Πανεπιστήμιο Δυτικής Μακεδονίας
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

# Σχεδιασμός και Υλοποίηση Επιταχυντή Υλικού για τον Αλγόριθμο Εκτίμησης Κίνησης Εξαντλητικής Αναζήτησης

Διπλωματικη Εργασια

Θωμας Ι. Μακρυνιωτης

Επιβλεