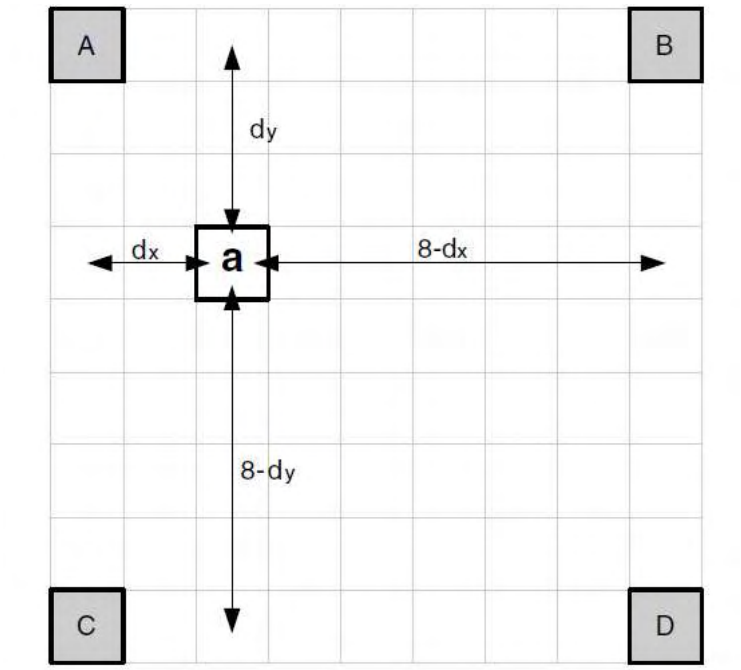
Συνάρτηση	**Μη βελτιστοποιημένη**	**Βελιστοποιημένη**	**Επιτάχυνση x**
xPredElse	**1031100000**	**424200000**	**2.43**
xPredDy0Dx2	**96600000**	**37800000**	**2.55**
xPredDx0Dy2	**291900000**	**75600000**	**3.85**
xPredDx2Dy2	**81900000**	**35700000**	**2.29**
xPredDy0Dx13	**207900000**	**81900000**	**2.54**

xPredDx2Dy13	**212100000**	**115500000**	**1.8**
xPredDx2	**226800000**	**71400000**	**2.9**
xPredDx0Dy13	**571200000**	**163800000**	**3.48**
xPredDy2Dx13	**470400000**	**151.200.000**	**3.1**
Άθροισμα	**3244500000**	**1148700000**	**2.82**

Από τον παραπάνω πίνακα διακρίνεται ότι η συνολική επιτάχυνση είναι 2.82 x που αντιστοιχεί σε ποσοστό 65 % .

4.4 Motion Compensation για το χρώμα (Συνιστώσες Cb, Cr)

Όπως ήδη αναφέρθηκε στην ενότητα 4.3 η διαδικασία του motion compensation για το χρώμα είναι μια απλή διαδικασία . Η μέθοδος που χρησιμοποιείται για την διαδικασία αυτή είναι διγραμμική παρεμβολή. Η διγραμμική παρεμβολή για το χρώμα γίνεται με ακρίβεια 1/8. Η θέση ενός pixel καθορίζει αν η παρεμβολή που θα γίνει αφορά τα οριζόντια, τα κάθετα ή τα διαγώνια pixels. Το σχήμα που ακολουθεί δείχνει την θέση ενός pixel και τον τρόπο που υπολογίζεται η τιμή του.



4.4.1 Motion Compensation για το χρώμα

Η υλοποίηση της κάθετης, της οριζόντιας και της διαγώνιας παρεμβολής γίνεται μέσα από την ίδια συνάρτηση **(xPredChromaPel)**. Ένα block χρώμα έχει μέγεθος 64 pixels ή διαστάσεις 8x8 . Κάθε υποδιαίρεση ενός block luma αντιστοιχίζεται στην αντίστοιχη υποδιαίρεση block χρώμα, διαιρώντας τις διαστάσεις με 2. Για παράδειγμα ένα block luma 8x16 αντιστοιχεί σε ένα block 4x8 και αντίστοιχα ένα block 4x8 σε ένα block 4x2. Ο τύπος που δίνει την τιμή ενός pixel στην διγραμμική παρεμβολή είναι για δύο pixels a, b και τιμές του φίλτρου x, y, δίνεται σύμφωνα με το ορισμό της γραμμικής παρεμβολής αλλά και τον αρχικό κώδικα από τον τύπο (x*a+y*b+4)>>3. Προφανώς επειδή y=8-x μπορεί να γραφτεί σαν (x*(a-b) +8b +4)>>3. Η συγκεκριμένη αλλαγή βελτιώνει την απόδοση ,διότι δε χρειάζεται να γίνουν 2 πολλαπλασιασμοί, που γενικά είναι αργές πράξεις, αλλά ένας πολλαπλασιασμός και μια ολίσθηση με το 3. Η συνάρτηση xPredChromaPel, δέχεται σαν είσοδο μια περιοχή αναφοράς. Ο αριθμός των pixels που πρέπει να χρησιμοποιηθούν εξαρτάται από το τύπο του block και την θέση του pixel και στις 3 περιπτώσεις επιβάλεται η αντιγραφή σε μια ευθυγραμισμένη τοπική μνήμη. Επιπλεόν κάθε μια απο 3 περιπτώσεις, χωρίζεται σε υποπεριπτώσεις ανάλογα με το μήκος του block. Στην συνέχεια θα περιγραφεί η διαδικασία της διαγραμμικής παρεμβολής

.Όπου γινεται αναφορα στην μεταβλητή iSizeX, είναι το μήκος του block και η αναφορά στο iSizeY είναι είναι το πλάτος. Το πιθανό μήκος ενός block, μπορεί να είναι είτε 2 είτε 4 είτε 8.

Αν το υπόλοιπο της οριζόντιας συντεταγμένης του motion vector με το 8 είναι 0 και το υπόλοιπο της κάθετης συντεταγμένης του motion vector με το 8 είναι διαφορετική του 0, τότε εκτελείται κάθετη διγραμμική παρεμβολή. Για την κάθετη παρεβολή χρειαζόμαστε pixels, ο αριθμός των οποιών ισούτε με το γινόμενο (iSizeX)*(iSizeY+1). Για το κάθετο interpolation λαμβάνεται υπόψιν ότι τα pixels μιας γραμμής, απέχουν όλα την ίδια απόσταση απο το pixel αναφοράς, άρα είναι δυνατόν να γίνει υπολογίζεται σε κάθε επανάληψη όλα τα οριζόντια pixels που παρεμβάλονται κάθετα. Σε πρώτη φάση θα πρέπει να φορτωθούν τα δεδομένα κάθε γραμμής που απαιτούνται για την διαδικασία τηςπαρεμβολής. Η φόρτωση ανάλογα με το μήκος γίνεται είτε με την εντολή movdqa ή την εντολή movq. Στην συνέχεια γίνονται οι κατάλληλες πράξεις για τον υπολογισμό του pixel αναφοράς. Για ευκολία θα χρησιμοποιούνται οι ονομασίες των καταχωρητών xmm0 και xmm1 που αντιστοιχούν στα pixels της γραμμής i και i+1. Αρχικά χρησιμοποιείται η πράξη της αφαίρεσης για τους καταχωρητες xmm0,xmm1, psubw που αποθηκεύεται στο προσωρινό αποτέλεσμα [1]. Έπειτα, θα πρέπει να πολλαπλασιαστεί με το κατάλληλο φίλτρο. Όλα τα pixels μια γραμμής πολλαπλασιάζονται με την ίδια τιμή, άρα αρκεί να οριστεί ένα πίνακας μήκους 128 x16 bits, από την i θέση μέχρι την θέση i+7, θα έχει επαναλβανόμενη την τιμή i+1. Ο πολλαπλασιασμός γίνεται με την χρήση της εντολής pmullw και αποθηκεύεται στο προσωρινό αποτέλεσμα [1] . Ο καταχωρητής xmm1 πολλαπλασιάζεται με 8 χρησιμοποιώντας την εντολή psllw και αποθηκεύεται στο προσωρινό αποτέλεσμα [2]. Τα 2 προσωρινά αποτελέσματα προστίθενται με την εντολή paddw. Τέλος με την χρήση της ίδιας εντολής προστίθεται η σταθερά 4 και διαιρείται το αποτέλεσμα με 8 ,χρησιμοποιώντας την εντολή psraw. Το τελικό αποτέλεσμα γράφεται στο τρέχων block. Θεωρούμε μήκος block 4 .

Στην περίπτωση την οποία το υπόλοιπο της διαίρεσης της κάθετης συνιστώσας του motion vector με το 8 είναι 0 και της οριζόντιας διάφορο του 0 τότε εκτελείται οριζόντια παρεμβολή. Το σύνολο των pixels που απαιτούνται είναι ίσο με το γινόμενο (iSizeY)*(iSizeX+1). Για μήκος του τρέχοντος block 2 ή 4 χρησιμοποιείται η εντολή movqa για φόρτωση των δεδομένων. Για μήκος 8 η εντολή movdqa. Σε κάθε καταχωρητή πρέπει να υπάρχουν τα κατάλληλα περιεχόμενα . Αν θεωρήσουμε μήκος block 4 και τα pixels **[a b c d e]** ,τότε τα περιεχόμενα των καταχωρητών xmm0 και xmm1 θα πρέπει να είναι αυτά που φαίνονται στον παρακάτω πίνακα

a	b	c	d
b	c	d	e

Για να δημιουργήσουμε αυτά τα περιεχόμενα θα πρέπει να ακολουθήσουμε την διαδικασία που περιγράφεται στην συνέχεια .Αρχικά για μήκος block που ισούται με 2 θα πρέπει να ολισθήσουμε τον καταχωρητή xmm0 κατά 2 θέσεις για να προκύψει ο καταχωρητής xmm1. Αν το μήκος είναι 4 ή 8 ,τότε φορτώνουμε σε δύο καταχωρητές τα χαμηλότερα και υψηλότερα pixels και ολισθαίνουμε τον καταχωρητή xmm0 κατά 2 και τον xmm1 κατά 14 θέσεις. Συγχωνεύουμε τα αποτελέσματα χρησιμοποιώντας την εντολή por. Στην συνέχεια πρέπει να φορτώσουμε σε κατάλληλο καταχωρητή τα κατάλληλα φίλτρα. Επαναλαμβάνουμε την ίδια διαδικασία με αυτή της κάθετης παρεμβολής .

Στην τελευταία περίπτωση που τα υπόλοιπα της διαίρεσης με το 8 και των 2 συνιστωσών είναι διάφορα του 0 και δεν ισχύει καμία από τις παραπάνω περιπτώσεις, λαμβάνει χώρα η διαγώνια παρεμβολή. Σε αυτήν την περίπτωση το πλήθος των pixels που χρειάζονται είναι (iSizeY+1)*(iSizeX+1). Η διαδικασία που ακολουθείται είναι: Για την περίπτωση που για το υπολογισμό της τιμής ενός pixel πρέπει να χρησιμοποιηθούν το pixel στη θέση 0 και στην θέση 1 υπολογίζεται αρχικά μια τιμή που δίνεται από τον τύπο **xF0* (yF0 * pucSrc[0] + yF1 * pucSrc[iSrcStride])** αποθηκεύεται σε μια προσωρινή μεταβλητή a. Στην συνέχεια για το pixel στην θέση j, υπολογίζεται η κάθετη παρεμβολή του pixel που βρίσκεται ανάμεσαι στο j+1 της i γραμμής και j+1 της (i+1) γραμμής, το αποτέλεσμα αποθηκεύται σε μια προσωρινή μεταβλητή b, χωρίς να γίνει ολίσθηση κατά 8. Στην συνέχεια το αποτέλεσμα αυτό πολλαπλασιάζεται με την διαφορά του υπολοίπου της οριζόντιας συνιστώσας με το 8 και αποθηκεύεται σε μια προσωρινή

μεταβλητή c. Τέλος υπολογίζεται η τιμή του τρέχωντος pixel .Η τιμή του a για όλες τις αλλές θέσεις διαφορες του x υπολογίζεται απο τον τύπο (a=b<<3+c) .

Η διαδικασία που πραγματοποιείται είναι η ακόλουθη. Αρχικά φορτώνουμε στους καταχωρητές xmm0 xmm1 τα pixels της i και i+1 γραμμής που απαιτούνται. Ολισθαινουν τα περιεχόμενα των καταχωρητών ετσί ώστε j να βρεθούν τα περιεχόμενα της j+1. Για να γίνει αυτό για πλάτος 2 αρκεί να ολισθήσουμε τους καταχωρητές xmm0 και xmm1 κατά 2. Για μήκος 4 αρκει να φορτώσουμε στους καταχωρητές xmm0 και xmm1 8 pixels αντί για 4 .Στην συνέχεια τους ολισθαίνουμε δεξια κατά 8 pixel και το αποτέλεσμα που προκύπτει κατά 14 θέσεις . Τα αρχικά περιεχόμενα ολισθαίνουν κατά 2 θέσεις δεξιά και τα νέα αποτελέσμα συγχωνεύονται με χρήση της εντολής por. Για ευκολία οι νέοι καταχωρητές θα ονομαστούν xmm2 και xmm3. η τιμή του a θα υπολογιστεί και για τις 2 περιπτώσεις,ταυτόχρονα και θα γίνει συγχώνευση στο τέλος. Αρχικά υπολογίζουμε για την πρώτη περίπτωση που δίνεται από τον τύπο τύπο **xF0* (yF0 * pucSrc[0] + yF1 * pucSrc[iSrcStride]).** Η αριθμητική παράσταση μέσα στις παρενθέσεις έχει περιγραφτεί σε προηγούμενη παράγραφο. Στην συνέχεια πολλαπλασιάζεται με ένα καταχωρητή, στον οποίο φορτώνεται η τιμή xF0 με χρήση της εντολής modqa. Ο πολλαπλασιασμός γίνεται με χρήση της εντολής pmullw και οι καταχωρητές που χρησιμοποιούνται είναι οι xmm0 και xmm1. Έπειτα μηδενίζονται όλες οι θέσεις του καταχωρητή εκτος του 0 με ολίσθηση αρχικα αριστερα κατα 14, εντολη pslldq και στην συνέχεια με χρήση της εντολής psrldq δεξιά ολίσθηση κατα 14 θέσεις. Στην συνέχεια υπολογίζουμε την τιμή του a για όλες τις άλλες περιπτώσεις. Πολλαπλασιάζουμε τον καταχωρητή b με 8 χρησιμοποιώντας την εντολή psllw και αφαιρείται χρησιμοποιώντας την εντολή psubw ο καταχωρητής c . Στην συνέχεια τα 2 αποτελέσματα συγχωνεύονται με χρήση της εντολής por. Για το υπολογισμό της τιμής του καταχωρητή b χρησιμοποιούνται οι ολισθημένες τιμές των καταχωρητών xmm0 και xmm1. Ο τύπος που υπολογίζει το b δίνεται από **(yF0 * pucSrc[0] + yF1 * pucSrc[iSrcStride])** και ήδη έχει περιγραφεί. Τέλος έχωντας το b και το c υπολογίζουμε το a. Η διαδικασία αυτή επιτρέπει να δημιουργούνται τα κατάλληλα περιεχόμενα στον καταχωρητή a, έτσι ώστε η τιμή του a της προηγούμενης επανάληψης να χρησιμοποιείται στην επόμενη. Τελειώνοντας την διαδικασία, γράφουμε τα κατάλληλα pixels στο τρέχων block.

Πριν την βελτιστοποίηση ο αρχικός κώδικας δαπανούσε συνολικά **856800000** κύκλοι μηχανής**.** Μετά τις βελτιστοποιήσεις οι συνολικοί κύκλοι μηχανής μειώνονται σε **573300000** κύκλους. Συνολικά επιτυγχάνεται βελτιστοποίηση 1.49 x, που αντιστοιχεί σε ποσοστό 32 %.

4.4.2 Σύνοψη αποτελεσμάτων για φωτεινότητα και χρώμα

Στις προηγούμενες ενότητες περιγράφτηκαν οι αλγόριθμοι του motion compensation και τα βήματα που ακολουθήθηκαν για την βελτιστοποίηση τους. Στο παρακάτω πίνακα συγκεντρώνονται τα αποτελέσματα για την βελτιστοποίηση του motion compensation για τα pixels της φωτεινότητας και του χρώματος.

Motion Compensation	**Μη Βελτιστοποιημένη**	**Βελτιστοποιημένη**	**Επιτάχυνση**
Luma	**3244500000**	**1148700000**	**2.82**
Chroma	**856800000**	**573300000**	**1.49**
Άθροισμα	**4101300000**	**1722000000**	**2.38**

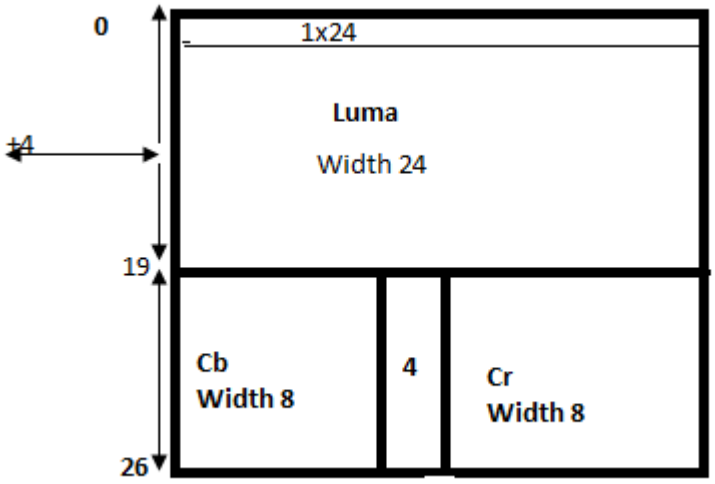
Από τον παραπάνω πίνακα διακρίνεται ότι η συνολική επιτάχυνση των αλγορίθμων για τον υπολογισμό των pixels από τα καρέ αναφοράς είναι 2.38x. Στα παραπάνω πρέπει να προστεθεί και η τελική πρόσθεση που λαμβάνει χώρα ανάμεσα στα υπολογισμένα pixels και στα υπόλοιπα που έχουν υπολογιστεί κατά την διαδικασία κωδικοποίησης και έχουν αποκωδικοποιηθεί.

4.5 Συνάρτηση πρόσθεσης υπολοίπων

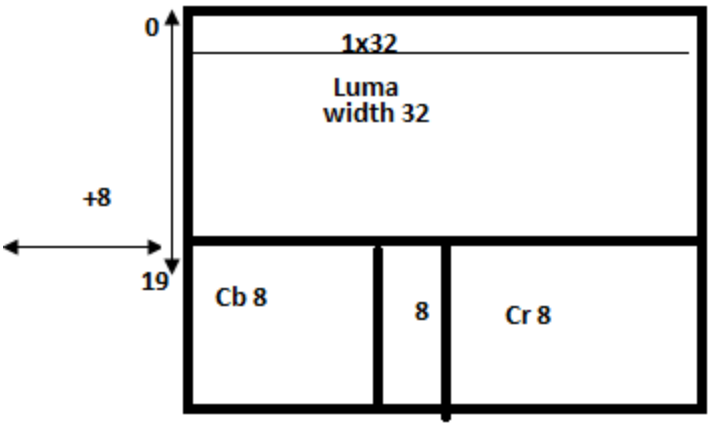
Σε αυτή την ενότητα θα περιγραφεί η διαδικασία που ακολουθήθηκε για να γίνει η τελική φάση του motion compensation με διανυσματικές πράξεις. Όπως ήδη αναφέρθηκε η τελική φάση είναι μια πρόσθεση των τιμών που υπολογίστηκαν από το motion compensation και των υπολοίπων που υπολογίστηκαν στην διαδικασία της κωδικοποίησης, το οποίο συμβαίνει μόνο για το τελευταίο ιεραρχικά επίπεδο. Πριν την διαδικασία αυτή προστίθενται οι τιμές των υπολοίπων που έχουν προκύψει από την διαδικασία του upsampling του inter layer prediction και τα υπόλοιπα που προκύπτουν από την διαδικασία του inverse transform για το τρέχων

layer. Επιπλέον η συνάρτηση αυτή χρησιμοποιείται για την δημιουργία των υπολοίπων που θα χρησιμοποιηθούν για το upsampling των υπολοίπων του επόμενου επιπέδου του επόμενου επιπέδου

Η πρώτη αλλαγή η οποία έγινε, αφορούσε την αλλαγή του μεγέθους της τοπικής αποθήκης των pixels για luma και χρώμα έτσι ώστε να είναι ευθυγραμμισμένα. Η τοπική αποθήκη, περιέχει ένα επεκταμένο macroblock μεγέθους 24x19 καθώς επίσης και ένα block για την συνιστώσα Cb και μια συνιστώσα Cr μεγέθους 8x16.Το παρακάτω σχήμα δείχνει την αποθήκη .



Για να προσπελαστεί ένα στοιχείο πχ το Cb πρέπει να πολλαπλασιαστεί το 19*24 και στην συνέχεια να προστεθεί το 4, το αποτέλεσμα δίνει 460 το οποίο δεν είναι πολλαπλάσιο του 16. Για το λόγο αυτό αυξάνεται το μήκος κάθε γραμμής της αποθήκης σε 32. Αυξάνοντας το μήκος σε 32, πλέον για να προσπελαστεί το Luma πρέπει να προσθέσουμε 32+8, για το Cb 19*32+8,για το Cr 32+19*32+24. Πλέον όλες οι θέσεις μνήμης είναι ευθυγραμμισμένες. Η νέα αποθήκη φαίνεται στην παρακάτω εικόνα.



Εφόσον έχει ευθυγραμμιστεί η τοπική αποθήκη, στην συνέχεια μπορούν να χρησιμοποιηθούν διανυσματικές εντολές . Χρησιμοποιώντας τις εντολές movdqa , φορτώνονται τα υπόλοιπα από τα αντίστοιχα layers . Στην συνέχεια προστίθενται τα περιεχόμενα των δύο καταχωρητών χρησιμοποιώντας την εντολή paddw. Το αποτέλεσμα της πρόσθεσης πρέπει να περιοριστεί από -255 έως 255. Για να συμβεί αυτό πρέπει να φορτωθούν σε 2 καταχωρητές οι τιμές 32512 και -32513, που αποτελούν την αφαίρεση του 255 από το 32767(μέγιστη τιμή για signed short),και του -255 από το -32768(ελάχιστη τιμή για signed short). Στην συνέχεια προστίθενται τα περιεχόμενα της πρόσθεσης με το καταχωρητή που περιέχει την τιμή 32512,χρησιμοποιώντας την εντολή paddsw. Από το αποτέλεσμα της πρόσθεσης αφαιρείται ο καταχωρητής που έχει περιεχόμενο την τιμή 32512,χρησιμοποιώντας την εντολή psubusw. Χρησιμοποιώντας αυτές τις πράξεις περιορίζονται τα περιεχόμενα των καταχωρητών μέχρι 255.Το παρακάτω παράδειγμα απεικονίζει αυτά που περιγράφτηκαν παραπάνω.

256	127	234	345	125	258	89	90
+	+	+	+	+	+	+	+
32512	32512	32512	32512	32512	+32512	32512	32512
32768	32639	32746	32857	32637	32770	32601	32602
saturate	saturate	saturate	saturate	saturate	saturate	Saturate	saturate
32767	32639	32746	32767	32637	32767	323601	32602
-	-	-	-	-	-	-	-
255	127	234	255	125	255	89	90

Συνεχίζοντας θα πρέπει να περιοριστεί το εύρος μέχρι το -255. Αρχικά στο τελικό αποτέλεσμα των παραπάνω πράξεων προστίθεται ο καταχωρητής που περιέχει την τιμή -32513,με χρήση της εντολής paddsw. Το αποτέλεσμα της πρόσθεσης αφαιρεί το καταχωρητή που περιέχει την τιμή psubsw. Χρησιμοποιώντας αυτές τις 2 πράξεις το αποτέλεσμα περιορίζεται με κάτω όριο -255 και πάνω όριο 255.

Συνοψίζοντας, η συνάρτηση addRes πριν την χρήση διανυσματικών πράξεων χρειαζόταν για την εκτέλεση της **1860600000** κύκλους μηχανής . Μετά την βελτιστοποίηση οι συνολικοί χρόνοι μειώνονται σε **178500000,** δηλαδή η συνάρτηση είναι 10.48 φορές πιο γρήγορη και σε ποσοστό 90. %.

Παρακάτω, στο τελευταίο επίπεδο εκτελείται η πρόσθεση των pixels που έχουν παραχθεί από το motion compensation και των υπολοίπων . Η συνάρτηση αυτή ονομάζεται addClip. Για την υλοποίηση της με διανυσματικές εντολές πρέπει αρχικά να φορτωθούν σε δυο καταχωρητές με χρήση των εντολών movdqa τα υπόλοιπα και τα pixels του τρέχοντος layer. Κατόπιν, προστίθενται με χρήση της εντολής paddw . Στο τέλος γίνεται περιορισμός του εύρους από 0-255 με χρήση της εντολής packuswb .Η αρχική έκδοση της συνάρτησης δαπανούσε **816900000** κύκλους μηχανής **.** Με χρήση διανυσματικών πράξεων χρειάζεται μόνο **115500000** κύκλους μηχανής, δηλαδή επιτάχυνση 7.6 x ,που αντιστοιχεί σε ποσοστό 86 %.

4.6 Συνολική Βελτιστοποίηση του motion compensation

Με την περιγραφή και της συνάρτησης πρόσθεσης των υπολοίπων ολοκληρώνεται η διαδικασία του motion compensation. Η συνολική βελτιστοποίηση που επιτεύχθηκε για όλη την διαδικασία είναι 2.38 x για την διαδικασία του motion compensation και 10.42 x για την πρόσθεση των υπολοίπων των 2 layers και 7.6x για την τελική πρόθεση. Άρα συνολικά από τους **6759900000** κύκλους μηχανής, πλέον χρειάζονται μόνο **2013900000,** δηλαδή η διαδικασία του motion compensation βελτιστοποιήθηκε κατά 3.35x,σε ποσοστό 70.3 %.

Ο αναγνώστης θα διαπιστώσει ότι η βελτιστοποίηση του motion compensation δε είναι η καλύτερη δυνατή. Το πρόβλημα διαπιστώνεται στο γεγονός ότι πρέπει να αντιγράφονται

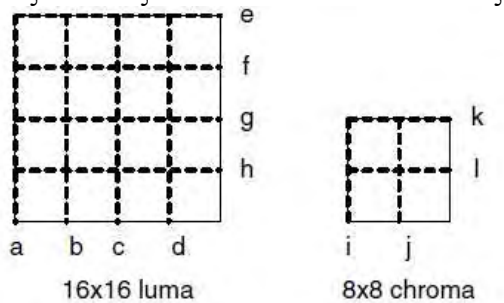
δεδομένα σε τοπική μνήμη και να μεταφέρονται από την τοπική μνήμη στο τρέχων block. Όμως η αλληλεπίδραση με την μνήμη είναι πολύ αργή σε σχέση με τους υπολογισμούς με SSE2 εντολές.

4.7 Deblocking Filter

Στην ενότητα αυτή θα περιγραφή η βελτιστοποίηση του deblocking filter. Το deblocking filter είναι ένα φίλτρο το οποίο εφαρμόζεται μετά τη διαδικασία του inverse transform. Το φίλτρο αυτό εξομαλύνει τις ακμές ενός block ,βελτιώνοντας έτσι την απεικόνιση του αποκωδικοποιημένου frame. Το φίλτρο εφαρμόζεται είτε στις οριζόντιες είτε στις κάθετες ακμές ενός 4x4 block με την ακόλουθη σειρά.

1. Φιλτράρει 4 κάθετα όρια για τα blocks που περιέχουν pixels που αντιστοιχούν στην φωτεινότητα.
2. Φιλτράρει 4 οριζόντια όρια για τα blocks που περιέχουν pixels που αντιστοιχούν στην απεικόνιση της φωτεινότητας.
3. Φιλτράρει τα 2 κάθετα όρια για τα blocks που περιέχουν pixels που αντιστοιχούν στο chroma.
4. Φιλτράρει τα 2 οριζόντια όρια για τα blocks που περιέχουν pixels που αντιστοιχούν στο chroma.

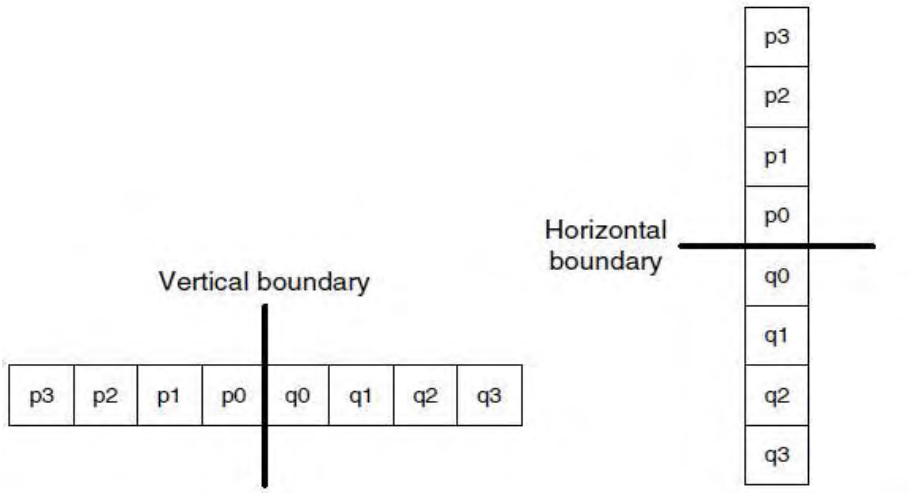
Στις εικόνες που ακολουθούν απεικονίζονται τα παραπάνω.



Edge filtering order in a macroblock

Για τα pixels της φωτεινότητας επηρεάζονται μέχρι 3 pixels σε κάθε πλευρά του ορίου. Η παρακάτω εικόνα απεικονίζει 4 pixels από κάθε πλευρά για οριζόντια και κάθετε όρια για τα

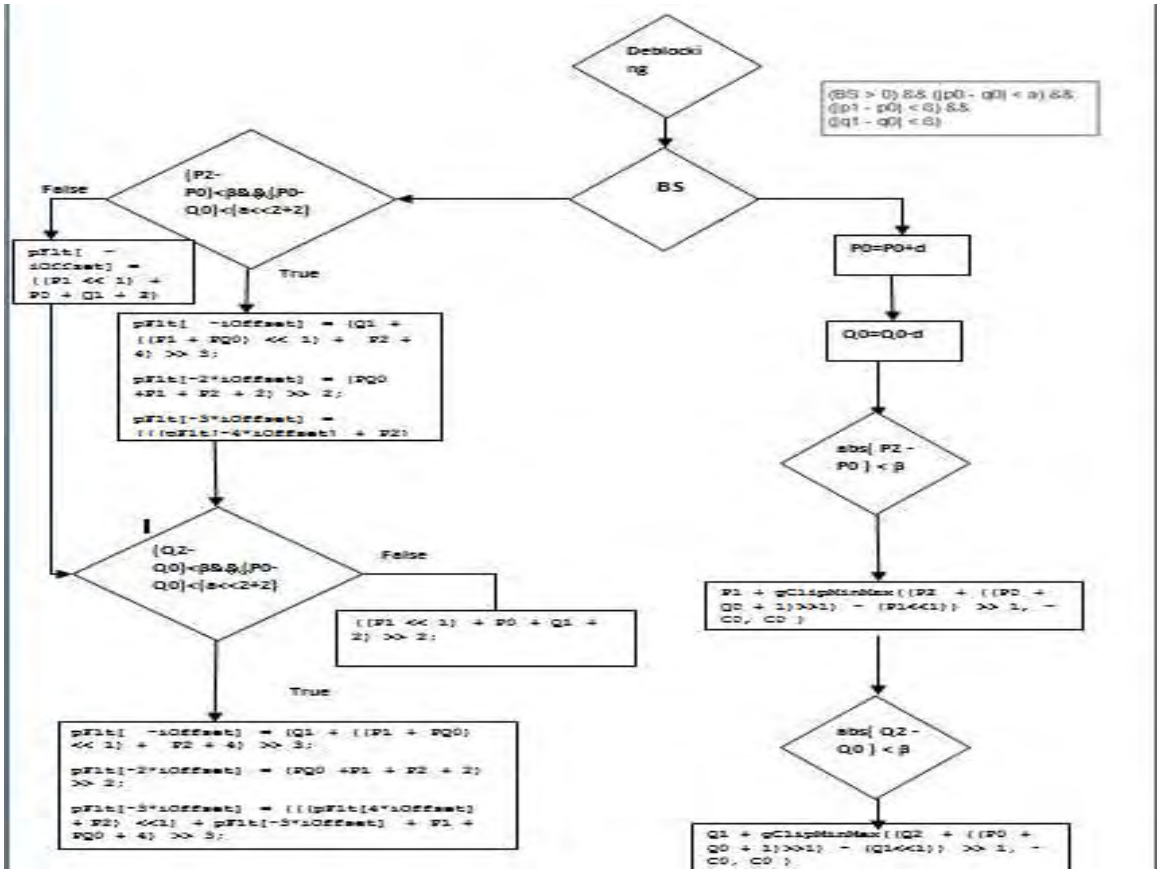
blocks p και q (p0,p1,p2,p3 q0,q1,q2,q3). Ο αριθμός των pixels που θα επηρεαστούν εξαρτάται τρέχουσα κβαντικοποίηση, τους τύπους κωδικοποίησης των γειτονικών macroblock και τις διαφορές της εικόνας στα όρια. Η παρακάτω εικόνα απεικονίζει τα κάθετα και οριζόντια όρια.



Samples adjacent to vertical and horizontal boundaries

Ένα σύνολο από τιμές {p2,p1,p0,q0,q1,q2} φιλτράρεται μόνο αν το BS(ποσότητα που δείχνει το πλήθος των pixels που πρέπει να αλλαχθούν) είναι μεγαλύτερο του 0. Επιπλέον θα πρέπει |p0-q0|α και |p1-p0|<β and |q1-q0|<β. Τα α και β είναι κατώφλια τα οποία καθορίζονται από το πρότυπο και αυξάνονται με την αύξηση του παράγοντα κβαντικοποίησης.

Η πρώτη αλλαγή που πρέπει να λάβει χώρα για να μπορέσει το κομμάτι αυτό να παραλληλοποιηθεί με χρήση διανυσματικών εντολών, είναι η δημιουργία 2 νέων συναρτήσεων για το luma(κάθετο και οριζόντιο) και 2 νέων για το chroma (κάθετο και οριζόντιο). Εφόσον έχουν δημιουργηθεί οι δύο νέες συναρτήσεις πλέον μπορεί να βελτιστοποιηθεί η συνάρτηση με χρήση διανυσματικών εντολών. Η μέθοδος που χρησιμοποιείται για την παραλληλοποίηση ονομάζεται software prediction. Σύμφωνα με την μέθοδο αυτή υπολογίζονται οι τιμές των pixels για όλες τις συνθήκες (δυνατά μονοπάτια) και στο τέλος επιλέγεται η κατάλληλη τιμή. Στο παρακάτω σχήμα παρουσιάζονται όλα τα δυνατά μονοπάτια για τις τιμές της φωτεινότητας.



4.7.1 Οριζόντιο filtering

Στην ενότητα αυτή θα περιγραφεί η διαδικασία που ακολουθήθηκε για την παραλληλοποίηση της διαδικασίας του οριζόντιου filtering. Στην εισαγωγή της ενότητας 4.7 αναφέρθηκε ότι η διαδικασία του φιλτραρίσματος λαμβάνει χώρα σε block διαστάσεων 4x4. Όμως κάτω από συγκεκριμένες προϋποθέσεις που θα περιγραφτούν σε επόμενο κεφάλαιο, το BS δε αλλάζει ανάμεσα σε 2 blocks 4x4. Αυτό δίνει την δυνατότητα παραλληλοποίηση της διαδικασίας ,φιλτράροντας ταυτόχρονα 2 blocks μαζί. Για να γίνει αυτό αρκεί πριν την κλήση της συνάρτησης ,ένας έλεγχος αν το BS δύο διαδοχικών block είναι το ίδιο.

4.7.1.1 Luma Filtering

Για την διαδικασία του οριζόντιου Luma filtering θα πρέπει να φορτωθούν σε 4 καταχωρητές όλα τα pixels όλων των απαραίτητων γραμμών που πρέπει να χρησιμοποιηθούν στην διαδικασία του φιλτραρίσματος. Σε αυτή την φάση δε γίνεται έλεγχος αν η τιμή της

μεταβλητής που προσδιορίζει αν δυο blocks έχουν την ίδια τιμή .Απλά φορτώνονται σε κάθε καταχωρητή 8 τιμές μήκους 16 bits .Οι καταχωρητές που χρησιμοποιούνται είναι οι P0 ,P1,P2,P3,Q0,Q1,Q2,Q3. Στην συνέχεια φορτώνεται η τιμή α σε ένα καταχωρητή, με την ονομασία iAlpha. Η τιμή αυτή προσδιορίζεται από ένα πίνακα τύπου AlphaClip . Η θέση του πίνακα που θα προσπελαστεί ορίζεται απο το indexA. Το indexA είναι η τιμή που επιστρέφεται από ένα πίνακα, στην θέση που προκύπτει από την πρόσθεση της τιμής α και του συντελεστή κβαντικοποιήσης και της τιμής 6, διότι η ελάχιστή τιμή που μπορεί να έχει αυτό το άθροισμα είναι -6 και η μεγιστη 58. Με το ίδιο τρόπο προσδιορίζεται η τιμή iBeta . Προκύπτει απο τον ίδιο πίνακα αλλά η θέση της προσδιορίζεται απο το άθροισμα της τιμής β και του συντελεστή κβαντικοποιήσης .Στην συνέχεια χρησιμοποιώντας την εντολή psubw υπολογίζεται η διαφορά Q0-P0. Το αποτέλεσμα αποθηκεύεται στον καταχωρητή iDelta. Στην συνέχεια υπολογίζεται η απόλυτη τιμής της διαφοράς τους και αποθηκεύεται στον καταχωρητή AbsDelta. Η διαδιασία που ακολουθείται είναι αρχικα η αφαίρεση του P0-Q0 με χρήση της εντολής psubusw και του Q0-P0 με χρήση της ίδιας εντολής και συγχώνευση των αποτελεσμάτων με την εντολή por. Στην συνέχεια πρέπει να υπολογιστούν οι συνθήκες που επιτρέπουν την εκτέλεση η μη εκτέλεση της διαδικασίας του deblocking. Χρησιμοποιώντας την εντολή pcmpltw, ελέγχεται αν τα περιεχόμενα του καταχωρητή iAbsDelta είναι μικρότερα από τα αντίστοιχα του Alpha. Τα αποτελέσματα των παραπάνω πράξεις αποθηκεύονται σε ένα προσωρινό καταχωρητή [1]. Στην συνέχεια πρέπει να ελεγχθεί αν η απόλυτη διαφορά του P1-P0 και Q1-Q0 είναι μικρότερη από την iBeta. Ο τρόπος που υπολογίζεται η απόλυτη διαφορά των τιμών των καταχωρητών P1, P0 Q1,Q0 είναι ίδιος με αυτόν υπολογισμού του AbsDelta. Αφού υπολογιστούν οι απόλυτες διαφορές τα αποτελέσματα οδηγούνται σαν είσοδο στην εντολή pcmpltw. Τα 2 νεά αποτελέσματα οδηγούνται σε μια pand και το αποτέλεσμα αποθηκεύεται στον προσωρινό καταχωρητή [2] . Τα αποτελέσματα των 2 καταχωρητών [1] και [2] οδηγούνται σε μια pand και το αποτέλεσμα αποθηκεύεται στον καταχωρητή [1]. Στην συνέχεια ελέγχεται αν τα 2 blocks έχουν το ίδιο BS .Αν δε έχουν το ίδιο BS η διαδικασία διαφέρει σε κάποια σημεία. Αρχικά ελέγχεται η τιμή του BS . Αν το BS είναι μικρότερο της τιμής 4 τότε η διαδικασία που ακολουθείται είναι η ακόλουθη. Υπολογίζεται η τιμή των περιεχόμενων του καταχωρητή P1 υποθέτοντας ότι η συνθήκη είναι αληθής . Ο τύπος που υπολογίζει την νέα τιμή του P1 είναι $\mathbf{P1 + gClipMinMax((P2 + ((P0 + Q0 + 1) >> 1) - (P1 << 1)) >> 1, -C0, C0)}$.
Οι πράξεις αυτές αντικαθιστούνται από τις αντίστοιχες διανυσματικές. Για την διαδικασία

περιορισμού του εύρους ανάμεσα στις τιμές [–C0,C0], είναι παρόμοια με αυτή στην ενότητα 4.5 . Σε αυτήν την περίπτωση δε είναι γνωστό το άνω και κάτω όριο. Για το λόγο αυτό πρέπει να υπολογιστεί το άνω και κάτω όριο με το οποίο θα γίνουν οι πράξεις αποκοπής. Η διαδικασία που ακολουθείται είναι απλή. Ορίζονται 2 πίνακες ένας περιέχει την μέγιστη τιμή για προσημασμένους αριθμούς δηλαδή την τιμή 0x7FFF και αντιστοιχα την τιμή -0x8000 που αποτελεί την ελάχιστη για αυτή την κατηγορία αριθμών. Στη συνέχεια έχωντας φορτώσει αυτές τις τιμές χρησιμοποιώντας την εντολής movdqa, υπολογίζονται το άνω και κάτω όριο. Το πάνω όριο υπολογίζεται αφαιρώντας απο τον καταχωρητή που περιέχει την μέγιστη τιμή αυτό που περιέχει την τιμή C0. Το κάτω όριο υπολογίζεται αφαιρώντας απο τον καταχωρητή που περιέχει την ελάχιστη τιμή, τον καταχωρητή που περιέχει την τιμή –C0. Έχωντας υπολογίσει την άνω και κάτω τιμή, χρησιμοποιούνται οι κατάλληλες πράξεις για τον περιορισμό του εύρους. Στην συνέχεια υπολογίζεται νέα τιμή του καταχωρητή Q1. Η σχέση που δίνει την νέα τιμή του Q1 στον αρχικό κώδικα είναι **Q1 + gClipMinMax((Q2 + ((P0 + Q0 + 1)>>1) - (Q1<<1)) >> 1, -C0, C0).** Οι πράξεις αυτές αντικαθίστανται απο τις αντίστοιχες διανυσματικές. Ο περιορισμός του εύρους των τιμών έχει ήδη αναφερθεί. Συνεχίζοντας στα περιεχόμενα του C0 προσθέτουμε τις τιμές ενός νέου καταχωρητή. Για τα περιεχόμενα του νέου καταχωρητή αρχικα προσθέτουμε δυο προσωρινούς καταχωρητές που έχουν είτε ως περιεχόμενα την τιμή 1 η -1. Έχοντας υπολογίσει τις τιμές των καταχωρητών ap και aq που δίνονται απο τους ακόλουθους τύπους **aq = ((abs(Q2 - Q0) < iBeta)** και **ap = ((abs(P2 - P0) < iBeta)** . Οι παραπάνω τύποι αντικαθιστώνται απο τις αντίστοιχες διανυσματικές εντολές . Τα τελικά περιεχόμενα των καταχωρητών aq και ap θα είναι είτε 0 είτε 65536. Άρα ολισθαίνοντας τα περιεχόμενα κατά 15 θέσεις δεξία χρησιμοποιώντας την εντολή psrlw, προκύπτουν δυο προσωρινοί καταχωρητές με τιμές 1 ή -1 . Στην συνέχεια το αποτέλεσμα της πρόσθεσης οδηγείται σε μια paddw και προστίθεται με ένα καταχωρητή στον οποίο έχει φορτωθεί η τιμή -1. Τέλος το τελικό αποτέλεσμα αποθηκεύεται σε προσωρινό καταχωρητή ο οποίος προστίθεται με τον καταχωρητή C0. Ο καταχωρητής C0 που περιέχει τα νέα περιεχόμενα προστίθεται με το καταχωρητή που περιέχει την τιμή 1. Στην συνέχεια ορίζεται ένας νέος καταχωρητής οι τιμές του οποίου δίνονται από τον τύπο **gClipMinMax(((iDelta << 2) + (P1 - Q1) + 4) >> 3, -C0, C0).** Ο καταχωρητής iDelta έχει ήδη υπολογιστεί . Αντικαθίστωνται όλες οι πράξεις απο τις αντίστοιχες διανυσματικές . Συνεχίζοντας πρέπει να υπολογιστούν οι νέες τιμές P0 και Q0. Στον αρχικό κώδικα οι νέες τιμές δίνονται απο τον τύπο **gClip(P0 + iDiff)** και **gClip(Q0 - iDiff)**. Οι

αριθμητικές πράξεις αντικαθίστανται απο τις αντίστοιχες διανυσματικές και η gClip απο τις αντίστοιχες πράξεις περιορισμού του εύρους απο 0 -255 για τις οποίες ήδη έχει δοθεί περιγραφή. Εύκολα γίνεται η διαπίστωση ότι για BS μικρότερο του 4 αλλάζουν 2 pixels. Στο τέλος της διαδικασίας πρέπει να γίνει η επιλογή των κατάλληλων τιμών για τα pixel που πρέπει εν δυνάμει να αλλάξουν τιμή. Για την επιλογή των κατάλληλων περιεχομένων απο τους καταχωρητές χρησιμοποιείται η εντολή pblendvb. Η εντολή αυτή δέχεται ως όρισμα δυο καταχωρητές και ένα καταχωρητή που αποτελεί μια μάσκα για να γίνει η σωστή επιλογή των καταχωρητών. Ο τρόπος που λειτουργεί η εντολή ορίζεται σύμφωνα με τα παρακάτω

r0 := (mask0 & 0x80) ? b0 : a0

r1 := (mask1 & 0x80) ? b1 : a1

...

r15 := (mask15 & 0x80) ? b15 : a15

Δηλαδή για κάθε τιμή της μάσκας ελέγχεται αν το λογικό and με το 128 επιστρέφει 0 ή 1. Αν επιστρέφει 1 επιλέγεται ο δεύτερος καταχωρητής αλλιώς ο πρώτος καταχωρητής. Επειδή τα δεδομένα που χρησιμοποιούνται θα πρέπει να είναι τύπου 8x16 θα πρέπει να χρησιμοποιηθεί η εντολή packsswb, για όλα τα δεδομένα που χρησιμοποιούνται από την pblendvb. Για τον υπολογισμό των τιμών που πρέπει να αποθηκευτούν P1, αρχικά πρέπει να υπολογιστεί η μάσκα .Χρησιμοποιείται η εντολή pand για τους καταχωρητές [1] και ap και το αποτέλεσμα δημιουργεί την κατάλληλη μάσκα .Στην συνέχεια σαν πρώτο όρισμα στην εντολή pblendvb, δηλώνεται ο καταχωρητής P1 και σαν δεύτερο όρισμα ο καταχωρητής που περιέχει τα αποτελέσματα αν ίσχυε η συνθήκη. Στην συνέχεια τα δεδομένα πακετάρονται σε 8x16 χρησιμοποιώντας την εντολή punpcklwd. Τέλος τα δεδομένα αποθηκεύονται πίσω στην μνήμη χρησιμοποιώντας την εντολή movdqa. Για τον καταχωρητή Q1 η διαδικασία που ακολουθείται είναι ίδια με την παραπάνω, αλλά χρησιμοποιείται η εντολή pand με είσοδο τους καταχωρητές aq και [2]. Τέλος χρησιμοποιείται ο καταχωρητής [1] σαν μάσκα για τον υπολογισμό των P0 και Q0. Τα αποτελέσματα αποθηκεύονται χρησιμοποιώντας την εντολή movdqa.

Στην δεύτερη περίπτωση το BS είναι ίσο με 4. Στην περίπτωση αυτή τα pixels που ελέγχονται είναι 4 και τιμή αλλάζουν τα 3. Εκτός από τις αρχικές συνθήκες που είναι ίδιες ορίζονται 2 νέοι καταχωρητές aqBs4 και apBs4, που χρειάζονται στην συνέχεια για την επιλογή των σωστών τιμών που πρέπει να γραφτούν πίσω στην μνήμη. Κάθε περιεχόμενο του

καταχωρητή aqBS4 δίνεται από τον μαθηματικό τύπο $\mathbf{bEnable} \,\&\, (\,\mathbf{abs}(\,\mathbf{Q2} - \mathbf{Q0}\,) < \mathbf{iBeta}\,)$ και $\mathbf{bEnable} \,\&\, (\,\mathbf{abs}(\,\mathbf{P2} - \mathbf{P0}\,) < \mathbf{iBeta}\,)$. Οι λογικές και αριθμητικές πράξεις αντικαθιστώνται από τις διανυσματικές πράξεις . Για την τιμή bEnable ορίζεται ένας νέος καταχωρητής τα περιεχόμενα του οποίου δίνονται από το μαθηματικό τύπο $\mathbf{iAbsDelta} < ((\mathbf{iAlpha} >> 2) + 2$. Ομοίως οι λογικές και αριθμητικές πράξεις αντικαθίστανται από τις αντίστοιχες διανυσματικές .Αγνοώντας την ύπαρξη των αρχικών συνθηκών καθώς και των τιμών των καταχωρητών aqBs4 και apBs4 υπολογίζουμε όλες τις τιμές και για τους 2 βρόγχους if – else. Για τις τιμές του καταχωρητή Q0, υπάρχουν 3 περιπτώσεις .

- Οι τιμές του καταχωρητή να μείνουν ίδιες .Αν οι αρχικές συνθήκες δε ικανοποιούνται
- Αν aq =1, για κάθε περιεχόμενο του καταχωρητή η μαθηματική σχέση που δίνει την νέα τιμή είναι $\mathbf{P1} + ((\mathbf{Q1} + \mathbf{PQ0}) << 1) + \mathbf{Q2} + \mathbf{4}) >> 3.$
- Αν aq =0 ,για κάθε περιεχόμενο του καταχωρητή η μαθηματική σχέση που δίνει την νέα τιμή είναι $\big((\mathbf{Q1} << 1) + \mathbf{Q0} + \mathbf{P1} + \mathbf{2}\big) >> 2$

Για τον καταχωρητή P0 ομοίως υπάρχουν 3 περιπτώσεις

- Οι τιμές του καταχωρητή να μείνουν οι ίδιες . Αν οι αρχικές συνθήκες δε ικανοποιούνται
- Αν ap=1 για κάθε περιεχόμενο του καταχωρητή η μαθηματική σχέση που δίνει την νέα τιμή για τον καταχωρητή P0 είναι $\big(\mathbf{Q1} + ((\mathbf{P1} + \mathbf{PQ0}) << 1) + \mathbf{P2} + 4\big) >> 3$
- Αν ap =0 για κάθε περιεχόμενο του καταχωρητή η μαθηματική σχέση που δίνει την νέα τιμή για τον καταχωρητή P0 είναι $\big((\mathbf{Q1} << 1) + \mathbf{Q0} + \mathbf{P1} + 2\big) >> 2$. Για τις υπόλοιπες τιμές αρκεί να ισχύει η αρχική συνθήκη και μια εκ των ap==1 ή aq==1 . Όπως και για την περίπτωση που το BS είναι μικρότερο του 4 χρησιμοποιείται η εντολή pblendvb στο τέλος της διαδικασίας .Όμως τα δεδομένα που δέχεται ως είσοδο είναι η εντολή είναι οργανωμένα σε 16x8 άρα θα πρέπει όλοι οι καταχωρητές που θα χρησιμοποιηθούν να πακεταριστούν σε 16x8 χρησιμοποιώντας την εντολή punpcklwd. Για τον καταχωρητή Q0 και P0 πρέπει να χρησιμοποιηθούν 2 εντολές pblendvb. Συγκεκριμένα για τον καταχωρητή Q0, αρχικά έχουν υπολογιστεί και τιμές και για τις 2 περιπτώσεις .Δηλαδή για aq =0 ή aq=1. Τα αποτελέσματα έχουν αποθηκευτεί σε προσωρινούς καταχωρητές [3] [4] . Ο καταχωρητής aqBs4 περιέχει τις τιμές των aq. Επομένως ο καταχωρητής αυτός χρησιμοποιείται σαν μάσκα είναι ο aqBs4. Πρώτο όρισμα ο καταχωρητής [2] που περιέχει τις τιμές για aq =0 και

δεύτερο όρισμα ο καταχωρητής 2. Το αποτέλεσμα αποθηκεύεται στον προσωρινό καταχωρητή [3]. Στην συνέχεια χρησιμοποιείται η δεύτερη εντολή pblendvb που έχει σαν τρίτο όρισμα τον καταχωρητή [1]. Πρώτο όρισμα είναι ο καταχωρητής Q0 και δεύτερο όρισμα ο καταχωρητής [3]. Η διαδικασία τελειώνει γράφοντας τις τελικές τιμές πίσω στην μνήμη με χρήση της εντολής movdqa. Για τους καταχωρητές Q1 και Q2 για την επιλογή της τελικής τιμής χρησιμοποιείται η εντολή pblendvb χρησιμοποιώντας για μάσκα τον καταχωρητή [1] που έχει το αποτέλεσμα της πράξης της εντολής pand για τους καταχωρητές [1] και aqBs4. Τα τελικά δεδομένα γράφονται πίσω στην μνήμη χρησιμοποιώντας την εντολή movdqa. Η διαδικασία είναι ίδια για τους καταχωρητές P0,P1,P2 σε αντιστοίχηση με τους Q0,Q1,Q2,Q3 . Η μόνη διαφορά είναι ότι χρησιμοποιείται ο καταχωρητής apBs4 αντί για τον καταχωρητή aqBs4.

Αν 2 4x4 blocks έχουν διαφορετικό BS τότε το μόνο που διαφέρει είναι ο τρόπος αποθήκευσης. Έχοντας φορτώσει ήδη τα δεδομένα 16x8 bits ,είτε τα πρώτα 4 είτε τα τελευταία 4 pixels τα κουβαλάμε σε όλη την διάρκεια της διαδικασίας .Τα πρώτα 4 pixels αντιστοιχούν στο πρώτο 4x4 block ,άρα για να γραφτούν τα σωστά δεδομένα στην μνήμη αρκεί να γράψουμε τις κατάλληλες τιμές χρησιμοποιώντας την εντολή movq. Για τα pixels στις υψηλότερες θέσεις πρέπει αυτά να μεταφερθούν στις χαμηλότερες 4 χρησιμοποιώντας την εντολή psrldq (ολίσθηση κατά 8 θέσεις) και να γίνει εγγραφή χρησιμοποιώντας την εντολή movq.

Συνοψίζοντας η αρχική έκδοση της συνάρτησης που εκτελεί την παραπάνω διαδικασία χρειαζόταν για ολοκληρωθεί επιτυχώς **842100000** κύκλους μηχανής**.** Μετά την χρήση διανυσματικών εντολών οι συνολικοί κύκλοι μηχανής που απαιτούνται είναι **462000000.** Άρα η συνολική βελτιστοποίηση είναι 1.82x που αντιστοιχεί σε ποσοστό 46 %

4.7.1.2 Chroma Filtering

Για το οριζόντιο φιλτράρισμα των pixels που αντιστοιχούν στα pixels του chroma η διαδικασία είναι πιο απλή . Σε αυτή την περίπτωση ελέγχονται 2 pixels και αλλάζει τιμή 1. Εκτός από τον έλεγχο που γίνεται αν 2 blocks έχουν το ίδιο BS ,πρέπει να ληφθεί υπόψη ότι και οι δύο συνιστώσες του Chroma (Cb Cr) θα έχουν το ίδιο προφανώς BS . Άρα είναι δυνατόν

ένας καταχωρητής στις 4 πρώτες θέσεις να έχει τις τιμές του Cb και στις επόμενες 4 θέσεις τις τιμές του Cr . Η διαδικασία που ακολουθείται είναι ίδια και για την περίπτωση που ισχύει ότι 2 blocks έχουν το ίδιο BS και για την περίπτωση που δε είναι. Αλλάζει μόνο ο τρόπος που γράφονται πίσω στην μνήμη . Αφού φορτωθούν οι απαραίτητες γραμμές που απαιτούνται για το φιλτράρισμα τόσο για Cb όσο και για το Cr στην συνέχεια θα πρέπει να ενωθούν σε ένα καταχωρητή. Για να γίνει η ένωση αρχικά πρέπει τα δεδομένα να πακεταριστούν σε 32x4 και στην συνέχεια να συγχωνευτούν σε 16x8. Το πακετάρισμα σε 32x4 γίνεται με την εντολή punpcklwd. Η συγχώνευση γίνεται με χρήση των εντολών packssdw. Στην συνέχεια εκτελούνται οι πράξεις του φιλτραρίσματος. Αγνοώντας σε πρώτη φάση αν τα διαδοχικά blocks 4x4 έχουν διαφορετικό BS. Υπολογίζονται οι τιμές των καταχωρητών iDelta και AbsDelta. Επιπλέον φορτώνονται οι κατάλληλες τιμές στους καταχωρητές iAlpha και iBeta . Υπολογίζεται η αρχική συνθήκη που δείχνει αν θα εφαρμοστεί το φιλτράρισμα ή όχι και το αποτέλεσμα αποθηκεύεται στο καταχωρητή [1]. Οι παραπάνω διαδικασίες είναι ίδιες με την διαδικασία του οριζόντιου φιλτραρίσματος για την φωτεινότητα .Στην συνέχεια ελέγχεται αν τα δυο διαδοχικά blocks έχουν το ίδιο BS.

Έπειτα ελέγχεται το BS .Αν η τιμή του είναι μικρότερη του 4 τότε αρχικά φορτώνεται σε ένα καταχωρητή έστω C0 μια τιμή ,που προσδιορίζεται από την τιμή του BS και του indexA. Συνεχίζοντας την διαδικασία προστίθεται ο καταχωρητής C0 με έναν καταχωρητή στον οποίο έχει φορτωθεί η τιμή 1 .Στο επόμενο βήμα υπολογίζεται ο καταχωρητής iDiff ,τα περιεχόμενα του οποίου δίνονται από τον τύπο $\boldsymbol{gClipMinMax((iDelta \ll 2) + ((P1 - Q1) + 4) \gg 3, -C0, C0)}$. Οι αριθμητικές πράξεις αντικαθιστώνται από τις αντίστοιχες διανυσματικές. Ο περιορισμός του εύρους τιμών από –C0 εώς C0 έχει ήδη περιγραφεί. Στην συνέχεια ο καταχωρητής iDiff προστίθεται με τον καταχωρητή P0, με χρήση της εντολής paddw και το αποτέλεσμα αποθηκεύεται στον P0. Ο καταχωρητής iDiff αφαιρείται από τον καταχωρητή Q0 και το αποτέλεσμα αποθηκεύεται εκ νέου στον Q0. Στο τέλος και για τους 2 καταχωρητές γίνεται περιορισμός του εύρους τιμών από 0 -255.Η διαδικασία αυτή έχει περιγραφεί και αυτή αρκετές φορές σε προηγούμενες ενότητες.

Αν το BS έχει τιμή ίση με 4 τότε τα περιεχόμενα του καταχωρητή Q0 αν ισχύει η συνθήκη που είναι αποθηκευμένη στο καταχωρητή [1] υπολογίζεται από την σχέση $((\mathbf{Q1} << \mathbf{1}) + \mathbf{Q0} + \mathbf{P1} + \mathbf{2}) >> 2$. Ομοίως οι διανυσματικές πράξεις αντικαθιστούν τις αντίστοιχες αριθμητικές πράξεις. Τα αποτελέσματα αποθηκεύονται στον καταχωρητή[2]. Ομοίως για τον

καταχωρητή P0 αν ισχύει η συνθήκη που είναι αποθηκευμένη στο καταχωρητή [1] τότε όλα τα περιεχόμενα του καταχωρητή υπολογίζονται από την μαθηματική σχέση $((\mathbf{P1} << \mathbf{1}) + \mathbf{P0} + \mathbf{Q1} + \mathbf{2}) >> 2$. Τα αποτελέσματα αποθηκεύονται στον καταχωρητή [3].

Η τελική επιλογή των σωστών τιμών για τους καταχωρητές γίνεται με χρήση της εντολής pblendvb. Τέλος για την εγγραφή πίσω στην μνήμη των Cb τιμών χρησιμοποιείται η εντολή movq. Για τις συνιστώσες Cr θα πρέπει να χρησιμοποιηθεί η εντολή psrldq (ολίσθηση κατά 8) για να έρθουν στις χαμηλές θέσεις και στην συνέχεια να εγγραφούν πίσω στην μνήμη με χρήση της εντολής movq.

Αν το BS διαφέρει ανάμεσα σε διαδοχικά blocks η διαδικασία διαφέρει μόνο στο τρόπο εγγραφής πίσω στην μνήμη. Για το πρώτο block οι δύο πρώτες τιμές των καταχωρητών αντιστοιχούν σε αυτό . Άρα χρησιμοποιώντας την εντολή movd που μεταφέρει τα χαμηλότερα 32 bits σε μια μεταβλητή. Με αυτό τον τρόπο αποθηκεύονται και οι δύο τιμές. Στην συνέχεια με την χρήση της memcpy αντιγράφουμε αυτή την ακέραια μεταβλητή σε 2 θέσεις ενός πίνακα short. Με αυτό τον τρόπο και για τους δυο καταχωρητές γράφονται σωστά τα περιεχόμενα πίσω στην μνήμη .Οι επόμενες δύο τιμές των καταχωρητών αντιστοιχούν στο δεύτερο block . Θα πρέπει αυτές οι τιμές να μεταφερθούν στις 2 χαμηλότερες θέσεις, με χρήση της εντολής psrldq (ολίσθηση κατά 4). Στην συνέχεια και για τους δυο καταχωρητές χρησιμοποιείται η εντολή psrldq, ολίσθηση κατά 8 για να έρθουν στις χαμηλότερες θέσεις οι συνιστώσες Cr. Στην συνέχεια επαναλαμβάνεται η διαδικασία

Πριν την χρήση διανυσματικών πράξεων η συνάρτηση αυτή χρειαζόταν συνολικά **327300000** κύκλους μηχανής**.** Με την χρήση διανυσματικών πράξεων η συνάρτηση αυτή χρειάζεται συνολικά **235200000** κύκλους μηχανής ,που αντιστοιχεί σε επιτάχυνση 1.39x και σε ποσοστό 30 %.

4.7.2 Κάθετο φιλτράρισμα

Στην ενότητα αυτή θα περιγραφεί η διαδικασία του κάθετου φιλτραρίσματος. Η λογική που ακολουθείται για την εκτέλεση διανυσματικών πράξεων στις κάθετες ακμές ενός block είναι να γίνει εφικτό η διαδικασία αυτή να γίνει οριζόντια αντί για κάθετα. Αυτό μπορεί να συμβεί μόνο μετατρέποντας τις γραμμές ενός πίνακα σε στήλες και τις στήλες σε γραμμές όπως δείχνει το παρακάτω σχήμα

$$\begin{bmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} (0_0 & 0_1 & 0_2 & 0_3) \\ (1_0 & 1_1 & 1_2 & 1_3) \\ (2_0 & 2_1 & 2_2 & 2_3) \\ (3_0 & 3_1 & 3_2 & 3_3) \end{bmatrix} \longrightarrow \begin{bmatrix} (0_0 & 1_0 & 2_0 & 3_0) \\ (0_1 & 1_1 & 2_1 & 3_1) \\ (0_2 & 1_2 & 2_2 & 3_2) \\ (0_3 & 1_3 & 2_3 & 3_3) \end{bmatrix}$$

4.7.2.1 Φιλτράρισμα των pixel που αντιστοιχούν στην φωτεινότητα

Για να εκτελεστεί κάθετο filtering για το Luma, θα πρέπει να γίνει αντιμετάθεση γραμμών και στηλών σε ένα πίνακα 4x4. Για να γίνει αυτό πρέπει να χρησιμοποιηθούν οι εντολές που κάνουν ανάμειξη των δεδομένων. Αρχικά επειδή τα δεδομένα που πρέπει να χρησιμοποιηθούν πρέπει να είναι 32x4 και αυτά που έρχονται ως είσοδος είναι 16x8 θα πρέπει να χρησιμοποιηθεί η εντολή punpcklwd. Επιπλέον επειδή χρησιμοποιούνται σαν είσοδο block 4x4 πρέπει να φορτωθούν με την εντολή movq και όχι με την εντολή movdqa. Στην συνέχεια αφού σε όλους τους καταχωρητές έχουν φορτωθεί οι κατάλληλες τιμές γίνεται η διαδικασία της μετατροπής των γραμμών σε στήλες και των στηλών σε γραμμές. Η διαδικασία απεικονίζεται στους παρακάτω πίνακες. Θα παρουσιαστεί η διαδικασία αρχικά για τους καταχωρητές Q0,Q1,Q2,Q3 που περιέχουν τις τιμές του τρέχων block και ύστερα των καταχωρητών P0,P1,P2,P3 που περιέχουν τις τιμές του .

I0	punpckldq(Q0,Q1)
I1 →	punckldq(Q2,Q3)
I2	punckhdq(Q0,Q1)
I3	punckhdq(Q2,Q3)

Q0	punpcklqdq(I0,I1)
Q1 →	punckhqdq(I0,I3)
Q2	puncklqdq(I2,I3)
Q3	punckhqdq(I2,I3)

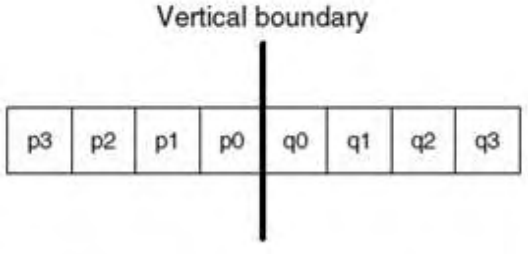
Τέλος θα πρέπει να πακεταριστούν τα δεδομένα σε 16x8. ΄Αρα για κάθε ένα από τα τελικά αποτελέσματα χρησιμοποιείται η εντολή packssdw.

Στην συνέχεια θα περιγραφτεί η διαδικασία για την μετατροπή των γραμμών σε στήλες και των στηλών σε γραμμές του block που βρίσκεται αριστερά από το τρέχων και επηρεάζονται τα pixels στις 3 τελευταίες γραμμές του. Όπως δείχνει ο πίνακας η διαδικασία είναι η ίδια με την προηγούμενη

I0	punpckldq(P0,P1)
I1 →	punckldq(P2,P3)
I2	punckhdq(P0,P1)
I3	punckhdq(P2,P3)

P0	punpcklqdq(I0,I1)
P1 →	punckhqdq(I0,I3)
P2	puncklqdq(I2,I3)
P3	punckhqdq(I2,I3)

Τα δεδομένα όπως και πριν φορτώνονται με χρήση με την εντολή movq. Στο σημείο αυτό πρέπει να δοθεί ιδιαίτερη σημασία στο παρακάτω σχήμα:

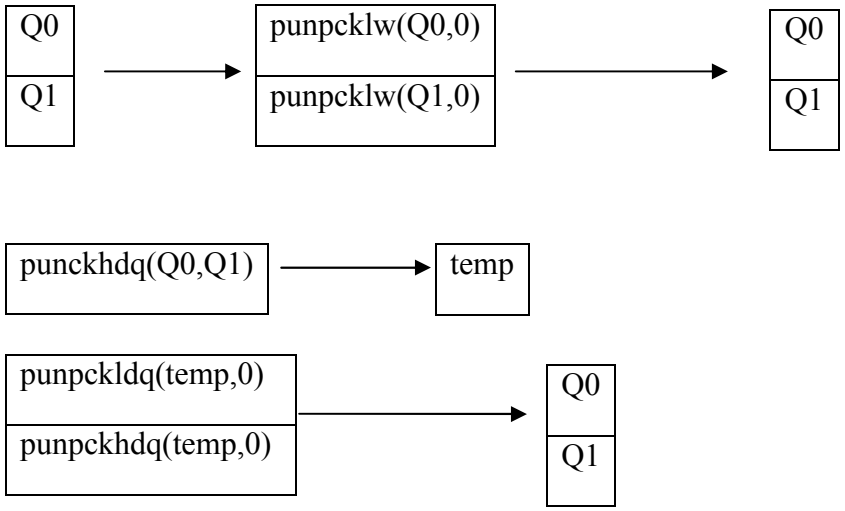


Τα pixels που χρησιμοποιούνται από το τρέχων block για το filtering είναι q0, q1, q2, q3. Από το block που βρίσκεται αριστερά από αυτό είναι τα p0, p1, p2, p3. Δηλαδή πρέπει στις χαμηλότερες θέσεις να βρίσκονται τα pixels p0 q0, όμως φορτώνοντας τις γραμμές του αριστερά block, τα pixels φορτώνονται από αριστερά προς τα δεξιά, με αποτέλεσμα στην θέση 0 να βρίσκεται το p3. Άρα και στην εναλλαγή των γραμμών με στηλών τα περιεχόμενα των καταχωρητών για το αριστερό block θα είναι αντίστροφα από ότι χρειάζονται για την διαδικασία του deblocking . Επομένως μετά την διαδικασία της εναλλαγής των γραμμών πρέπει να αντιστραφούν τα περιεχόμενα των καταχωρητών που αντιστοιχούν στο αριστερό block. Η εναλλαγή αυτή γίνεται χρησιμοποιώντας την εντολή PSHUFD, που δέχεται σαν πρώτο όρισμα τον κατάλληλο καταχωρητή και σαν δεύτερο την macro εντολή _MM_SHUFFLE(z,y,x,w), με ορίσματα z=0 y=1 x=2 w=3. Η macro εντολή αυτή, προσδιορίζει την νέα θέση των δεδομένων, και ορίζεται ως **εξής (z << 6) | (y <<4) | (x << 2) | w**. Στην συνέχεια εκτελείται το filtering οριζόντια, όπως περιγράφτηκε στην 4.7.2.1. Στο τέλος της διαδικασία αποθηκεύονται ξανά τα δεδομένα σωστά με εναλλαγή των γραμμών και των στηλών, όπως ακριβώς περιγράφτηκε παραπάνω .

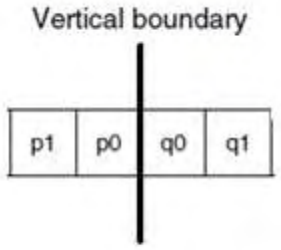
Η βελτιστοποίηση που επιτυγχάνεται με την χρήση διανυσματικών εντολών είναι πολύ μικρή. Συγκεκριμένα η αρχική έκδοση της συνάρτησης χρειαζόταν **812700000** κύκλους μηχανής, με την χρήση διανυσματικών εντολών χρειάζεται **646800000** κύκλους μηχανής. Συνολική βελτιστοποίηση μόνο 1.25x, που αντιστοιχεί σε ποσοστό 21 %.

4.7.2.2 Φιλτράρισμα των pixel που αντιστοιχούν στις συνιστώσες του Chroma Cb Cr

Στην ενότητα αυτή θα περιγραφεί η διαδικασία για το κάθετο filtering για τις συνιστώσες Cb Cr. Όπως και για το οριζόντιο filtering που περιγράφτηκε στην ενότητα 4.7.1.1, και στην περίπτωση αυτή θα γίνει έλεγχος αν δύο διαδοχικά block έχουν το ίδιο BS. Επιπλέον στον ίδιο καταχωρητή θα περιέχονται και συνιστώσες του Cb (θέσεις 0-4) και του Cr (θέσεις του 4-8). Για το πρώτο block φορτώνονται οι απαραίτητες τιμές του τρέχων block καθώς και του αριστερά που απαιτείται για το deblocking σε δυο καταχωρητές Q0,Q1,P0,P1 για το τρέχων και το αριστερά. Πριν την διαδικασία εναλλαγής των γραμμών με τις στήλες θα πρέπει τα δεδομένα που έχουν φορτωθεί να πακεταριστούν σε 32 x4 με χρήση της εντολής punpcklwd. Η εναλλαγή γραμμών αφορά 2x2 blocks. Οι παρακάτω πίνακες δείχνουν την διαδικασία που ακολουθείται για τους καταχωρητές Q0, Q1 του τρέχοντος block.



Η διαδικασία που ακολουθείται για το αριστερό block του τρέχοντος του block είναι ίδια με την παραπάνω διαδικασία. Αντί για τους καταχωρητές Q0, Q1 χρησιμοποιούνται οι καταχωρητές P0,P1. Ιδιαίτερη προσοχή πρέπει να δοθεί στο παρακάτω σχήμα.



Στο τρέχων block αντιστοιχούνται τα blocks q0 και q1. Τα p1 p0 αντιστοιχούν στα pixel του αριστερού block. Όμως στην θέση 0 πρέπει να βρίσκεται το p0 και όχι το p1. Για το λόγο αυτό χρησιμοποιείται η εντολή pshufd με πρώτο όρισμα τους καταχωρητές p1,p0 και δεύτερο όρισμα

την macro εντολή _MM_SHUFFLE2(x,y) με x=0 και y=1. Η εντολή αυτή ορίζεται ως εξής (x<<1) | y. Αφού τελειώσει η διαδικασία εκτελείται οριζόντιο filtering όπως περιγράφτηκε στην ενότητα 4.7.1.2. Η διαδικασία επαναλαμβάνεται για το δεύτερο διαδοχικό κάθετο block που έχει το ίδιο BS. Η συγχώνευση των αποτελεσμάτων για τα δυο blocks γίνεται με χρήση της εντολής packssdw. Ακολουθείται η ίδια διαδικασία και για τις συνιστώσες Cr και τους καταχωρητές που περιέχουν τις τιμές για το Cb και το Cr τους συγχωνεύουμε σε ένα καταχωρητή με χρήση της εντολής packssdw.

Στο τέλος της διαδικασίας πρέπει τα αποτελέσματα να αποθηκευτούν σωστά στην μνήμη .Για τις συνιστώσες Cb του τρέχοντος block και του αριστερά block αρχικά μετατρέπονται οι 2 χαμηλότερες τιμές των καταχωρητών σε integer με χρήση της εντολής movd και αποθηκεύεται στην ακέραια μεταβλητή [1]. Στην συνέχεια αποθηκεύονται σωστά τα 16 χαμηλότερα bits καθώς και τα 16 υψηλότερα. Για το pixel που αντιστοιχεί στην πρώτη γραμμή η τιμή του παράγεται από την λογική πράξη "και " της μεταβλητής [1] με την τιμή 0xFFF (4095). Η τιμή του pixel της δεύτερης γραμμής παράγεται με λογική ολίσθηση κατά 16 θέσεις 16 της μεταβλητής [1].Για το δεύτερο block ολισθαίνεται ο καταχωρητής με χρήση της ενολής psrldq κατά 4 θέσεις για να μεταφερθούν στις χαμηλότερες θέσεις .Χρησιμοποιείται η ίδια διαδικασία για να γραφτούν τα σωστά pixels πίσω στην μνήμη. Για τις συνιστώσες Cr θα πρέπει οι καταχωρητές να ολισθήσουν κατά 8 θέσεις δεξιά με χρήση της εντολής psrldq και μετά επαναλαμβάνεται η διαδικασία.

Συνοψίζοντας η αρχική έκδοση του κώδικα χωρίς χρήση διανυσματικών εντολών χρειαζόταν **323400000** κύκλους μηχανής .Η τελική έκδοση με χρήση διανυσματικών εντολών απαιτεί **266700000** κύκλους μηχανής ,συνολική βελτιστοποίηση 1.21x που αντιστοιχεί σε ποσοστό 18 %.

4.8 Συνολική βελτιστοποίηση deblocking filter

Εκτός από τις παραπάνω συναρτήσεις πρέπει να ληφθεί υπόψη και η συνάρτηση που κάνει περιορισμό του εύρους των τιμών ανάμεσα σε δύο όρια [min max]. Η συνάρτηση αυτή αντικαθιστάται από διανυσματικές εντολές. Στο πίνακα που ακολουθεί συνοψίζονται τα αποτελέσματα της βελτιστοποίησης.

Διαδικασία	Μη Βελτιστοποιημένη	Βελτιστοποιημένη	Επιτάχυνση

Horizontal Filtering	1169700000	697200000	1.68
Vertical Filtering	1136100000	913500000	1.24
Περιορισμός του εύρους τιμών(gClipMinMax)	583800000	-	-
‘Αθροισμα	2889600000	1610270000	1.78

Άρα η συνολική επιτάχυνση είναι 1.79x που αντιστοιχεί σε ποσοστό 44.3 %.

4.9Inverse Transform

Από την διαδικασία του inverse transform παράγονται οι τιμές των pixels που είτε θα γραφτούν πίσω στην μνήμη για να κατασκευαστεί η εικόνα περνώντας από την διαδικασία του deblocking είτε χωρίς την εκτέλεση της διαδικασίας του deblocking, για τα intra macroblocks. Για τα inter macroblocks υπολογίζει τα υπόλοιπα που στην συνέχεια θα προστεθούν είτε για να χρησιμοποιηθούν στο επόμενο επίπεδο είτε για να χρησιμοποιηθούν για την κατασκευή του inter καρέ. Η μαθηματική σχέση που υλοποιεί το inverse transform δίνεται ως εξής $Y = Ci^{T}(YxEi)Ci$. Όπου ο πίνακας Ci ορίζεται ως εξής. Το inverse transform λαμβάνει χώρα σε blocks 4x4.

1	1	1	½
1	½	-1	-1
1	½	-1	-1
1	-1	1	-1/2

Ο πίνακας Yi περιέχει τους συντελεστές που δε έχουν υποστεί κλιμάκωση και πολλαπλασιάζονται με τον πίνακα Ei που περιέχει τους παράγοντες κλιμάκωσης . Στο τέλος για κάθε γραμμή [i,j] προστίθεται μια τιμή που είτε προέρχεται από την διαδικασία intra prediction είτε από την διαδικασία του upsampling στον μηχανισμό του inter-layer prediction.

Στην συνέχεια θα αναλυθεί η διαδικασία που ακολουθήθηκε. Στο αρχικό κώδικα η διαδικασία εκτελείται τόσο για τις γραμμές όσο και για τις στήλες . Για την οριζόντια διαδικασία σε κάθε επανάληψη φορτώνονται 4 συντελεστές(coeff0, coeff1, coeff2, coeff3, coeff4) και τα αποτελέσματα αποθηκεύονται αντίστοιχα στον προσωρινό πίνακα a στις αντίστοιχες θέσεις [0][i],[1][i],[2][i],[3][i] ,όπου i ο αριθμός της επανάληψης.

Σε μία προσωρινή μεταβλητή αποθηκεύεται το αποτέλεσμα της πρόσθεσης των coeff0 και coeff2. Σε μια δεύτερη μεταβλητή αποθηκεύεται το αποτέλεσμα της πρόσθεσης του coeff3 ολισθημένου κατά 1 θέση δεξιά και coeff1. Οι δυο προσωρινές μεταβλητές προστίθενται και το αποτέλεσμα αποθηκεύεται στην θέση [0][i] του πίνακα ,επίσης στην θέση [3][i] αποθηκεύεται η διαφορά τους .Στην συνέχεια σε μια προσωρινή μεταβλητή αποθηκεύεται το αποτέλεσμα της διαφορά coeff1 και coeff2, και σε μια άλλη το προσωρινό αποτέλεσμα της αφαίρεσης του coeff3 ολισθημένου κατά μια θέση δεξιά με το coeff1. Οι 2 προσωρινές μεταβλητές προστίθενται και το αποτέλεσμα τους αποθηκεύεται στην θέση [1][i] και αντίστοιχα αφαιρούνται και το αποτέλεσμα αποθηκεύεται στην θέση [2][i]. Οι παραπάνω πράξεις μπορούν να πολλαπλασιαστούν και να διαιρεθούν στο τέλος με 2 για να μην αλλοιωθούν τα αποτελέσματα . Άρα για την οριζόντια διαδικασία προκύπτουν οι εξής σχέσεις

- $\boldsymbol{a[0][i] = \big((coeff0 + coeff2) \ll 1 + coeff3 + coeff1 \ll 1\big) \gg 1}$
- $\boldsymbol{a[3][i] = (coeff0 + coeff2) \ll 1 - \big(coeff3 + (coeff1 \ll 1)\big) \gg 1}$
- $\boldsymbol{a[1][i] = (coeff0 - coeff2) \ll 1 + (coeff1 - coeff3 \ll 1) \gg 1}$
- $\boldsymbol{a[2][i] = (coeff0 - coeff2) \ll 1 - (coeff1 - coeff3 \ll 1) \gg 1}$

Για τον υπολογισμό των παραπάνω με διανυσματικές πράξεις μπορεί να χρησιμοποιηθεί η εντολή pmaddw αρκεί να οριστούν τα κατάλληλα φίλτρα .Επιπλέον είναι δυνατός ο υπολογισμός ολόκληρης γραμμής σε μία επανάληψη .

Ο παρακάτω πίνακας παρουσιάζει τις τιμές των φίλτρων για κάθε στήλη.

Γραμμή								
0	2	2	1	2	2	2	2	1
1	2	1	-2	-2	2	1	-2	-2
2	2	-1	-2	-2	2	-1	-2	-2

3	2	-2	2	-1	2	-2	2	-1

Τα φίλτρα τα φορτώνουμε σε κατάλληλους καταχωρητές χρησιμοποιώντας την εντολή movdqa. Στην συνέχεια φορτώνουμε τους συντελεστές που έχουν πολλαπλασιαστεί με τους κατάλληλους συντελεστές κλιμάκωσης. Χρησιμοποιώντας την εντολή pmaddw παράγονται τα αποτελέσματα για κάθε γραμμή . Για παράδειγμα για την γραμμή 0 χρησιμοποιείται το φίλτρο [2 2 1 2 2 2 1 2] .Έτσι γίνει η υπόθεση ότι υπάρχουν οι συντελεστές c0 c1 c2 c3 c4 c5 c6 c7. Τότε με χρήση της εντολής madd προκύπτει το ακόλουθο αποτέλεσμα

c0*2+c1*2	c2*2 c3*1	c4*2+c5*2	c6*2+ c7*1

Με τον ίδιο τρόπο προκύπτουν και τα αποτελέσματα για τους συντελεστές c8,c9..c15.

Ενώνονται οι δυο καταχωρητές χρησιμοποιώντας την εντολή packssdw. Στην συνέχεια φορτώνεται το φίλτρο [1 1 1 1 1 1 1 1] και χρησιμοποιείται η εντολή pmaddw για να προστεθούν ανά 2 τα προσωρινά αποτελέσματα της πρώτης pmaddw. Επιπλέον σε ένα καταχωρητή μπορούν να συνενωθούν 2 καταχωρητές που περιέχουν τις τιμές 2 διαδοχικών γραμμών με χρήση της εντολής packssdw. Άρα η διαδικασία αφού ολοκληρωθεί προκύπτουν 2 νέοι καταχωρητές που περιέχουν τις γραμμές 0-1 και 2-3 αντίστοιχα. Τα περιεχόμενα των καταχωρητών αυτών διαιρούνται με το 2 με χρήση της εντολής psraiw.

Για την κάθετη διαδικασία στον αρχικό κώδικα για κάθε επανάληψη υπολογίζεται η τιμή του pixel στη θέση j της αντίστοιχης γραμμής . Οι σχέσεις δίνονται απο τους παρακάτω τύπους:

- $puc[0] = (a[y][0] + a[y][2] + a[y][3] \gg 1 + a[y][1] + 32) \gg 6$
- $puc[3 * iStride] = (a[y][0] + a[y][2] - a[y][3] \gg 1 - a[y][1] + 32) \gg 6$
- $puc[iStride] = (a[y][0] - a[y][2] + a[y][1] \gg 2 - a[y][3] + 32) \gg 6$
- $puc[iStride2] = (a[y][0] - a[y][2] - a[y][1] \gg 2 + a[y][3] + 32) \gg 6$

Οπώς και για την προηγούμενη διαδικασία και σε αυτήν την φάση είναι δυνατόν να πολλαπλασιαστούν οι παραπάνω τύποι με 2 και πριν την πρόσθεση του 32 και την ολίσθηση κατά 6 να διαιρεθούν με 2. Ήδη έχουν δημιουργηθεί 2 καταχωρητές με περιεχόμενα τα ακόλουθα :

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Η προηγούμενη διαδικασία επαναλαμβάνεται χρησιμοποιώντας τους νέους καταχωρητές. Στο τέλος της διαδικασίας πρέπει να προστεθεί η τιμή 32 και να διαιρεθεί το περιεχόμενο των καταχωρητών με 64 .Αυτό γίνεται χρησιμοποιώντας την εντολή paddw και αφού έχουμε φορτώσει σε κατάλληλο καταχωρητή την τιμή 32. Στη συνέχεια το αποτέλεσμα διαιρείται με το 64 χρησιμοποιώντας την εντολή psraw (ολίσθηση κατά 6). Χρησιμοποιώντας την εντολή movq φορτώνουμε 4 καταχωρητές με τις γραμμές 0 iStride 2*iStride και 3*iStride. Συγχωνεύονται τα αποτελέσματα ανά 2 με την σειρά που δίνονται ολισθαίνοντας τους καταχωρητές που περιέχουν τις γραμμές iStrisde και 3*iStride κατά 8 θέσεις αριστερά με χρήση της εντολής pslldq και της εντολής por. Στην συνέχεια προσθέτουμε τον νέο καταχωρητή στους αντίστοιχους καταχωρητές. Κατόπιν, αποθηκεύονται τα αποτελέσματα χρησιμοποιώντας την εντολή movq και για τα pixels που βρίσκονται στις υψηλότερες θέσεις μετατοπίζονται στις χαμηλότερες χρησιμοποιώντας την εντολή psrldq και στην συνέχεια γράφονται πίσω στην μνήμη.

Αρχικά η υλοποίηση του inverse transform χρειαζόταν συνολικά 1566600000 κύκλους μηχανής. Μετά την χρήση διανυσματικών πράξεων συνολικά χρειάζονται 485100000, που αντιστοιχεί σε επιτάχυνση 3.2 x. Σε ποσοστό δηλαδή 70 %.

4.10 Συνάρτηση storeToPicBuffer

Η συνάρτηση αυτή καλείται όταν το καρέ που έχει αποκωδικοποιηθεί δε είναι καρέ αναφοράς. Κατά την διάρκεια της αποκωδικοποίησης μέσα στον αποκωδικοποιητή υπάρχουν 2

τοπικές προσωρινές αποθήκες που αποθηκεύονται τα καρέ αναφοράς. Επιπλέον υπάρχει μια εξωτερική προσωρινή αποθήκη που αποθηκεύονται όλα τα καρέ που δε είναι καρέ αναφοράς και στην συνέχεια θα εμφανιστούν στην οθόνη. Η αποθήκη αυτή κάνει χρονική αναδιάταξη των καρέ και κάθε φορά επιλέγεται το τελευταίο χρονικά καρέ για απεικόνιση ή εγγραφή στον δίσκο. Στην περίπτωση αυτή καλείται η storeToPicBuffer. Η συνάρτηση αυτή γράφει τα καρέ από τον τοπικό προσωρινό καταχωρητή που τα δεδομένα αποθηκεύονται ως short στον εξωτερικό buffer, που τα δεδομένα έχουν μήκος 8-bits, αφού πρώτα γίνει περιορισμός του εύρους από 0 - 255. Με χρήση διανυσματικών εντολών, αρχικά φορτώνονται τα δεδομένα που πρέπει να γραφτούν στον εξωτερικό buffer με χρήση της εντολής movdqa και στην συνέχεια αφού πακεταριστούν και περιοριστεί το εύρος των τιμών τους από 0-255 γράφονται στον εξωτερικό buffer. Ο περιορισμός του εύρος από 0 -255 γίνεται με χρήση της εντολής packusb και η εγγραφή πίσω στην μνήμη με χρήση της εντολής movq. Η υλοποίηση της συνάρτησης χωρίς χρήση διανυσματικών εντολών χρειαζόταν για την ολοκλήρωση της συνολικά **1110900000** κύκλους μηχανής. Με την χρήση διανυσματικών εντολών οι συνολικοί κύκλοι που χρειάζονται είναι **436800000** κύκλοι μηχανής. Άρα συνολικά η συνάρτηση επιταχύνθηκε κατά 2.54x, που αντιστοιχεί σε ποσοστό 60.4 %.

4.11 Αντιγραφή δεδομένων και μηδενισμός θέσεων μνήμης με χρήση διανυσματικών εντολών

Το σύνολο διανυσματικών εντολών που παρέχονται στις αρχιτεκτονικές x86,δίνουν την δυνατότητα αντιγραφής block μνήμης από την μια περιοχή μνήμης σε μια άλλη με χρήση διανυσματικών εντολών. Χωρίς χρήση διανυσματικών εντολών η αντίστοιχη εντολή που χρησιμοποιείται για είναι η εντολή memcpy(pDes,pSrc,sizeof(type)*BlockSize). Αυτή η εντολή μπορεί να αντικατασταθεί από δύο διανυσματικές εντολές. Στην πρώτη φάση αντιγράφονται τα περιεχόμενα από την διεύθυνση μνήμης που ξεκινάει από pSrc σε ένα καταχωρητή __m128i, με χρήση της εντολής movdqa . Στην συνέχεια τα δεδομένα που έχουν αποθηκευτεί στον καταχωρητή __m128i αντιγράφονται στην διεύθυνση pDes με χρήση της εντολής movdqa. Η χρήση διανυσματικών εντολών παρέχει την δυνατότητα αντιγραφής ομάδες bytes αντί για αντιγραφή byte προς byte. Επιπλέον με χρήση διανυσματικών εντολών μπορεί να μηδενιστεί

μια περιοχή μνήμης αρκετά γρήγορα. Αρχικά χρησιμοποιείται ένας καταχωρητής τα περιεχόμενα του οποίου μηδενίζονται χρησιμοποιώντας την εντολή pxor. Στην συνέχεια τα περιεχόμενα του καταχωρητή αντιγράφονται στην κατάλληλη περιοχή μνήμης που πρέπει να μηδενιστεί .

Κεφάλαιο 5

5.1 Εισαγωγή

Στο κεφάλαιο αυτό θα παρουσιαστούν οι αρχιτεκτονικές αλλαγές οι όποιες έγιναν εκτός από την παραλληλοποίηση των δεδομένων με διανυσματικές εντολές

5.2 Intra Prediction με χρήση τοπικού buffer

Αν ένα block έχει κωδικοποιηθεί σαν intra , και δε έχει παραχθεί από την διαδικασία του upsampling σύμφωνα με την διαδικασία του Inter layer prediction , τότε αυτό το block μπορεί να αποκωδικοποιηθεί χρησιμοποιώντας ως βάση για την πρόβλεψη των δικών του pixels ένα προηγούμενο block που έχει ήδη αποκωδικοποιηθεί αλλά δε έχει περάσει από την διαδικασία του φιλτραρίσματος. Οι τύποι κωδικοποίησης ενός intra macroblock στον SVC είναι 4 :

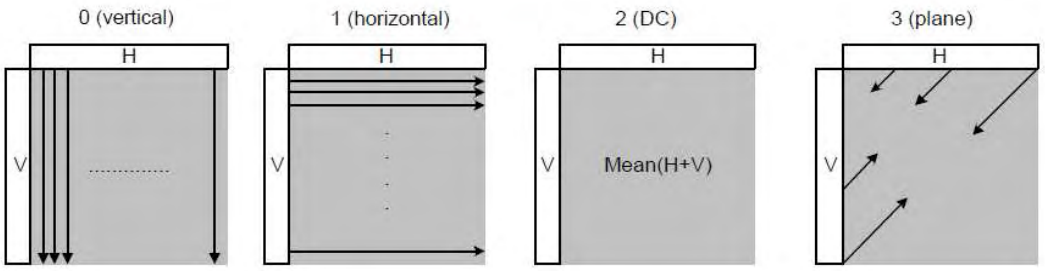
- **INTRA_BL**: Σε αυτή την περίπτωση χρησιμοποιείται το προηγούμενο επίπεδο για να παραχθεί το τρέχον macroblock
- **INTRA_4x4**: Το intra prediction λαμβάνει χώρα σε επίπεδο block 4x4
- **INTRA_16x16**: Το intra prediction λαμβάνει χώρα σε επίπεδο macroblock
- **INTRA_8x8**: Το intra prediction λαμβάνει χώρα σε επίπεδο block 8x8

Στην εκπόνηση της παρούσας διπλωματικής εργασίας θα δοθεί βάση στους 3 πρώτους τύπους.

Για το chroma η μέθοδος που χρησιμοποιείται είναι Intra8x8. Για να προβλεφθεί ένα block ή ένα macroblock χρειάζεται η τελευταία γραμμή του πάνω block ή macroblock, ένα pixel του πάνω αριστερά block η macroblock καθώς και η τελευταία στήλη του αριστερά macroblock ή block. Για το λόγο αυτό ορίζονται 2 τοπικοί buffers ένας που αποθηκεύονται οι τιμές της τελευταίας γραμμής με μέγεθος $1 * iWidth$ που iWidth το μήκος του καρέ και ένας buffer που αποθηκεύονται οι τιμές της τελευταίας στήλης του αριστερού macroblock και το διαγώνιο, 17 θέσεων. Οι προσωρινές αυτές αποθήκες είναι τύπου short. Η χρήση τοπικού buffer χρησιμοποιείται για να αποφεύγονται οι πολλές προσπελάσεις στην μνήμη για γίνεται η ανάγνωση των pixels που απαιτούνται για το intra prediction που συνήθως είναι ακριβές. Στην συνέχεια για κάθε περίπτωση ξεχωριστά θα περιγραφεί η διαδικασία που ακολουθήθηκε.

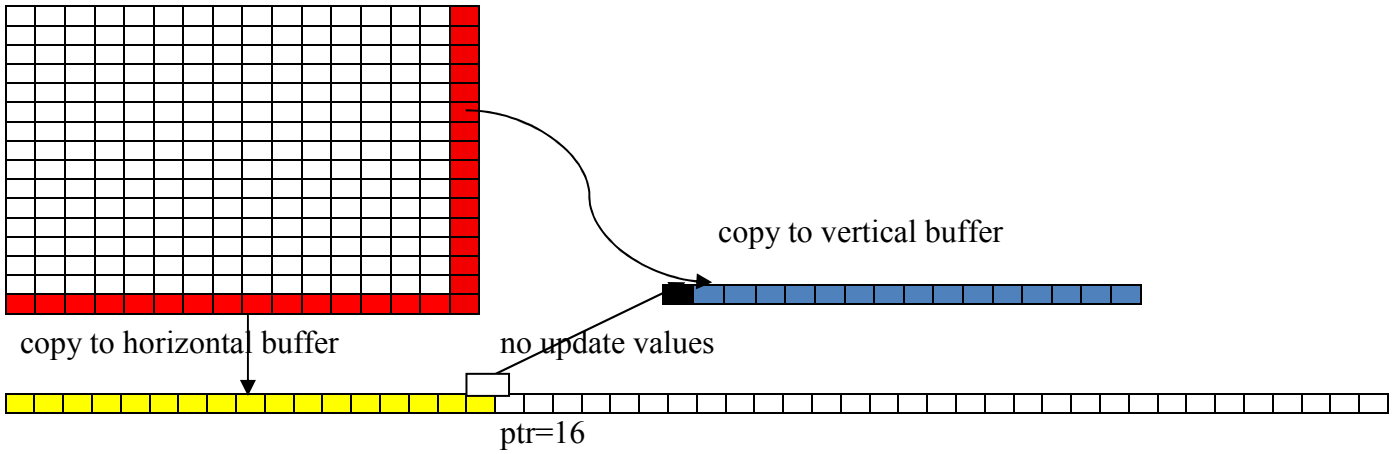
5.2.1 Intra_16x16

Για την μέθοδο του intra prediction με χρήση του προηγούμενου macroblock υπάρχουν 4 διαφορετικοί τρόποι που είναι δυνατή να γίνει η διαδικασία της πρόβλεψης . Οι τρόποι αυτοί καθορίζονται από τον encoder . Το παρακάτω σχήμα δείχνει τους 4 πιθανούς τρόπους πρόβλεψης για intra_16x16.



H.26L Intra 16x16 prediction modes (all predicted from pixels H and V)

Συνεχίζοντας θα περιγραφεί ο τρόπος που αποθηκεύονται οι τιμές των pixels που χρειάζονται για το intra prediction 16x16 . Για την αποθήκευση των τιμών στο buffer που αποθηκεύονται οι τιμές της τελευταίας γραμμής χρησιμοποιείται η έννοια του παραθύρου ολίσθησης . Δηλαδή υποθέτουμε ότι η τοπική αποθήκη είναι ήδη γεμάτη και περιέχει όλες τις τιμές των macroblock της γραμμής i που περιέχει όλα τα macroblock με διεύθυνση [i,0] [i,1],[i, 2]…[i, iWidth-1]. Την τρέχουσα χρονική στιγμή ο αποκωδικοποιητής πρέπει να αποκωδικοποιήσει την γραμμή i+1. Ο δείκτης που δείχνει σε ποιά θέση θα αποθηκευτούν τα νέα pixels βρίσκεται στο τέλος της τοπικής αποθήκης .Άρα θα πρέπει να επιστρέψει στην θέση 0 και να γραφτούν τα νέα δεδομένα του νέου macroblock. Εφόσον οι τιμές του πάνω macroblock χρησιμοποιηθούν μπορούν να ανανεωθούν με τις νέες και να αυξηθεί ο δείκτης κατά 16 έτσι ώστε να δείχνει στο νέο macroblock. Όμως πριν ανανεωθούν θα πρέπει να αποθηκευτεί η τιμή του διαγώνιου pixel στην θέση 0 της προσωρινής μνήμης που αποθηκεύονται οι τιμές της τελευταίας στήλης του αριστερά macroblock . Τα παρακάτω σχήματα απεικονίζουν την διαδικασία.



Στο παραπάνω σχήμα με κόκκινο χρώμα απεικονίζονται η γραμμή και η αντίστοιχη στήλη που πρέπει να αποθηκευτεί στην τοπική προσωρινή μνήμη .Με μπλε χρώμα απεικονίζονται οι 16 θέσεις στις οποίες πρέπει να αποθηκευτεί η τελευταία στήλη και με κίτρινο οι ανανεωμένες τιμές στον οριζόντιο buffer και με άσπρο οι τιμές των macroblock της προηγούμενης γραμμής που χρειάζονται για να αποκωδικοποιηθούν σωστά τα macroblocks δεξιά του τρέχοντος. Αυτές οι τιμές θα ανανεωθούν όταν αποκωδικοποιηθούν τα macroblocks. Με μαύρο απεικονίζεται το διαγώνιο pixel που πρέπει να αποθηκευτεί στον vertical buffer πριν ανανεωθεί ο οριζόντιος buffer.Το διαγώνιο pixel απεικονίζεται με άσπρο ορθογώνιο πάνω από το αντίστοιχο κίτρινο.

5.2.2 Intra Prediction 4x4

Στην διαδικασία αυτή οι πιθανοί τρόποι πρόβλεψης ενός intra macroblock είναι 9. Τα παρακάτω σχήματα απεικονίζουν όλους τους δυνατούς τρόπους της μεθόδου intra-prediction 4x4.

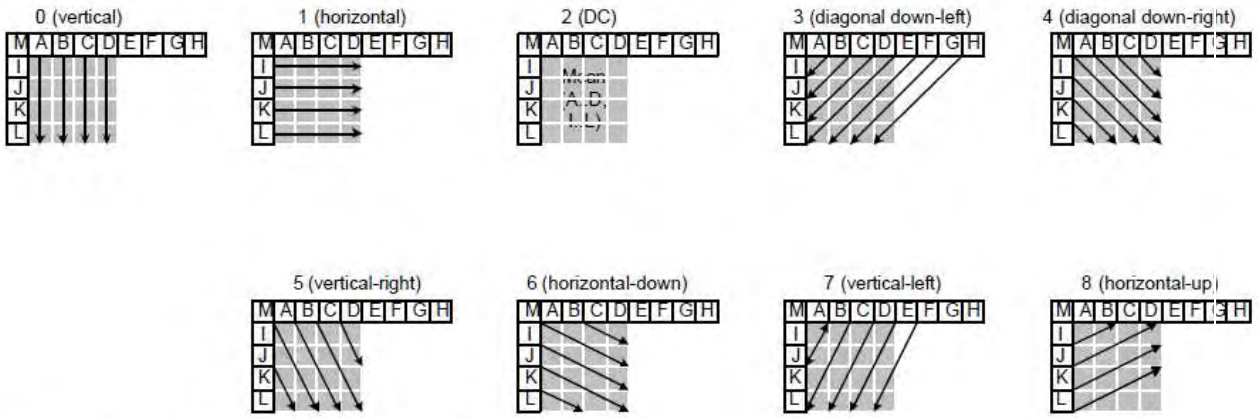
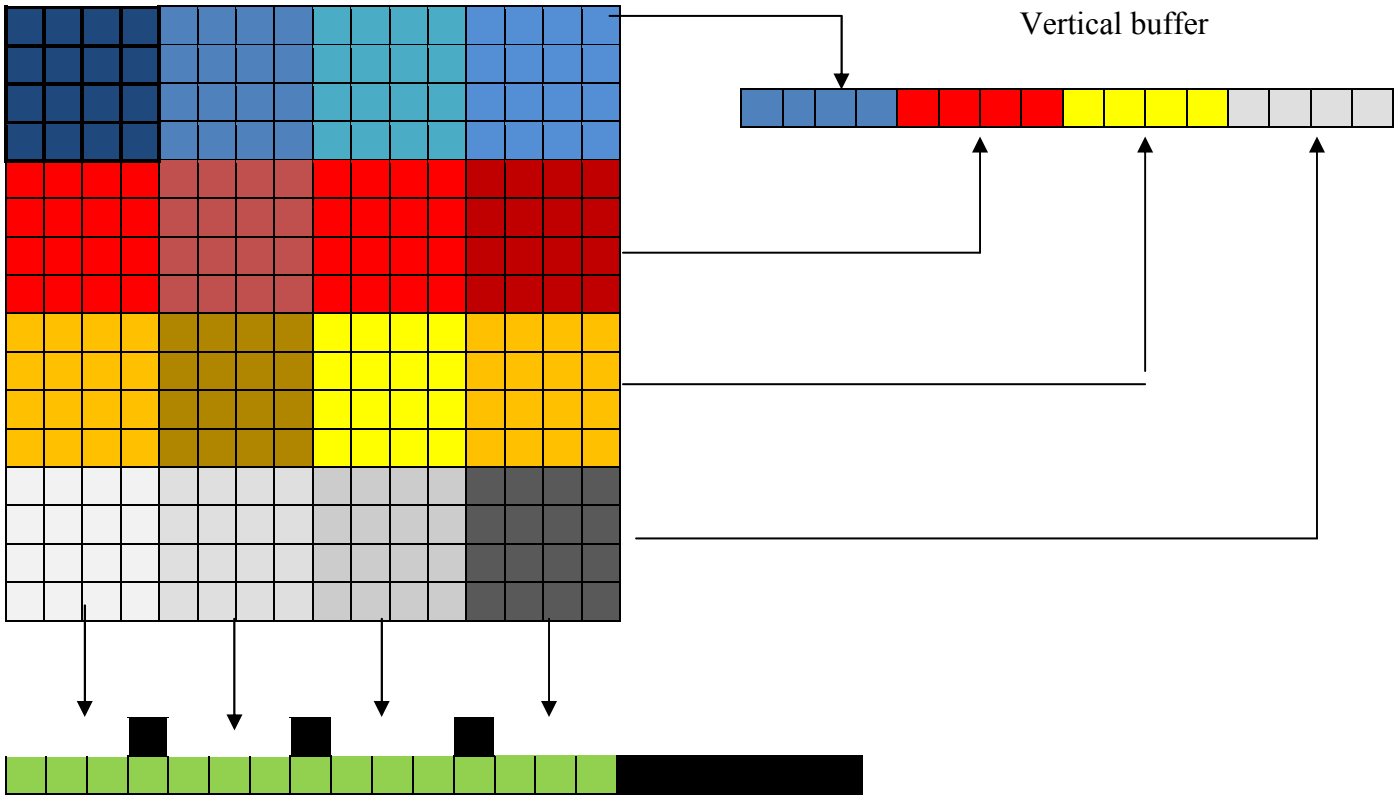


Figure 3 4x4 luma prediction modes

Όπως φαίνεται και από τις εικόνες στις οριακές περιπτώσεις χρειάζεται το διαγώνιο pixel του πάνω αριστερά macroblock και 4 pixel από το πάνω δεξιά macroblock.Η διαδικασία διαφοροποιείται λίγο από την προηγούμενη διαδικασία. Και στην περίπτωση αυτή χρειαζόμαστε δυο τοπικές προσωρινές αποθήκες, ίδιου μεγέθους με πριν . Επιπλέον χρειαζόμαστε άλλη μια αποθήκη για να αποθηκευτούν τα διαγώνια pixels της προηγούμενης γραμμής . Κατά την εκτέλεση της διαδικασίας αυτή όταν ολοκληρωθεί η αποκωδικοποίηση ενός block 4x4 η τελευταία γραμμή και η τελευταία στήλη τοποθετούνται στις κατάλληλες θέσεις στις προσωρινές θέσεις αποθήκευσης. Πιο αναλυτικά κατά την διαδικασία αποκωδικοποίησης ενός macroblock που είναι κωδικοποιημένο σε blocks 4x4 αποτελείται από 4 blocks σε κάθε γραμμή και έχει συνολικά 4 γραμμές. Άρα υπάρχουν 16 4x4 blocks. Κατά την αποκωδικοποίηση μιας σειράς από 4x4 blocks, κάθε φορά που αποκωδικοποιείται ένα block, στις 4 αντίστοιχες θέσεις της τοπικής προσωρινής μνήμης αποθηκεύεται η τελευταία γραμμή του 4x4 block και στις αντίστοιχες 4x4 θέσεις της τοπικής μνήμης, που αποθηκεύονται οι τιμές της τελευταίας στήλης ενός macroblock, αποθηκεύεται η τελευταία στήλη του macroblock. Επιπλέον πριν την ανανέωση των 4 θέσεων της τοπικής μνήμης, που κρατάει τα οριζόντια pixels, αποθηκεύονται

σε ένα τοπικό buffer τα αντίστοιχα διαγώνια pixels(το τελευταίο του τρέχοντος block) που πρέπει να χρησιμοποιηθούν από το κάτω δεξιά block 4x4. Ολοκληρώνοντας την διαδικασία, δηλαδή αποκωδικοποιώντας όλο το macroblock στις αντίστοιχες θέσεις και των 2 τοπικών αποθηκών έχουν αποθηκευτεί και οι 16 τιμές που χρειάζονται για το επόμενο macroblock καθώς και το διαγώνιο pixel που χρειάζεται για το κάτω δεξιά macroblock .

Τα παρακάτω σχήματα περιγράφουν την διαδικασία.



Στο μεγάλο σχήμα που απεικονίζει ένα macroblock οι πρώτες 4x4 γραμμές ,με τις αποχρώσεις του μπλε απεικονίζουν τα 4x4 blocks, 0, 1, 2, 3. Όταν καθένα από αυτά αποκωδικοποιηθεί, η τελευταία του στήλη γράφεται στο vertical buffer (παραπάνω σχήμα) στις

θέσεις που είναι με μπλε χρώμα. Έτσι όταν τελειώσει και το block 3 την αποκωδικοποίηση στο vertical buffer έχουν αποθηκευτεί τα δεδομένα που χρειάζονται από το επόμενο macroblock . Ομοίως η διαδικασία συνεχίζει και για τα υπόλοιπα blocks που φαίνονται στην εικόνα. Τελικά όταν έχουν αποκωδικοποιηθεί τα blocks 3, block 7 (το 4x4 block που απεικονίζεται με χρώμα σκούρο κόκκινο),block 12 (το αριστερότερο 4x4 block που απεικονίζεται με σκούρο κίτρινο χρώμα), το block 15 (το αριστερό 4x4 block που απεικονίζεται με σκούρο γκρι), τότε έχουν αποθηκευτεί όλες οι κατάλληλες τιμές της τελευταίας στήλης που χρειάζονται από το επόμενο αριστερό macroblock. Επιπλέον στα παραπάνω σχήματα εμφανίζεται και ο horizontal buffer που περιέχει τις τιμές που χρειάζεται το block και κατ' επέκταση στο macroblock που βρίσκεται κάτω από το τρέχων. Και για την διαδικασία του intra_4x4 υιοθετείται η έννοια του παραθύρου ολίσθησης. Με πράσινο χρώμα φαίνονται οι ανανεωμένες τιμές μετά από την αποκωδικοποίηση του τρέχοντος block που πρέπει να χρησιμοποιηθούν από το επόμενο macroblock.Με μαύρο χρώμα φαίνονται οι τιμές που αντιστοιχούν σε μη ανανεωμένες τιμές που πρέπει να χρησιμοποιήσει το δεξιά macroblock και θα ανανεωθούν από αυτό. Επιπλέον με μαύρο χρώμα πάνω από τα αντίστοιχα πράσινα φαίνονται τα τελευταία pixels που πρέπει να χρησιμοποιηθούν από τα αντίστοιχα κάτω δεξιά macroblocks και δε πρέπει να χαθούν κατά την διάρκεια της ανανέωσης. Αφού αποκωδικοποιηθούν και τα τελευταία blocks 4x4 (γκρι χρώμα) δημιουργείται η τελευταία γραμμή που θα χρησιμοποιηθεί για το κάτω από το τρέχων macroblock.

5.2.3INTRA_BL

Στην διαδικασία αυτή δε γίνεται intra prediction .Τα pixels που χρησιμοποιούνται στην διαδικασία του inverse transform προέρχονται από την διαδικασία του Inter Layer Intra Prediction. Όταν τελειώσει η διαδικασία αυτή τα pixels αποθηκεύονται στις κατάλληλες τοπικές αποθήκες για να χρησιμοποιηθούν από τα επόμενα macroblocks που θα είναι κωδικοποιημένα σαν intra 16x16 ή intra 4x4.

5.2.4 Chroma Intra Prediction με χρήση τοπικής προσωρινής μνήμης

Η διαδικασία του intra prediction για το chroma είναι ακριβώς ίδια με την διαδικασία του intra 16x16 που περιγράφτηκε στην ενότητα 5.2.1 . Η μόνη διαφορά είναι ότι το μέγεθος ενός block χρώμα είναι 8x8 και το μέγεθος του buffer που χρησιμοποιείται είναι $\mathbf{1} * (\boldsymbol{iWidth} \gg \mathbf{1})$.

5.2.5 Intra Macroblocks σε inter frames

Εκτός από τα intra frames που έχουν μόνο intra macroblocks, υπάρχει περίπτωση και τα P και B frames να έχουν intra macroblocks, αλλά με πολύ μικρή πιθανότητα .Για το λόγο κατά την εκτέλεση της αποκωδικοποίησης ενός macroblock που είναι είτε intra 4x4 είτε 16x16 ελέγχεται αν το αριστερό η το πάνω από αυτό καθώς και το δεξιά αν αποκωδικοποιήθηκε σαν inter . Ο έλεγχος αυτός λαμβάνει χώρα διότι αν ένα από αυτά τα macroblocks ή περισσότερα από ένα έχουν αποκωδικοποιηθεί σαν inter δε έχουν αποθηκευτεί στις τοπικές προσωρινές αποθήκες οι τιμές των pixels που χρειάζονται για να αποκωδικοποιηθεί σωστά το τρέχων macroblock. Έτσι αν το τρέχων macroblock είναι intra 4x4 ή 16x16 και ένα η περισσότερα των macroblocks από τα οποία εξαρτάται είναι inter τότε θα πρέπει να αποθηκευτούν από τα ήδη αποκωδικοποιημένα macroblocks οι κατάλληλες τιμές των pixel έτσι ώστε να αποκωδικοποιηθεί σωστά το τρέχων .

5.2.6 Σύνοψη της χρήσης τοπικών χώρων αποθήκευσης για το *intra prediction*

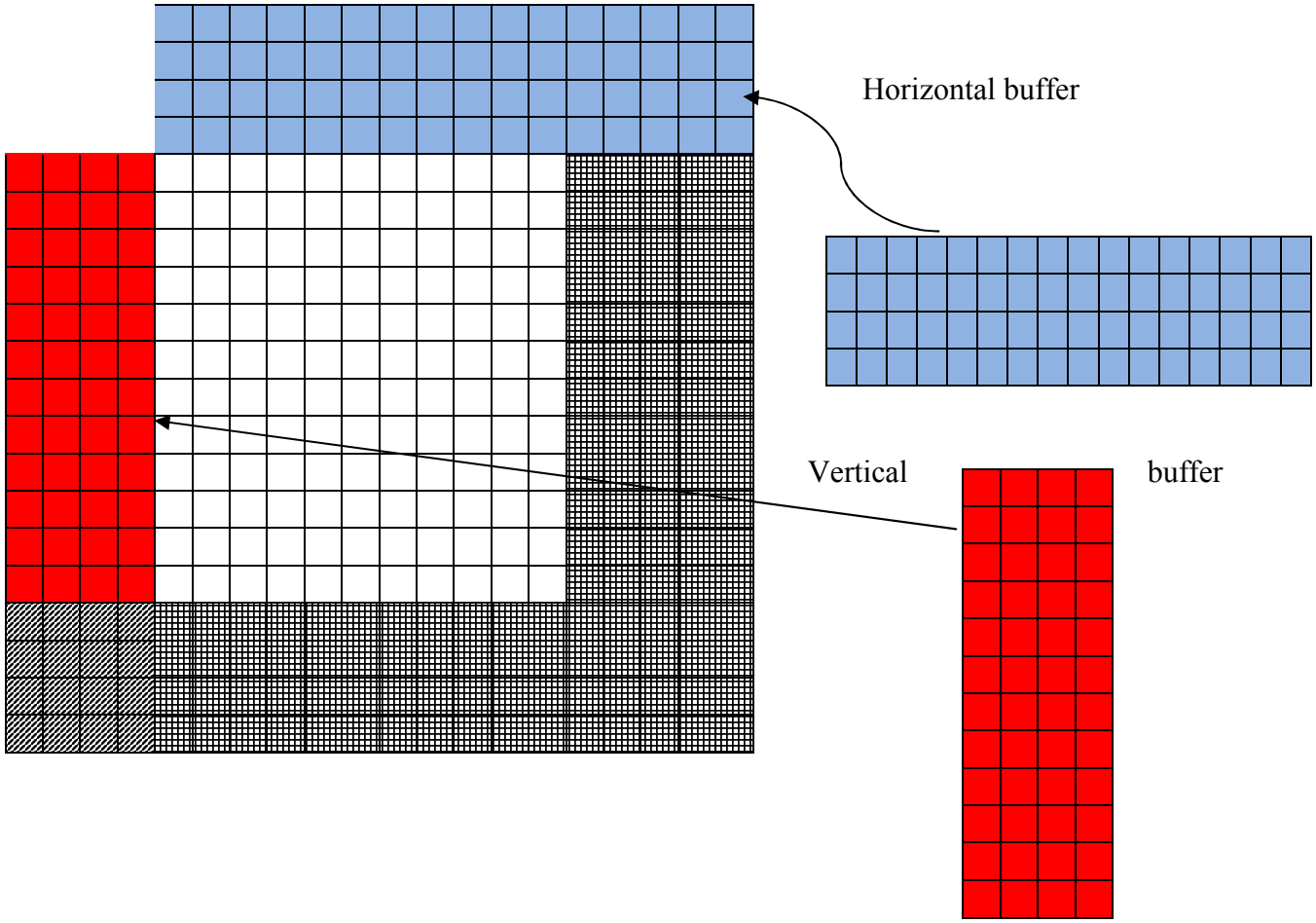
Η αρχιτεκτονική αλλαγή αυτή δε έδωσε περαιτέρω επιτάχυνση στην διαδικασία της αποκωδικοποίησης. Αυτό συνέβη κυρίως λόγω της μεγάλης κρυφής μνήμης που έχει το υπολογιστικό σύστημα, 3MB. Η αλλαγή αυτή είχε ως στόχο να μη φορτώνεται ολόκληρο το frame από την μνήμη για τον υπολογισμό των blocks και κατ' επέκταση των macroblock που χρειάζονται για την διαδικασία του intra prediction. Η μείωση χρήσης μνήμης είναι πολύ σημαντική διότι πλέον δε χρειάζεται να φορτωθεί όλο το καρέ, που αντιστοιχεί σε χρήση μνήμης 0.7 MB υπολογίζοντας και την τοπική αποθήκη στην οποία αποθηκεύεται το τρέχων macroblock και αντιστοιχεί 512 Bytes για τις τιμές της φωτεινότητας και 0.35 MB και 128 bytes για τις τιμές του chroma . Με την χρήση τοπικών αποθηκών, χρειάζονται μόνο 512 bytes για το τρέχων macroblock καθώς 1.4 KB για την αποθήκη που κρατάει τα pixel της τελευταίας γραμμής για κάθε macroblock και 1.4 KB για το chroma. Επιπλέον 32 bytes για την αποθήκη που κρατάει τα κάθετα pixels και 6 byte για τα διαγώνια, καθώς και 32 bytes για το chroma. Ο αναγνώστης μπορεί εύκολα να διαπιστώσει ότι η μείωση χρήσης μνήμης είναι τεράστια. Μια άλλη σημαντική ιδιότητα χρήσης τοπικών buffers είναι πολύ εύκολη η παραλληλοποίηση της διαδικασίας του decode αφού λύνονται όλες οι εξαρτήσεις με την χρήση των τοπικών αποθηκών. Σε μια μελλοντική δουλειά στο ίδιο πρότυπο θα μπορούσε πολύ εύκολα να παραλληλοποιηθεί η διαδικασία .

5.3 Deblocking filter με χρήση τοπικών χώρων αποθήκευσης

5.3.1 Περιγραφή της διαδικασίας

Στο κλασσικό πρότυπο του H.264 μετά την ολοκλήρωση της αποκωδικοποίησης ολόκληρου του καρέ χρησιμοποιούνται φίλτρα για να εξομαλύνουν τις ακμές όλων των 4x4 block. Επιπλέον στον SVC οι αποκωδικοποιημένες τιμές ενός καρέ, χρησιμοποιούνται ως αναφορά για να δημιουργηθεί το ανώτερο ιεραρχικά επίπεδο στην χωρική κλιμάκωση. Αυτές οι τιμές ενδέχεται να περάσουν από την διαδικασία του φιλτραρίσματος πριν την διαδικασία δημιουργίας του ανώτερου ιεραρχικά επίπεδου. Επομένως όταν τελειώσει η διαδικασία της αποκωδικοποίησης όλο το καρέ φιλτράρεται. Επιπλέον πριν αρχίσει η διαδικασία του upsampling για το νέο καρέ φιλτράρονται τα αποκωδικοποιημένα pixel του προηγούμενου επιπέδου με βάση όμως νέες συνθήκες, για τον λόγο αυτό δε μπορούν χρησιμοποιηθούν τα ήδη φιλτραρισμένα pixel. Μια σημαντική παρατήρηση είναι ότι το φιλτράρισμα μπορεί να γίνει σε 2 επαναλήψεις. Σε αυτή την περίπτωση οι εσωτερικές ακμές φιλτράρονται σε διαφορετική επανάληψη από ότι οι εσωτερικές. Σύμφωνα με το πρότυπο η διαδικασία του deblocking μπορεί να γίνει μόλις τελειώνει η αποκωδικοποίηση ενός macroblock . Βέβαια υπάρχουν σαφής περιορισμοί. Κατά την διαδικασία του deblocking ένα intra macroblock δε αλλάζει μόνο τα δικά του pixels αλλά τα pixels των 3 τελευταίων στηλών και των αντίστοιχων των 3 τελευταίων γραμμών, των macroblock που βρίσκονται στις θέσεις πάνω από το τρέχων και αριστερά από το τρέχων. Όμως για το intra prediction οι τιμές που πρέπει να χρησιμοποιηθούν είναι αυτές που προέρχονται από την διαδικασία της αποκωδικοποίησης χωρίς να υποστούν φιλτράρισμα.. Για τον λόγο αυτό για να μπορέσει να λειτουργήσει η διαδικασία του deblocking σύμφωνα με τα πρότυπα του standard θα πρέπει να χρησιμοποιηθούν τοπικές αποθήκες . Θα πρέπει να χρησιμοποιηθεί μια αποθήκη μεγέθους $\mathbf{4} * \boldsymbol{iWidth}$ όπου 4 το πλήθος των γραμμών του προηγούμενου macroblock για το οριζόντιο φιλτράρισμα των τιμών της φωτεινότητας και μια αποθήκη μεγέθους **4*16** όπου 4 το πλήθος των στηλών για το κάθετο φιλτράρισμα .Αντίστοιχα για τις τιμές του chroma πρέπει η αποθήκη για το οριζόντιο φιλτράρισμα να έχει μέγεθος $\mathbf{2} * (\boldsymbol{iWidth} \gg \mathbf{1})$ και για το κάθετο φιλτράρισμα **2*8**. Επιπλέον για την διαδικασία αυτή εκτός από την μέθοδο του παραθύρου ολίσθησης που περιγράφηκε στην ενότητα 5.2 εισάγεται και η έννοια του ολισθημένου macroblock κατά x και y που θα αναλυθεί στην συνέχεια. Επιπροσθέτως, στην διαδικασία του deblocking δε περνάει σαν παράμετρος ολόκληρο το frame

αλλά μόνο το τρέχων macroblock και οι 2 τοπικοί buffers . Επομένως πλέον χρησιμοποιούνται 2 συναρτήσεις για κάθε κατηγορία φιλτραρίσματος, δηλαδή κάθετου και οριζόντιου. Η μία φιλτράρει τις εξωτερικές ακμές αλλάζοντας κατ' επέκταση και τις αντίστοιχες τιμές του προηγούμενου macroblock, οι οποίες όμως πλέον βρίσκονται αποθηκευμένες στον τοπικό buffer άρα αλλάζει τις τιμές στο τοπικό buffer χωρίς να επηρεάζει την διαδικασία του intra prediction. Η άλλη συνάρτηση φιλτράρει τις εσωτερικές ακμές κάθε block 4x4. Στην αρχή της διαδικασίας ελέγχεται αν το τρέχων frame πρέπει να υποστεί φιλτράρισμα. Αν δε ισχύει τότε γράφεται στο εξωτερικό buffer είτε για απεικόνιση είτε για να χρησιμοποιηθεί από το επόμενο layer. Σε αντίθετη περίπτωση το τρέχων macroblock που έχει ήδη αποκωδικοποιηθεί φιλτράρεται. Όταν τελειώσει η διαδικασία του φιλτραρίσματος τα αποτελέσματα γράφονται στις κατάλληλες θέσεις των τοπικών buffers για να χρησιμοποιηθούν για το επόμενο macroblock και δείκτης μέσα στην τοπική μνήμη για το οριζόντιο φιλτράρισμα μετακινείται κατά 16 ή 8 για το luma και chroma αντίστοιχα. Όταν τελειώσει η διαδικασία, από το τρέχων macroblock αντιγράφονται σε εξωτερική μνήμη που περιέχει τα αποκωδικοποιημένα macroblocks του τρέχοντος καρέ, οι 12 πρώτες γραμμές και στήλες για τις τιμές της φωτεινότητας και οι 8 πρώτες γραμμές και στήλες για τιμές του χρώματος. Οι υπολειπόμενες στήλες και γραμμές αντιγράφονται από τους τοπικές θέσεις μνήμης. Τελικά στο εξωτερικό buffer αντιγράφεται ένα macroblock 16x16 το οποίο όμως δε είναι το αρχικό. Στην πραγματικότητα είναι νέο ένα macroblock που έχει υποστεί ολίσθηση κατά 4 θέσεις αριστερά και 4 θέσεις προς τα πάνω. Δηλαδή μετά την ολοκλήρωση του φιλτραρίσματος αντιγράφονται 12 γραμμές και 4 στήλες για το αριστερά συμπληρώνοντας το αριστερά macroblock και 4 γραμμές και 16 στήλες για το πάνω από το τρέχων macroblock. Οι 4 γραμμές που δε αντιγράφονται από την τοπική μνήμη που αποθηκεύονται τα pixels για το αριστερό macroblock, συμπληρώνονται από το οριζόντιο φιλτράρισμα που εκτελείται τελευταίο. Εδώ πρέπει να δοθεί προσοχή στις οριακές περιπτώσεις. Τα δεξιότερα macroblocks πρέπει να αντιγράψουν στο εξωτερικό buffer τις 12 πρώτες γραμμές και 16 στήλες, οι υπόλοιπες γραμμές αντιγράφονται από την τοπική προσωρινή μνήμη. Επιπλέον για την τελευταία γραμμή ενός καρέ θα πρέπει να αντιγραφούν 16 γραμμές και 12 στήλες. Οι υπόλοιπες στήλες θα αντιγραφούν από την τοπική προσωρινή μνήμη. Το τελευταίο macroblock αντιγράφεται ολόκληρο στην εξωτερική μνήμη. Στο επόμενο σχήμα φαίνεται ποιες περιοχές αντιγράφονται στην εξωτερική μνήμη και ποιές στις προσωρινές θέσεις αποθήκευσης.


Horizontal buffer
Vertical
buffer

Στο παραπάνω σχήμα φαίνονται οι περιοχές που αντιγράφονται όταν τελειώσει ένα macroblock την αποκωδικοποίησή .Με μπλε χρώμα εμφανίζονται οι τιμές των pixels που πρέπει να αντιγραφούν από τον horizontal buffer στην εξωτερική μνήμη και αντιστοιχούν στις 4 τελευταίες γραμμές του macroblock που βρίσκεται στην θέση πάνω από το τρέχον .Με κόκκινο χρώμα εμφανίζονται οι τιμές των pixels που πρέπει να αντιγραφούν από τον vertical buffer στην εξωτερική μνήμη και αντιστοιχούν στις τελευταίες στήλες του αριστερά macroblock. Αντιγράφονται μόνο οι 12 πρώτες γραμμές, οι υπόλοιπες 4 θα αντιγραφούν από τον horizontal buffer όταν αποκωδικοποιηθεί το macroblock που βρίσκεται στην θέση κάτω από αυτό. Οι γραμμές και οι στήλες που είναι γραμμοσκιασμένες αντιγράφονται στις τοπικές θέσεις μνήμης για να χρησιμοποιηθούν από τα κατάλληλα macroblocks και δε αντιγράφονται στον εξωτερικό buffer.

Η ίδια διαδικασία επαναλαμβάνεται και κατά την εκτέλεση του μηχανισμού Inter Layer Intra Prediction, που τα pixel του προηγούμενου επιπέδου φιλτράρονται πριν την χωρική κλιμάκωση

5.3 .1Σύνοψη και συμπεράσματα για την διαδικασία του deblocking με χρήση τοπικής μνήμης

Η χρήση του τοπικής μνήμης για την εκτέλεση της διαδικασίας του φιλτραρίσματος δε έδωσε νέα βελτίωση στο κώδικα. Στόχος της βελτίωσης αυτής ήταν η χρησιμοποίηση της κρυφής μνήμης για την εκτέλεση του deblocking. Στον αρχικό κώδικα για την εκτέλεση του deblocking έπρεπε να έρθει από την μνήμη όλο το frame ,που για την συγκεκριμένη ανάλυση έχει μέγεθος 0.7 MB, το οποίο χωράει στην κρυφή μνήμη του συστήματος που όπως ήδη έχει αναφερθεί είναι 3MB. Όμως η αρχιτεκτονική αυτή αλλαγή όπως και η αλλαγή του intra prediction είναι πολύ σημαντική γιατί η διαδικασία του φιλτραρίσματος μπορεί να παραλληλοποιηθεί και να εκτελείται ταυτόχρονα με την διαδικασία της αποκωδικοποίησης .

Επιπλέον οι πόροι που καταναλώνει η συγκεκριμένη διαδικασία είναι ελάχιστοι αφού χρειάζεται μόνο 2.85K για τον horizontal buffer και 64 bytes για τον vertical bytes για τις τιμές του luma.Για το χρώμα χρειάζεται 1.4 K και 32 bytes για τον horizontal και τον vertical buffer αντίστοιχα.

5.4 Βελτιστοποίηση του αλγορίθμου υπολογισμού των BS για την διαδικασία του φιλτραρίσματος

5.4.1Εισαγωγη

Στην ενότητα αυτή θα γίνει αναφορά στην επόμενη μεγάλη αρχιτεκτονική αλλαγή που έγινε. Στον αρχικό κώδικα ο αλγόριθμος υπολογισμού των τιμών της μεταβλητής BS (Boundary Strength) γίνεται κατά την διάρκεια εκτέλεσης της διαδικασίας φιλτραρίσματος και για κάθε block 4x4. Όμως ο υπολογισμός των BS με αυτόν τρόπο δε είναι ο βέλτιστος δυνατός, διότι μόνο ένα υποσύνολο τού δέντρου απόφασης μπορεί να αλλάξει τιμές στις τιμές του boundary strength στα όρια 2 block 4x4. Σε όλες τις άλλες περιπτώσεις κατά μήκος ενός macroblock το BS παραμένει ίδιο.

5.4.2 Παρουσίαση του δέντρου απόφασης του BS κατά την διάρκεια του φιλτραρίσματος

Στην ενότητα αυτή θα περιγραφεί το δέντρο λήψης απόφασης για την τιμή του BS που θα πρέπει να έχει το 4x4 block για να αποκωδικοποιηθεί σωστά. Η λογική του τρόπου υπολογισμού της σωστής τιμής του BS για τα 4x4 blocks ,είναι η εξής : Ελέγχονται μια προς μια οι περιπτώσεις που πρέπει να ισχύων για να μπορεί το block 4x4 να λάβει μια τιμή από το 0 - 4. Κάθε φορά που δε ισχύει συνεχίζει ο αλγόριθμός στην επόμενη συνθήκη. Στους παρακάτω πίνακα παρουσιάζονται όλες οι δυνατές περιπτώσεις και δίπλα η τιμή του BS για την συγκεκριμένη περίπτωση. Αν δε ισχύει μια συνθήκη τότε ελέγχεται η επόμενη από πάνω προς τα κάτω.

- Παρουσίαση συνθηκών για το οριζόντιο φιλτράρισμα. των εσωτερικών ακμών

Συνθήκη	**BS**

Διαδικασία του deblocking disable ή φιλτράρισμα δεύτερης επανάληψης ή τρέχων macroblock όχι intra και εκτέλεση inter prediction	0
iFilterIdc ==1 ή δε υπάρχει macroblock πάνω από το τρέχων	0
Αν το πάνω από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0
Αν το πάνω είναι διαθέσιμο τότε αν είναι φιλτράρισμα δεύτερης επανάληψης	0
Αν το τρέχων macroblock δε είναι intra και εκτελείται inter prediction	0
Αν bSpatialScalableFlag == true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block ή το απο πάνω είναι κωδικοποιημένο σαν 4x4	1
Αλλιώς αν είναι bSpatialScalableFlag= true και ο τύπος του macroblock είναι Intra_BL αλλά δε ισχύει η παραπάνω συνθήκη	0
Αν ένα macroblock είναι intra	3
Αν το τρέχων ή το πάνω block 4x4 έχει υπόλοιπα	2
Αν το τρέχων ή το πάνω block 4x4 έχει κωδικοποιηθεί σαν DQId0	2
Αν το τρέχων macroblock είναι P και το κάθετο 4x4 block έχει διαφορετικό block	1

αναφοράς από το block που βρίσκεται πάνω από αυτό	
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι P και δεν ισχύουν οι παραπάνω συνθήκες	0
Αν το τρέχων macroblock είναι B τότε αν ο αριθμός των καρέ αναφοράς είναι διαφορετικός	1
Αν το τρέχων macroblock είναι B και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι B και δεν ισχύουν οι παραπάνω συνθήκες	0

- Παρουσίαση συνθηκών για το κάθετο φιλτράρισμα των εσωτερικών ακμών

Συνθήκη	**BS**
Διαδικασία του deblocking disable ή φιλτράρισμα δεύτερης επανάληψης ή τρέχων macroblock όχι intra και εκτέλεση inter prediction	0
iFilterIdc ==1 ή δε υπάρχει macroblock αριστερά από το τρέχων	0
Αν το αριστερά από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή	0

iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	
Αν το αριστερά είναι διαθέσιμο τότε αν είναι φιλτράρισμα δεύτερης επανάληψης	0
Αν το τρέχων macroblock δε είναι intra και εκτελείται inter prediction	0
Αν bSpatialScalableFlag == true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block ή το απο αριστερά είναι κωδικοποιημένο σαν 4x4	1
Αλλιώς αν είναι bSpatialScalableFlag= true και ο τύπος του macroblock είναι Intra_BL αλλά δε ισχύει η παραπάνω συνθήκη	0
Αν ένα macroblock είναι intra	3
Αν το τρέχων ή το αριστερά block 4x4 έχει υπόλοιπα	2
Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί σαν DQId0	2
Αν το τρέχων macroblock είναι P και το αριστερά 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται πάνω από αυτό	1
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του αριστερά block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι P και δεν ισχύουν οι παραπάνω συνθήκες	0

Αν το τρέχων macroblock είναι B τότε αν ο αριθμός των καρέ αναφοράς είναι διαφορετικός	1
Αν το τρέχων macroblock είναι B και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του αριστερού block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι B και δεν ισχύουν οι παραπάνω συνθήκες	0

- Παρουσίαση συνθηκών για το οριζόντιο φιλτράρισμα εξωτερικών ακμών

Συνθήκη	**BS**
Αν εκτελείται ο μηχανισμός του intra prediction και το τρέχων block ή το πάνω από αυτό είναι intra	0
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων ή το πανω από το τρέχων block είναι Intra_BL τότε αν το τρέχων είναι intra αλλά οχί Intra_BL ή αντίστοιχα το πάνω απο αυτό είναι intra αλλά όχι intra block	4
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το πανω από το τρέχων block είναι Intra_BL τότε αν ένα απο τα δύο blocks είναι κωδικοποιημένο σαν 4x4 block	1
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το πανω από το τρέχων block είναι Intra_BL τότε ανκαι τα 2 δεν είναι κωδικοποιημένα σαν 4x4 block	0
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων ή το πάνω από το τρέχων block είναι Intra_BL και ένα απο τα δύο είναι κωδικοποιημένο με υπόλοιπα	2
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το πάνω από το τρέχων block δεν είναι Intra_BL και ένα απο τα δύο είναι κωδικοποιημένο με υπόλοιπα	1
Αν το τρέχων macroblock είναι intra ή το πάνω από αυτό είναι intra	4
Αν το τρέχων ή το πάνω block 4x4 έχει κωδικοποιηθεί με υπόλοιπα	2
Αν το τρέχων ή το πάνω block 4x4 έχει κωδικοποιηθεί σαν DQId0	2

Αν το τρέχων macroblock είναι P και το κάθετο 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται πάνω από αυτό	1
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι P και δεν ισχύουν οι παραπάνω συνθήκες	0
Αν το τρέχων macroblock είναι B τότε αν ο αριθμός των καρέ αναφοράς είναι διαφορετικός	1
Αν το τρέχων macroblock είναι B και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι B και δεν ισχύουν οι παραπάνω συνθήκες	0

- Παρουσίαση των συνθηκών για το κάθετο φιλτράρισμα εξωτερικών ακμών

Συνθήκη	**BS**
Αν εκτελείται ο μηχανισμός του intra prediction και το τρέχων block ή το αριστερά από αυτό είναι intra	0
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων ή το αριστερά από το τρέχων block είναι Intra_BL τότε αν το τρέχων είναι intra αλλά οχί Intra_BL ή αντίστοιχα το αριστερά απο αυτό είναι intra αλλά όχι intra block	4
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το αριστερά από το τρέχων block είναι Intra_BL τότε αν ένα απο τα δύο blocks είναι κωδικοποιημένο σαν 4x4 block	1
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το αριστερά από το τρέχων block είναι Intra_BL τότε αν και τα 2 δεν είναι κωδικοποιημένα σαν 4x4 block	0
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων ή το αριστερά από το τρέχων block είναι Intra_BL και ένα απο τα δύο είναι κωδικοποιημένο με υπόλοιπα	2
Αν η μεταβλητή bSpatialScalableFlag και το τρέχων και το αριστερά από το τρέχων block δεν είναι Intra_BL και ένα απο τα δύο είναι κωδικοποιημένο με υπόλοιπα	1
Αν το τρέχων macroblock είναι intra ή το αριστερά από αυτό είναι intra	4
Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί με υπόλοιπα	2

Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί σαν DQId0	2
Αν το τρέχων macroblock είναι P και το κάθετο 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται αριστερά από αυτό	1
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του αριστερά block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι P και δεν ισχύουν οι παραπάνω συνθήκες	0
Αν το τρέχων macroblock είναι B τότε αν ο αριθμός των καρέ αναφοράς είναι διαφορετικός	1
Αν το τρέχων macroblock είναι B και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του αριστερά block είναι μεγαλύτερη του 4	1
Αν το τρέχων macroblock είναι B και δεν ισχύουν οι παραπάνω συνθήκες	0

Στο πίνακα που ακολουθεί δίνεται η επεξήγηση των διαφορετικών τιμών της μεταβλητής iFilterIdc

Τιμή	Εξήγηση
0	Όλες οι τιμές του luma και του chroma φιλτράρονται
1	Η διαδικασία του φιλτραρίσματος είναι ανενεργή
2	Φιλτράρονται τα blocks που δε ανήκουν στα όρια του slice και στην συνέχεια αυτά που ανήκουν στα όρια του slice
3	Γίνεται φιλτράρισμα σε 2 φάσεις. Αρχικά φιλτράρονται τα blocks που δε ανήκουν στα όρια του slice και στην συνέχεια αυτά που ανήκουν στα όρια του slice
4	Γίνεται φιλτράρισμα μόνο για τις τιμές της φωτεινότητας .Το φιλτράρισμα για το

	χρώμα είναι απενεργοποιημένο
5	Γίνεται φιλτράρισμα μόνο για τις τιμές της φωτεινότητας που δε βρίσκονται στα όρια του slice .Το φιλτράρισμα για το χρώμα είναι απενεργοποιημένο
6	Το φιλτράρισμα για το chroma είναι απενεργοποιημένο και το φιλτράρισμα των τιμών της φωτεινότητας γίνεται όπως περιγράφτηκε για την τιμή 3

Επιπλέον η μεταβλητή isAboveMbExisting είναι true όταν το macroblock δε έχει κάθετη συντεταγμένη ίση με 0 και δε έχει κάθετη συντεταγμένη =1 και ταυτόχρονα είναι τύπου field. Επίσης η μεταβλητή isLeftMbExisting είναι true οταν όταν ή οριζόντια συντεταγμένη του είναι διαφορη του 0 macroblock είναι true. Τέλος η μεταβλητή isLeftMbAvailable όταν το τρέχων macroblock και το αριστερά, αν φυσικά υφίσταται, ανήκουν στο ίδιο slice.

Παρατηρώντας τους παραπάνω πίνακες εύκολα ο αναγνώστης μπορεί να διαπιστώσει τεχνικές βελτιστοποίησεις τους. Για παράδειγμα η μεταβλητή iFilterIdc ορίζεται στο encoder και δε αλλάζει σε όλη την διάρκεια της αποκωδικοποίησης. Επομένως αναφέρεται σε ολόκληρο το καρέ και κατ' επεκταση σε ολόκληρο macroblock, αρα δε χρειάζεται να γίνετε έλεγχος κάθε σε κάθε 4x4 block. Επιπλεόν η μεταβλητή που ορίζει αν θα γίνει inter prediction και αυτή ισχύει για ολόκληρο macroblock. Επιπλέον δε χρειάζεται να ελέγχεται κατα την διάρκεια της διαδικασίας της αποκωδικοποίησης ενός layer, αφού η τιμή της είναι false. Ιδιαίτερη βάση πρέπει να δωθεί και στο έλεγχο αν ένα macroblock είναι intra , αφού και αυτό ισχύει για όλο το macroblock. Όμως οι έλεγχοι που πραγματικά κοστίζουν περισσότερο είναι αυτοί που γίνονται για τα inter frames. Αρχικά μόνο σε αυτή την περίπτωση πρέπει να γίνει έλεγχος για το αν η προσωρινή μνήμη με τα υπόλοιπα περιέχει υπόλοιπα διάφορα του μηδενός. Επιπλέον στον αρχικό κώδικα ανεξάρτητα απο τις υποδιαιρέσης ενός macroblock σε επιμέρους block (ενότητα 4.3), για τα P και B frames ελέγχονται για όλα τα εσωτερικά 4x4 block οι συνθήκες που πρέπει να πληρούνται για να είναι το BS =1. Ο τρόπος που γίνονται οι έλεγχοι είναι μη βέλτιστος γιατί πχ αν ένα macroblock τύπου inter δε υποδιαιρείται σε μικρότερα blocks, προφανώς υπάρχει ένα motion vector σε όλο το macroblock, άρα τα εσωτερικά 4x4 blocks έχουν το ίδιο σύνολο

εικόνων αναφοράς και διαφορά συντεταγμένων 0, άρα οι έλεγχοι που γίνονται είναι περιττοί .Στην συνέχεια θα δοθεί αναλυτική περιγραφή για κάθε τύπο block.

5.4.3 Περιγραφή των αλλαγών που έλαβαν χώρα

Στην ενότητα θα περιγραφεί αναλυτικά ο τρόπος που έγιναν οι αρχιτεκτονικές αλλαγές. Αρχικά ο υπολογισμός των BS μεταφέρεται στα εξής σημεία. Στο μηχανισμό του inter layer prediction πριν την διαδικασία του deblocking. Μετά την διαδικασία του inverse transform για τα intra macroblocks και μετά την διαδικασία του inverse transform για τα inter macroblocks .Συνεχίζοντας θα αναλυθούν και οι 3 περιπτώσεις. Στους πίνακες που ακολουθούν θα παρουσιαστεί το νέο δέντρο απόφασης και αν χρειάζεται να εφαρμοστεί σε ολόκληρο το macroblock ή ανά block

- Υπολογισμός BS για το deblocking του μηχανισμού inter-layer intra prediction

Αρχικά πρέπει να παρατηρηθεί ότι η αποθήκη των υπολοίπων δε χρειάζεται, και για τον λόγο αυτό δε ορίζεται, άρα δε έχει δεσμευτεί χώρος. Επιπλέον η συνθήκη που ελέγχει αν εκτελείται inter layer prediction είναι πάντα αληθής. Τα νέα δέντρο απόφασης για τις εσωτερικές και εξωτερικές ακμές παρουσιάζονται στον παρακάτω πίνακα.

- Πινάκας λήψης απόφασης για το οριζόντιο φιλτράρισμα εσωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για ολόκληρο macroblock**
Αν η τιμή του iFilterIdc είναι 1	0	Όχι	Ναί
Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη	0	Όχι	Ναί
Αν δε είναι intra το τρέχων macroblock	0	Όχι	Ναί
Αν bSpatialScalableFlag == true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block ή το απο πάνω είναι κωδικοποιημένο σαν	1	Ναί	Όχι

4x4			
Αν bSpatialScalabelFlag==true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block και το block που βρίσκεται στην θέση πάνω από αυτό δεν είναι κωδικοποιημένο σαν 4x4	0	Ναί	Όχι
Αν δε ισχύει καμία από τις παραπάνω συνθήκες γίνεται έλεγχος αν είναι Intra	3	Όχι	Ναι
Αλλιώς γίνεται έλεγχος αν αυτό η το πάνω από αυτό είναι κωδικοποιημένο σαν DQId0	2	Ναι	Όχι

- Πίνακας περιγραφής λήψης απόφασης για το οριζόντιο φιλτράρισμα εξωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για ολόκληρο macroblock**
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock πάνω από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί
Αν το πάνω από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0	Όχι	Ναί
Αν το πάνω από το τρέχων macroblock είναι διαθέσιμό αλλά εκτελείται το φιλτράρισμα της δεύτερης επανάληψης	0	Όχι	Ναί
Αν το από πάνω από το τρέχων macroblock δε είναι intra	0	Όχι	Ναί
Αν δε ισχύουν οι παραπάνω συνθήκες τότε αν η μεταβλητή bSpatialScalableFlag έχει τιμή true και ή το τρέχων macroblock ή το πάνω απο αυτό είναι	4	‘Οχι	Ναί

τύπου Intra_BL και όχι και τα 2 ταυτόχρονα			
Σε αντίθετη περίπτωση που και τα 2 macroblocks είναι Intra_BL αν είναι κωδικοποιημένο ένα απο τα 2 σαν 4x4	1	Ναί	Όχι
Σε αντίθετη περίπτωση που και τα 2 macroblocks είναι Intra_BL και δεν είναι κωδικοποιημένο σαν 4x4	0	Ναί	Όχι
Αν ισχύει μόνο ότι η μεταβλητή bSpatialScalableFlag έχει τιμή true και ή το τρέχων macroblock ή το πάνω απο αυτό είναι τύπου Intra_BL	1	Όχι	Ναί
Αν δε ισχύει καμία από τις παραπάνω σχέσεις τότε αν το τρέχων macroblock ή το macroblock που βρίσκεται πάνω απο το τρέχων είναι τύπου Intra	4	Όχι	Ναί
Αν το τρέχων ή το πάνω block 4x4 έχει κωδικοποιηθεί σαν DQId0	2	Ναί	Όχι

- Πίνακας περιγραφής λήψης απόφασης για την διαδικασία κάθετου φιλτραρίσματος εσωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για ολόκληρο macroblock**
Αν η τιμή του iFilterIdc είναι 1	0	΄Οχι	Ναί
Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη	0	΄Οχι	Ναί
Αν δε είναι intra το τρέχων macroblock	0	Όχι	Ναί
Αν bSpatialScalableFlag == true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block ή το αριστερά είναι κωδικοποιημένο σαν 4x4	1	Ναί	Όχι

Αν bSpatialScalabelFlag==true και ο τύπος του macroblock είναι Intra_BL τότε αν το τρέχων block και το block που βρίσκεται στην θέση αριστερά από αυτό δεν είναι κωδικοποιημένο σαν 4x4	0	Ναί	Όχι
Αν δε ισχύει καμία από τις παραπάνω συνθήκες γίνεται έλεγχος αν είναι Intra	3	Όχι	Ναι
Αλλιώς γίνεται έλεγχος αν αυτό η το αριστερό από αυτό είναι κωδικοποιημένο σαν DQId0	2	Ναι	Όχι

- Πίνακας περιγραφής λήψης απόφασης για την διαδικασία κάθετου φιλτραρίσματος εξωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για ολόκληρο macroblock**
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock αριστερά από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί
Αν το αριστερά από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0	Όχι	Ναί
Αν το αριστερά από το τρέχων macroblock είναι διαθέσιμό αλλά εκτελείται το φιλτράρισμα της δεύτερης επανάληψης	0	Όχι	Ναί
Αν το αριστερά από το τρέχων macroblock δε είναι intra	0	Όχι	Ναί
Αν δε ισχύουν οι παραπάνω συνθήκες τότε αν η μεταβλητή bSpatialScalableFlag έχει τιμή true και	4	‘Οχι	Ναί

ή το τρέχων macroblock ή το αριστερά απο αυτό είναι τύπου Intra_BL και όχι και τα 2 ταυτόχρονα			
Σε αντίθετη περίπτωση που και τα 2 macroblocks είναι Intra_BL αν είναι κωδικοποιημένο ένα από τα 2 σαν 4x4	1	Ναί	Όχι
Σε αντίθετη περίπτωση που και τα 2 macroblocks είναι Intra_BL και δεν είναι κωδικοποιημένο σαν 4x4	0	Ναί	Όχι
Αν ισχύει μόνο ότι η μεταβλητή bSpatialScalableFlag έχει τιμή true και ή το τρέχων macroblock ή το αριστερά απο αυτό είναι τύπου Intra_BL	1	Όχι	Ναί
Αν δε ισχύει καμία από τις παραπάνω σχέσεις τότε αν το τρέχων macroblock ή το macroblock που βρίσκεται αριστερά από το τρέχων είναι τύπου Intra	4	Όχι	Ναί
Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί σαν DQId0	2	Ναί	Όχι

Από τους παραπάνω πίνακες φαίνεται ότι σε ελάχιστες περιπτώσεις χρειάζεται να εξεταστεί κάθε block 4x4 ξεχωριστά και στις περισσότερες των περιπτώσεων η τιμή του BS υπολογίζεται για όλο το macroblock.

- Υπολογισμός BS για το deblocking των pixels που προκύπτουν μετά την αποκωδικοποίηση intra macroblocks

Σε αυτήν την περίπτωση για το τρέχων macroblock δε χρειάζεται να γίνει έλεγχος αν έχει κωδικοποιηθεί με υπόλοιπα διότι δε υφίσταται στην Intra αποκωδικοποίηση. Αλλά επειδή υπάρχει περίπτωση τα εξαρτώμενα macroblocks από αυτό να είναι κωδικοποιημένα σαν inter πρέπει να γίνει έλεγχος για αυτά. Επιπλέον μεταβλητή που ελέγχει αν εκτελείται inter-prediction προφανώς είναι false.

- Πίνακας περιγραφής λήψης απόφασης για τα εσωτερικά blocks για το οριζόντιο φιλτράρισμα

Συνθήκη	BS	Υπολογισμός για κάθε 4x4 block	Υπολογισμός για όλο το macroblock
Αν η τιμή του iFilterIdc είναι 1	0	Όχι	Ναί
Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη	0	Όχι	Ναί
Αν δε ισχύουν τα παραπάνω τότε αν bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock είναι τύπου IntraBL τότε αν το τρέχων block ή το πάνω block είναι κωδικοποιημένο σαν 4x4	1	Ναί	Όχι
Αν bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock είναι τύπου IntraBL τότε αν το τρέχων block και το πάνω block δεν είναι κωδικοποιημένο σαν 4x4	0	Ναι	Όχι
Αν δε ισχύει ότι bSpatialScalabilityFlag είναι αληθής ή το τρέχων macroblock είναι τύπου IntraBL τότε η τελευταία περίπτωση είναι να είναι intra	3	Όχι	Ναι

- Πίνακας περιγραφής λήψης απόφασης για τα εξωτερικά blocks για το οριζόντιο φιλτράρισμα

Συνθήκη	BS	Υπολογισμός για κάθε 4x4 block	Υπολογισμός για όλο το macroblock
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock πάνω από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί
Αν το πάνω από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα	0	Όχι	Ναί

δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης			
Αν η μεταβλητή bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock ή το πάνω απο αυτό είναι Intra_BL, αλλά μόνο ένα απο τα δύο	4	Όχι	Ναι
Αν η μεταβλητή bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock και το πάνω απο αυτό είναι Intra_BL τότε αν ένα απο τα 2 είναι κωδικοποιημένο σαν 4x4	1	Ναί	Όχι
Αν η μεταβλητή bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock και το πάνω απο αυτό είναι Intra_BL τότε αν και τα 2 δε είναι κωδικοποιημένα σαν 4x4	0	Ναί	Όχι
Αν η μεταβλητή bSpatialScalability είναι αληθής και το τρέχων macroblock είναι Intra_BL και δε ισχύει καμία από τις παραπάνω περιπτώσεις ελέγχεται αν το macroblock που βρίσκεται πάνω από το αρχικό δε είναι intra και έχει κωδικοποιηθεί με υπόλοιπα	2	Ναί	Όχι
Αν το macroblock που βρίσκεται πάνω από το αρχικό δε είναι intra και δεν έχει κωδικοποιηθεί με υπόλοιπα	1	Ναί	Όχι
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις αλλά η μεταβλητή bSpatialScalability είναι αληθής και το τρέχων ή το πάνω από αυτό macroblock είναι Intra_BL	1	Όχι	Ναί
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις	4	Όχι	Ναί

- Πίνακας περιγραφής λήψης απόφασης για τα εσωτερικά blocks για το κάθετο ο φιλτράρισμα

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για όλο το macroblock**
Αν η τιμή του iFilterIdc είναι 1	0	Όχι	Ναί

Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη	0	Όχι	Ναί
Αν δε ισχύουν τα παραπάνω τότε αν bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock είναι τύπου IntraBL τότε αν το τρέχων block ή το αριστερά block είναι κωδικοποιημένο σαν 4x4	1	Ναί	Όχι
Αν bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock είναι τύπου IntraBL τότε αν το τρέχων block και το αριστερά block δεν είναι κωδικοποιημένο σαν 4x4	0	Ναι	Όχι
Αν δε ισχύει ότι bSpatialScalabilityFlag είναι αληθής ή το τρέχων macroblock είναι τύπου IntraBL τότε η τελευταία περίπτωση είναι να είναι intra	3	Όχι	Ναι

- Πίνακας περιγραφής λήψης απόφασης για τα εξωτερικά blocks για το κάθετο φιλτράρισμα

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για όλο το macroblock**
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock αριστερά από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί
Αν το πάνω από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0	Όχι	Ναί
Αν η μεταβλητή bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock ή το αριστερά απο αυτό είναι Intra_BL, αλλά μόνο ένα απο τα δύο	4	Όχι	Ναι
Αν η μεταβλητή bSpatialScalabilityFlag είναι	1	Ναί	Όχι

αληθής και το τρέχων macroblock και το αριστερά απο αυτό είναι Intra_BL τότε αν ένα απο τα 2 είναι κωδικοποιημένο σαν 4x4			
Αν η μεταβλητή bSpatialScalabilityFlag είναι αληθής και το τρέχων macroblock και το αριστερά απο αυτό είναι Intra_BL τότε αν και τα 2 δε είναι κωδικοποιημένα σαν 4x4	0	Ναί	Όχι
Αν η μεταβλητή bSpatialScalability είναι αληθής και το τρέχων macroblock είναι Intra_BL και δε ισχύει καμία από τις παραπάνω περιπτώσεις ελέγχεται αν το macroblock που βρίσκεται αριστερά από το αρχικό δε είναι intra και έχει κωδικοποιηθεί με υπόλοιπα	2	Ναί	Όχι
Αν το macroblock που βρίσκεται αριστερά από το αρχικό δε είναι intra και δεν έχει κωδικοποιηθεί με υπόλοιπα	1	Ναί	Όχι
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις αλλά η μεταβλητή bSpatialScalability είναι αληθής και το τρέχων ή το αριστερά από αυτό macroblock είναι Intra_BL	1	Όχι	Ναί
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις	4	Όχι	Ναί

Και για αυτήν περίπτωση οι έλεγχοι που γίνονται είναι πολύ λιγότεροι σε σχέση με την αρχική έκδοση του κώδικα.

- Υπολογισμός του BS για inter macroblocks

  Σε αυτήν την περίπτωση βελτιστοποιείται κυρίως ο τρόπος υπολογισμού των BS που βασίζονται στα motion vectors για τα εσωτερικά blocks. Στους πίνακες που θα παρουσιαστούν δε θα δοθούν οι περιγραφές για το τρόπο υπολογισμού των BS με χρήση των motion vector θα δοθεί αναλυτική περιγραφή για αυτήν την περίπτωση.

- Πίνακας λήψης απόφασης για οριζόντιο filtering εσωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για όλο το macroblock**

Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Να
Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη	0	Όχι	Ναί
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις τότε αν το τρέχων macroblock ή το πάνω είναι κωδικοποιημένο με υπόλοιπα	2	Ναί	Όχι
Αλλιώς αν έχει κωδικοποιηθεί το τρέχων η το πάνω από το τρέχων σαν DQId0	2		
Σε κάθε άλλη περίπτωση χρήση των motion vectors	0 ή 1	Ναί/Όχι	Όχι/Ναί

- Πίνακας λήψης απόφασης για κάθετο filtering εσωτερικών blocks

Συνθήκη
Αν η τιμή του IfilterIdc είναι 1
Αν το φιλτράρισμα λαμβάνει χώρα στην δεύτερη επανάληψη
Αν δε ισχύει καμία από τις παραπάνω περιπτώσεις τότε αν το τρέχων macroblock ή το αριστερό είναι κωδικοποιημένο με υπόλοιπα
Αλλιώς αν έχει κωδικοποιηθεί το τρέχων η το αριστερό από το τρέχων σαν DQId0
Σε κάθε άλλη περίπτωση χρήση των motion vectors

- Πίνακας με λήψης απόφασης για οριζόντιο filtering εξωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για όλο το macroblock**
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock πάνω από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί

Αν το πάνω από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0	Όχι	Ναί
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται πάνω απο το τρέχων είναι Intra τότε αν δε έχει τύπο IBL	4	Όχι	Ναί
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται πάνω απο το τρέχων είναι intra και δε ισχύει η προηγούμενη συνθήκη τότε αν το τρέχων είναι κωδικοποιημένο με υπόλοιπα	2	Ναί	Όχι
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται πάνω από το τρέχων είναι intra και δε ισχύουν οι προηγούμενες συνθήκες	1	Ναί	Όχι
Αν δεν ισχύουν όλες οι προηγούμενες συνθήκες και το macroblock που βρίσκεται πάνω από το τρέχων είναι intra αλλά όχι IBL	4	Όχι	Ναί
Αν δεν ισχύει η προηγούμενη συνθήκη αν το τρέχων ή το macroblock που βρίσκεται στην θέση πάνω από το τρέχων έχει κωδικοποιηθεί με υπόλοιπα	2	Ναί	Όχι
Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί σαν DQId0	2	Ναί	Όχι
Αν το τρέχων macroblock είναι P και το κάθετο 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται πάνω από αυτό	1	Ναί	Όχι
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1	Ναί	Όχι

Αν το τρέχων macroblock είναι P και δεν ισχύουν οι παραπάνω συνθήκες	0	Ναί	Όχι
Αν το τρέχων macroblock είναι B και το κάθετο 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται πάνω από αυτό	1	Ναί	Όχι
Αν το τρέχων macroblock είναι B και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του πάνω block είναι μεγαλύτερη του 4	1	Ναί	Όχι
Αν το τρέχων macroblock είναι B και δεν ισχύουν οι παραπάνω συνθήκες	0	Ναί	Όχι

- Πίνακας με λήψης απόφασης για κάθετο filtering εξωτερικών blocks

Συνθήκη	**BS**	**Υπολογισμός για κάθε 4x4 block**	**Υπολογισμός για όλο το macroblock**
Αν η τιμή του IfilterIdc είναι 1	0	Όχι	Ναί
Αν το macroblock αριστερά από το τρέχων δε είναι υπάρχει	0	Όχι	Ναί
Αν το αριστερά από το τρέχων macroblock δε είναι διαθέσιμο τότε αν iFilterIdc =2 ή iFilterIdc =5 ή iFilterIdc=3 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc=6 και φιλτράρισμα πρώτης επανάληψης ή iFilterIdc =0 και φιλτράρισμα δεύτερης επανάληψης ή iFilterIdc =4 και φιλτράρισμα δεύτερης επανάληψης	0	Όχι	Ναί
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται αριστερά απο το τρέχων είναι Intra τότε αν δε έχει τύπο IBL	4	Όχι	Ναί
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται αριστερά απο το τρέχων είναι intra και δε ισχύει η προηγούμενη συνθήκη τότε αν το τρέχων είναι κωδικοποιημένο με	2	Ναί	Όχι

υπόλοιπα			
Αν η μεταβλητή bSpatialScalabilityFlag έχει τιμή true και το macroblock που βρίσκεται αριστερά από το τρέχων είναι intra και δε ισχύουν οι προηγούμενες συνθήκες	1	Ναί	Όχι
Αν δεν ισχύουν όλες οι προηγούμενες συνθήκες και το macroblock που βρίσκεται αριστερά από το τρέχων είναι intra αλλά όχι IBL	4	Όχι	Ναί
Αν δεν ισχύει η προηγούμενη συνθήκη αν το τρέχων ή το macroblock που βρίσκεται στην θέση αριστερά από το τρέχων έχει κωδικοποιηθεί με υπόλοιπα	2	Ναί	Όχι
Αν το τρέχων ή το αριστερά block 4x4 έχει κωδικοποιηθεί σαν DQId0	2	Ναί	Όχι
Αν το τρέχων macroblock είναι P και το κάθετο 4x4 block έχει διαφορετικό block αναφοράς από το block που βρίσκεται αριστερά από αυτό	1	Ναί	Όχι
Αν το τρέχων macroblock είναι P και η διαφορά της οριζόντιας συντεταγμένης ή της κάθετης του motion vector από την αντίστοιχη του αριστερά block είναι μεγαλύτερη του 4	1	Ναί	Όχι

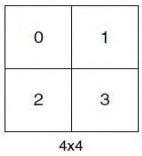
Στην συνέχεια θα δοθεί αναλυτική περιγραφή του τρόπου υπολογισμού των BS με χρήση motion vectors.

Αρχικά δημιουργείτε ένα look-up table το οποίο για κάθε block 8x8 περιέχει τα εσωτερικά 4x4 block. Στην συνέχεια ελέγχεται αν το macroblock χωρίζεται σε blocks ή sub blocks. Αν χωρίζεται σε sub blocks και δημιουργείται ένα βρόγχος επανάληψης που προσπελαύνει όλα τα 8x8 blocks και κατ' επέκταση τα 4x4 block του κάθε 8x8 block. Αρχικά θα εξεταστεί η περίπτωση των sub blocks. Το παρακάτω σχήμα δείχνει πως αριθμούνται τα blocks μέσα σε κάθε 8x8 block. Κάθε 8x8 block παρουσιάζεται με διαφορετικές αποχρώσεις του γκρι ξεκινώντας την αρίθμηση από πάνω αριστερά.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Οι κατηγορίες των sub blocks είναι

1. 4x4



4x4

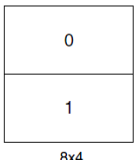
- Υπολογισμός του BS για το οριζόντιο φιλτράρισμα

Για υποδιαιρέσεις 4x4, το τρέχων block και το πάνω από αυτό έχουν διαφορετικά motion vector, άρα θα πρέπει να γίνει έλεγχος για τα καρέ αναφοράς και αν οι διαφορές των συντεταγμένων τους είναι μεγαλύτερες από το ορισμένο κατώφλι.

- Υπολογισμός του BS για το κάθετο φιλτράρισμα

Για υποδιαιρέσεις 4x4 ,το τρέχων block και το αριστερά από αυτό έχουν διαφορετικά motion vector ,άρα θα πρέπει να γίνει έλεγχος για τα καρέ αναφοράς και αν οι διαφορές των συντεταγμένων τους είναι μεγαλύτερες από το ορισμένο κατώφλι.

2. 8x4



8x4

- Υπολογισμός του BS για το οριζόντιο φιλτράρισμα

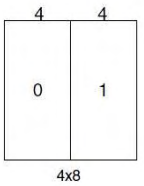
Για 4x4 blocks που βρίσκονται στα όρια 2 8x8 blocks πχ το 8 και το 9 που χρησιμοποιούν για το υπολογισμό του BS τα 4 και 5 έχουν διαφορετικά motion vectors, άρα θα πρέπει να γίνει έλεγχος για τα καρέ αναφοράς και αν οι διαφορές των συντεταγμένων τους είναι μεγαλύτερες από το ορισμένο κατώφλι. Για τα υπόλοιπα

εσωτερικά blocks δε χρειάζεται να υπολογιστούν τα BS και για τα 2 blocks ,αφού ανήκουν στο ίδιο 8x4, επομένως αρκεί να υπολογιστεί το ένα από τα 2 και στο άλλο να ανατεθεί η ίδια τιμή. Για παράδειγμα για το 12 και 13 ,υπολογίζεται το BS για το 12 με βάση το 8,αλλά επειδή το 9 και 13 ανήκουν στα ίδια blocks με τα αντίστοιχα 8 και 12 , δε χρειάζεται να υπολογιστεί το BS για το 13.Άρα το sub block 13 έχει το ίδιο το BS με το 12.

- Υπολογισμός του BS για το κάθετο φιλτράρισμα

Στην περίπτωση αυτή πρέπει να υπολογιστεί το BS μόνο στα κάθετα όρια δυο 8x8 blocks ,πχ για το 2 και το 6 με block αναφοράς το 1 και το 5. Για τα υπόλοιπα blocks είναι 0.

3. 4x8



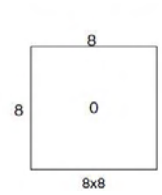
- Υπολογισμός του BS για το οριζόντιο φιλτράρισμα.

Σε αυτή την κατηγορία για το οριζόντιο φιλτράρισμα πρέπει να υπολογιστεί μόνο το BS για τα blocks που βρίσκονται στα οριζόντια όρια 2 8x8 blocks. Για τα υπόλοιπα 4x4 blocks είναι 0.

- Υπολογισμός του BS για το κάθετο φιλτράρισμα

Σε αυτή την περίπτωση για τα blocks που βρίσκονται στα κάθετα όρια ενός 8x8 block πρέπει να υπολογιστεί το BS ξεχωριστά. Πχ για τα blocks 2 και 6 . Όμως για τα εσωτερικά 4x4 blocks ενός εσωτερικού 8x8 block ,ανήκουν στο ίδιο 4x8 block έχουν το ίδιο BS .΄Αρα αρκεί να υπολογιστεί για ένα από τα δύο το BS. Για παράδειγμα το 3 και το 7 που χρησιμοποιούν ως αναφορά το 2 και το 6, χρειάζεται να υπολογιστεί μόνο το BS για το 3 ή το 7.

4. 8x8



- Υπολογισμός του BS για το οριζόντιο φιλτράρισμα

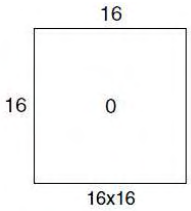
Σε αυτή την κατηγορία για το οριζόντιο φιλτράρισμα πρέπει να υπολογιστεί μόνο το BS για τα blocks που βρίσκονται στα οριζόντια όρια 2 8x8 blocks. Σε όλες τις άλλες περιπτώσεις είναι 0.

- Υπολογισμός του BS για το κάθετο φιλτράρισμα

Σε αυτή την κατηγορία για το κάθετο φιλτράρισμα πρέπει να υπολογιστεί μόνο το BS για τα blocks που βρίσκονται στα κάθετα όρια 2 8x8 blocks. Σε όλες τις άλλες περιπτώσεις είναι 0.

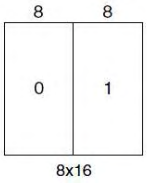
Συνεχίζοντας ,όπως έχει ήδη αναφερθεί μπορεί ένα macroblock 16x16 να κωδικοποιείται σε blocks. Ο τύπος των blocks δε αλλάζει ανά 8x8 blocks

1. 16x16



Σε αυτήν την περίπτωση τόσο για το οριζόντιο όσο και για κάθετο το BS είναι 0.

2. 8x16



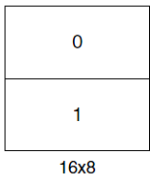
- Οριζόντιο φιλτράρισμα

  Για το οριζόντιο φιλτράρισμα τα BS όλων των blocks είναι 0 .

- Κάθετο φιλτράρισμα

  Για την διαδικασία του κάθετου φιλτραρίσματος πρέπει να υπολογιστούν μόνο τα BS στα κάθετα όρια δυο 8x8 block.Σε όλες τις άλλες περιπτώσεις είναι 0

3. 16x8



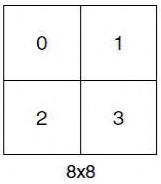
- Οριζόντιο φιλτράρισμα

  Για την διαδικασία του οριζόντιου φιλτραρίσματος αρκεί μόνο να υπολογιστεί η τιμή του BS για τα 4x4 blocks που βρίσκονται στα όρια δύο 8x8 blocks .

- Καθέτο φιλτράρισμα

  Για το κάθετο φιλτράρισμα τα BS όλων των blocks είναι 0.

4. 8x8



Για το υπολογισμό του BS των 4x4 blocks η διαδικασία που ακολουθείται είναι ίδια με αυτή με την περίπτωση 4 στην παράγραφο υπολογισμού των BS με χρήση sub blocks.

Συνοψίζοντας, εύκολα γίνεται κατανοητό ότι η προσέγγιση αυτή είναι πολύ καλύτερη από την αντίστοιχη στον αρχικό κώδικα. Αφού μόνο σε ελάχιστες περιπτώσεις απαιτείται ο υπολογισμός των BS για όλα τα 4x4 blocks.

Για όλο τον μηχανισμό λήψης απόφασης της τιμής του BS για τα 4x4 block στον αρχικό κώδικα χρειάζονταν συνολικά **1551900000** κύκλους μηχανής .Μέτα την αλλαγή του τρόπου λήψης απόφασης οι κύκλοι μηχανής που χρειάζονται είναι **661500000** κύκλοι μηχανής. Η συνολική επιτάχυνση 2.35x .Σε ποσοστό 58%. Συνολικά ο μηχανισμός του deblocking στο αρχικό κώδικα μαζί με την διαδικασία του φιλτραρίσματος συνολικά χρειάζονταν **4441500000** κύκλους μηχανής . Ενώ μετά τις βελτιστοποιήσεις χρειάζεται συνολικά **2262770000 .** Άρα η συνολική βελτιστοποίηση του μηχανισμού του deblocking είναι 1.96x . Σε ποσοστό 48%.

5.5 Γενική αλλαγή αρχιτεκτονικής για του τρόπου αποθήκευσης και κλιμάκωσης των συντελεστών

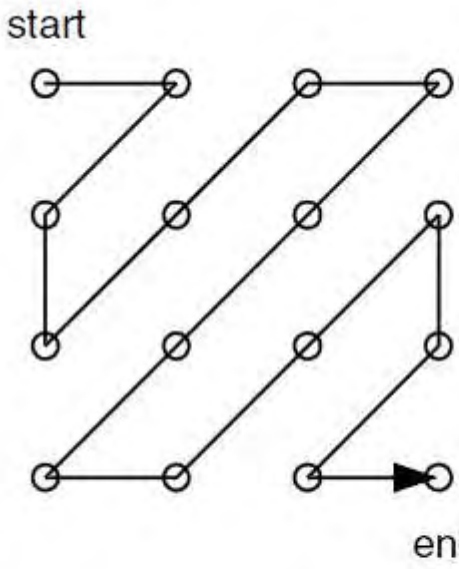
Σε αυτή την ενότητα θα περιγραφτούν οι αλλαγές που έγιναν για τον υπολογισμό των συντελεστών μετά την κλιμάκωση (coeff) και των συντελεστών πριν την κλιμάκωση (lengths).

Στο αρχικό κώδικα οι κωδικοποιημένοι συντελεστές που αποκωδικοποιούνται αλλά δε έχουν υποστεί κλιμάκωση αποθηκεύονται στον ίδια τοπική αποθήκη με τους συντελεστές που έχουν υποστεί κλιμάκωση .Όμως εκτός από την κοινή αποθήκη αποθηκεύονταν και σε μια άλλη αποθήκη ,η οποία όμως δε χρησιμοποιούταν από καμία διαδικασία του αποκωδικοποιητή. Αυτό είχε ως αποτέλεσμα την σπατάλη **1791300000** κύκλων μηχανή . ‘Ετσι η διαδικασία αντιγραφής από την μια αποθήκη στην άλλη αφαιρέθηκε από την υλοποίηση, δίνοντας σε ένα σημαντικό κέρδος. Ένα επιπλέον πρόβλημα ήταν η χρήση συντελεστών 32 bits, για να καλυφτεί μια οριακή περίπτωση η οποία λάμβανε χώρα μόνο για τους DC συντελεστές του chroma και αφορούσε ένα ενδιάμεσο αποτέλεσμα το οποίο είχε τιμή μεγαλύτερη του εύρους των signed short. Προφανώς το μέγεθός της αποθήκης των συντελεστών μπορεί να μειωθεί σε signed short και για την ειδική περίπτωση να χρησιμοποιηθεί προσωρινή μεταβλητή τύπου int για την αποθήκευση και στην συνέχεια αποθήκευση του τελικού αποτελέσματος στην αποθήκη των συντελεστών. Επιπλέον η αποθήκη αυτή ορίζεται σαν μέλος μια κλάσης και στην οποία περιέχεται δηλωμένη και η αποθήκη των συντελεστών πριν την κλιμάκωση (lengths),που όπως αναφέρθηκε είναι περιττή .Δηλαδή συνολικά σε αυτή την κλάση υπάρχουν δυο αποθήκες συνολικού μεγέθους 3072 Bytes ,ενώ στην πραγματικότητα χρειάζονται μόνο τα 768 Bytes ,δηλαδή χρειάζεται το ¼ της αρχικής μνήμης . Για το λόγο αυτό δηλώνεται έξω από την κλάση TCoeff, σε άλλη υπάρχουσα κλάση η οποία περιέχει την αποθήκη αποθήκευσης των συντελεστών πριν και μετά την κλιμάκωση και είναι ορατή από όλες τις κλάσεις που χρειάζονται την συγκεκριμένη αποθήκη. Η αποθήκη προφανώς δηλώνεται σαν short. Η αλλαγή

αυτή είναι πολύ σημαντική, διότι οι προσπελάσεις στην μνήμη πλέον είναι πολύ γρήγορές. Τέλος η κλιμάκωση των συντελεστών λαμβάνει χώρα κατά την διάρκεια αποκωδικοποίησης για κάθε block 4x4. Η προσέγγιση αυτή δε είναι βέλτιστη διότι οι περισσότεροι συντελεστές που διαβάζονται κατά την διάρκεια της αποκωδικοποίησης είναι 0. Επομένως η κλιμάκωση των συντελεστών αυτών ,πολλαπλασιάζοντας τους με μια σταθερά θα δίνει πάντα 0. Άρα σπαταλιούνται άσκοπα κύκλοι μηχανής για πολλαπλασιασμούς με το μηδέν. Επομένως μια βελτιστοποίηση αφορά την κλιμάκωση των συντελεστών κατά την διάρκεια ανάγνωσης από το αρχείο εισόδου. Στην συγκεκριμένη αποκωδικοποίηση χρησιμοποιείται το CABAC για την ανάκτηση των συντελεστών . Κατά την εκτέλεση του αλγορίθμου αυτού γίνεται έλεγχος αν ένας συντελεστής είναι 0. Σε αυτό το σημείο αν δε είναι 0 προστίθεται και ο κώδικας κλιμάκωσης .

Η αλλαγή αυτή είχε ως αποτέλεσμα συνολικό κέρδους από την συνάρτηση κλιμάκωσης, που πλέον εκτελείται μέσα στην διαδικασία της ανάκτησης των συντελεστών**, 1083600000** κύκλοι μηχανής. Επιπλέον οι κύκλοι μηχανής της συνάρτησης ανάκτησης των συντελεστών αυξήθηκε ελάχιστα , περίπου **153300000** κύκλους μηχανής ,δηλαδή σε ποσοστό 17.6% . Επομένως η διαδικασία κλιμάκωσης των συντελεστών βελτιστοποιήθηκε κατά 7.06x. Τέλος αυτή η αλλαγή σε συνάρτηση με την μη εκτέλεσης της αντιγραφής από την μια αποθήκη στην άλλη, μείωσε την συνολική εκτέλεση του αποκωδικοποιητή κατά **2721600000** κύκλους μηχανής.

Τέλος στην μη σημαντική αύξηση της συνάρτησης ανάγνωσης των συντελεστών κλιμάκωσης, συνέβαλε προφανώς το γεγονός ότι το πλήθος των συντελεστών που είναι διάφορο του 0 είναι πολύ μικρό και κάποιες μικρές αλλαγές. Συγκεκριμένα χρησιμοποιήθηκαν look up tables για να προσδιοριστεί όπου χρειαζόταν αν μια τιμή είναι ανάμεσα στο 0 και στο 4 . Επιπλέον η ανάγνωση των συντελεστών γίνεται με την μέθοδο του zig-zag για κάθε block 4x4.Ο τρόπος ανάγνωσης των συντελεστών φαίνεται στον παρακάτω σχήμα.



Zig-zag scan for 4 × 4 luma block (frame mode)

Η τιμή του start είναι 0 ή 1 και του end 15. Σύμφωνα με τον αρχικό κώδικα στην πρώτη φάση ελέγχεται αν οι συντελεστές δεν είναι μηδέν. Στην συνέχεια υπολογίζεται η τιμή του κάθε συντελεστή, όμως μη σειριακή προσπέλαση της μνήμης κοστίζει. Για το λόγο αυτό, για τον πρώτο έλεγχο ορίζεται μια τοπική ευθυγραμμισμένη μνήμη 16 θέσεων στην οποία αποθηκεύονται η πληροφορία για το αν είναι μη μηδενικός ο συντελεστής .Στην συνέχεια κατά την εκτέλεση του zig – zag διαβάζεται αυτή η πληροφορία . Με αυτό τον τρόπο αποφεύγεται η ανάγνωση και η εγγραφή από θέσεις μνήμης που δε είναι συνεχόμενες κατά την πρώτη φάση του αλγορίθμου. Στην δεύτερη φάση γίνεται η ανάγνωση της πληροφορίας πλέον με την μέθοδο του zig –zag και αποθηκεύεται στην αποθήκη των συντελεστών.

5.6 Αλλαγή του ελέγχου αν ένα block 4x4 έχει υπόλοιπα διάφορα του 0

Στον αρχικό κώδικα μετά την διαδικασία της αποκωδικοποίησης για τα inter macroblocks, για κάθε mmacroblock η τοπική αποθήκη στην οποία αποθηκεύονται τα υπόλοιπα αντιγράφεται σε ένα εξωτερικό buffer. Όταν ολοκληρωθεί η διαδικασία η εξωτερική αποθήκη χρησιμοποιείται από την διαδικασία λήψης απόφασης για τον υπολογισμό του BS. Μετά την

αλλαγή του τρόπου εκτέλεσης του φιλτραρίσματος και της μεθόδου λήψης απόφασης για το BS ,δε χρειάζεται αρχικά αυτή η αντιγραφή στον εξωτερικό buffer και επιπλέον μπορεί να υπολογιστεί αν ένα block 4x4 έχει μη μηδενικά υπόλοιπα κατά την εκτέλεση της συνάρτησης addRes. Στην διαδικασία πρόσθεσης των υπολοίπων που προκύπτουν από το inverse transform και αυτά του προηγούμενου layer. Επιπλέον η διαδικασία αυτή θα γίνει με εντολές SIMD. Για την υλοποίηση της διαδικασίας με διανυσματικές εντολές αρχικά έχουν προκύψει οι καταχωρητές xmm0,xmm1,xmm2,xmm3,xmm4,xmm5,xmm6,xmm7 απο την διαδικασία της πρόσθεσης που περιέχουν τις τιμές των υπολοίπων για ένα block 8x8. Στο παρακάτω σχήμα υπενθυμίζεται πως τοποθετούνται τα 4x4 blocks στα μεγαλύτερα blocks 8x8.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Άρα από την διαδικασία της πρόσθεσης σε κάθε επανάληψη έχουν αποθηκευτεί οι τιμές για 4 blocks. Για παράδειγμα στην πρώτη επανάληψη στον καταχωρητή xmm0 έχουν αποθηκευτεί οι τιμές για την πρώτη γραμμή των block 0 και 1. Στον xmm1 η πρώτη γραμμή των block 2 και 3. Στον xmm2 η δεύτερη γραμμή των block 0 και 1. Στον xmm3 η δεύτερη γραμμή των block 2 και 3. Ομοίως και στους υπόλοιπους καταχωρητές έχουν αποθηκευτεί οι αντίστοιχες γραμμές .

Η διαδικασία θα περιγραφτεί για τους καταχωρητές στους οποίους έχει αποθηκευτεί πληροφορία για τα blocks 0 1. Η διαδικασία είναι ίδια και για τους άλλου καταχωρήτες .Αρχικά όπως αναφέραμε οι καταχωρητές xmm0 ,xmm2 ,xmm4,xmm6 περιέχουν τα pixel και των 4 γραμμών. Οι τιμές των καταχωρητών συγχωνεύονται σε ένα καταχωρητή με χρήση της εντολής por. Αυτό γίνεται γιατί η πληροφορία που χρειάζεται είναι αν το block έχει μη μηδενικές τιμές. Στην συνέχεια για το block 0 ολισθαίνονται οι τιμές που βρίσκονται στις θέσεις 2 και 3 και αντιστοιχούν στις αντίστοιχες στήλες με χρήση της εντολής psrldq κατά 4 θέσεις δεξιά. Στην

συνέχεια ο νέος καταχωρητής συγχωνεύεται με τον αρχικό και το αποτέλεσμα μεταφέρεται σε μια ακέραια μεταβλητή με χρήση της εντολής movd και ελέγχεται αν είναι μηδέν. Για τα blocks 2 και 3 η διαδικασία είναι ίδια, αλλά με χρήση της εντολής psrldq και ολίσθηση κατά 8 θέσεις δεξιά τα αποτέλεσμα του πρώτου καταχωρητή που προέκυψε από την συγχώνευση μεταφέρονται στις χαμηλότερες θέσεις .

Η αρχική συνάρτηση υπολογισμού της συγκεκριμένης διαδικασίας χρειαζόταν **747600000** κύκλους μηχανής και η έκδοση της με διανυσματικές εντολές χωρίς αλλαγή αρχιτεκτονικής απαιτούσε **258300000 ,** δηλαδή 2.98 φορές πιο γρήγορη. Με την ενσωμάτωση της στην συνάρτηση addRes η συνάρτηση αυτή αυξήθηκε κατά **165900000**. Άρα η διαδικασία βελτιστοποιήθηκε σε σχέση με την αρχική κατά **4.55x** που αντιστοιχεί σε ποσοστό **78%** και από την διαδικασία που είναι υλοποιημένη με εντολές SSE είναι πιο γρήγορη κατά **1.55 x**.

6.1 Εισαγωγή

Εκτός από τις αλλαγές που αναφέρθηκαν στα προηγούμενα, παρεμετροποιήθηκε και ο compiler που χρησιμοποιήθηκε για να μεταγλωττιστεί και να παράγει το εκτελέσιμο. Στο σημείο αυτό πρέπει να αναφερθεί ότι ο compiler που χρησιμοποιήθηκε ήταν ο VC σε περιβάλλον visual studio 2009. Στο σημείο αυτό πρέπει να αναφερθεί ότι διαπιστώθηκε σφάλμα βελτιστοποίησης κατά την δημιουργία εκτελέσιμου που να τρέχει σε 64 –bit συστήματα. Συγκεκριμένα ο μεταγλωττιστής δε λάμβανε υπόψη κατά την εκτέλεση των εντολών SIMD στην διαδικασία του οριζόντιου φιλτραρίσματος για το Luma εξάρτηση ανάγνωσης μετά από εγγραφή με αποτέλεσμα να κάνει αναδιάταξη των εντολών και να χάνονται ενδιάμεσα αποτελέσματα.

6.2 Αλλαγές που έλαβαν χώρα

Στην ενότητα αυτή θα αναφερθούν μια προς μια οι αλλαγές καθώς και επεξήγηση.

- Η παράμετρος **C/C++→Optimization→Optimazation** τέθηκε στην τιμή **Maximize Speed(/02)**

  Η αλλαγή αυτή έχει ως στόχο την δημιουργία κώδικα που θα δοθεί έμφαση στην ταχύτητα.
- Η παράμετρος **C/C++→Optimization->Inline Function Expansion** τέθηκε στην τιμή **Any Suitable**

  Ο μεταγλωττιστής εκτός από τις συναρτήσεις που έχουν επιλεχθεί σαν inline επιλέγει και άλλες αυτόματα σαν inline
- Η παράμετρος **C/C++→Optimization→Enable Intrinsic Functions** τέθηκε στην τιμή **yes**.

  Χρησιμοποιείται για να παραχθεί γρηγορότερος αλλά μεγαλύτερος κώδικας στις συναρτήσεις που έχουν χρησιμοποιηθεί SSE εντολές
- Η παράμετρος **C/C++→ Optimization →Favor Speed or Size** τέθηκε στην τιμή **Favor Fast Code**

  Χρησιμοποιείται για την αύξηση της ταχύτητας των εκτελέσιμων αρχείων και των dll.
- Η παράμετρος C/C++→ **Optimization→Whole Program Optimization** τέθηκε στην τιμή **Enable link-time code generation (/GL).**

Η χρήση αυτής της παραμέτρου έχει 2 σημαντικά πλεονεκτήματα. Βελτιστοποιεί τους καταχωρητές που χρησιμοποιούν τα ορίσματα μιας συνάρτησης .Μπορεί να κάνει inline μια συνάρτηση ο μεταγλωττιστής, ακόμα και αν έχει δηλωθεί σε αλλό module. Για παράδειγμα αν μια συνάρτηση σαν extern μπορεί να την μετατρέψει σαν inline στο σημείο που πραγματικά δηλώθηκε.

- Η παράμετρος **C/C++→Code Generation→Enable Function Level Linking** τέθηκε στην τιμή **yes** .

  Ο linker αναγκάζει να πακεταριστούν οι συναρτήσεις ξεχωριστά σαν COMDATs. Αυτό έχει ως στόχο την μη μεταγλώττιση συναρτήσεων που δε καλούνται από κανένα module ή να κατηγοριοποίηση ξεχωριστά αυτές τις συναρτήσεις .
- Η παράμετρος **C/C++→Code Generation→Enhanced Instruction Set** τέθηκε στην τιμή **SSE Instruction Set**

  Δίνει την δυνατότητα στον compiler να χρησιμοποιεί όπου κρίνει απαραίτητο να χρησιμοποιεί SSE2 εντολές. Απαραίτητη προϋπόθεση είναι τα δεδομένα να είναι ευθυγραμμισμένα σε 16 –bit.
- Η παράμετρος **C/C++→Code Generation→Floationg Point Model** τέθηκε στην τιμή **fast(fp).**

  Προσπαθεί να δημιουργήσει το γρηγορότερο κώδικα για εκτέλεση πράξεων με δεκαδικούς .Η αλλαγή είναι απαραίτητη αν τεθεί η μεταβλητή **Enhanced Instruction Set** στην τιμή **SSE Instruction Set.**
- Η παράμετρος **C/C++→Advanced→Calling Convertions** τέθηκε στην τιμή **__fastcall**

  Προσπαθεί να φορτώσει τα ορίσματα των συναρτήσεων σε καταχωρητές όταν αυτό είναι δυνατόν.
- Η παράμετρος **Linker→Optimization→References** τέθηκε στην τιμή Eliminate Unreferenced Data (/OPT:REF)

  Ο μεταγλωττιστής διαγράφει οριστικά τις συναρτήσεις που είναι πακεταρισμένες σαν συναρτήσεις που δε καλούνται από κανένα module μέσα στον κώδικα.
- Η παράμετρος **Linker→Optimization→ Link Time Code Generation Use Link Time** τέθηκε στην τιμή **Use Link Time Code Generation (/ltcg)**

  Έχει ήδη περιγραφεί.

Η συνολική βελτιστοποίηση που έδωσε η παραμετροποίηση του compiler είναι 1.13x που αντιστοιχεί σε ποσοστό 10%.

Κεφάλαιο 7

Επίλογος και σύγκριση με H.264

Μετά από παραπάνω από 3 μήνες δουλειάς η διπλωματική εργασία ολοκληρώθηκε επιτυχώς. Ο στόχος επετεύχθη και ο αποκωδικοποιητής SVC πλέον αποκωδικοποιεί καρέ σε πραγματικό χρόνο. Συγκεκριμένα η αρχική του έκδοση ο αποκωδικοποιητής αποκωδικοποιούσε με ρυθμό ανάμεσα σε 10.0 και 10.3 fps, δηλαδή για 300 frames απαιτούνταν 29.0 secs με 30.0 secs. Πλέον μετά από ένα σύνολο βελτιστοποιήσεων ο αποκωδικοποιητής πλέον αποκωδικοποιεί με ρυθμό κατά μέσο όρο 30.3 fps και χρόνο 9.9 που αντιστοιχεί σε επιτάχυνση 2.9x και σε ποσοστό βελτιστοποίησης περίπου στο 69~70%. Στις μετρήσεις που έγιναν η πιο γρήγορη αποκωδικοποίηση διήρκησε 9.75 που δίνει ένα ρυθμό κοντά στα 31 fps και επιτάχυνση 3x .

Ενδιαφέρον παρουσιάζει και η σύγκριση της αποκωδικοποίησης σε πολλαπλά layers με αυτή του ενός που ισοδυναμεί με αποκωδικοποίηση του κλασικού προτύπου H.264. Η βελτιστοποιημένη έκδοση του αποκωδικοποιητή για 300 frames και για το ίδιο bitrate χρειάζεται 7.217 sec που αντιστοιχεί σε ρυθμό 41.56 fps . Ο αρχικός κώδικας για την αποκωδικοποίηση ενός layer χρειάζεται περίπου 16.3 δευτερόλεπτα , δηλαδή 18.4 fps. Στο παρακάτω πίνακα παρουσιάζεται και το PSNR του SVC αποκωδικοποιητή πολλαπλών layers και του SVC αποκωδικοποίησης ενός layer .

	Luma	Cb	Cr
SVC-13 Layers	36.01	43.78	45.98
SVC-Single Layer	36.175	42.6383	44.685

Συνοψίζοντας θα ήταν χρήσιμο να αναφερθούν και μελλοντικές εργασίες πάνω σε αυτό το πρότυπο. Ένα μάλλον δύσκολο αλλά και πολύ ενδιαφέρον εγχείρημα θα ήταν να ξαναγραφτεί ο κώδικας σε C. Επιπλέον έχοντας γίνει αρκετές αρχιτεκτονικές αλλαγές δε θα ήταν εξαιρετικά χρονοβόρο η παραλληλοποίηση του αποκωδικοποιητή. Και τέλος το ποιο ενδιαφέρον κατά την προσωπική μου άποψη είναι ο κώδικας αυτός να υλοποιηθεί σε ένα σύστημα ειδικού σκοπού όπως οι επεξεργαστές ARM .

8. Βιβλιογραφία

- H.264 and MPEG-4 compression ,Iain E.G Richardson
- An Efficient SIMD-based Quarter-Pixel Interpolation Method for H.264/AVC, Chae-Bong Sohn, and Hye-Jeong Cho
- Improved Intra Coding Methods for H.264/AVC, Li Song, Yi Xu, Cong Xiong and Leonardo Traversoni
- In-loop Deblocking Filter for H.264/AVC Video, Gulistan Raja1and Muhammad Javed Mirza
- H.264 DECODER OPTIMIZATION EXPLOITING SIMD INSTRUCTIONS, Chae-Bong Sohn, Hye-Jeong Cho
- SIMD OPTIMIZATION OF THE H.264/SVC DECODER WITH EFFICIENT DATA STRUCTURE, Joohyun Lee, Gwanggil Jeon, Sangjun Park, Taeyoung Jung, Jechang Jeong
- Overview of the Scalable Video Coding Extension of the H.264/AVC Standard, Heiko Schwarz, Detlev Marpe
- wikipedia,google
- Διαφάνειες μαθήματος "Τεχνικές Συμπίεσης Ήχου και Βίντεο "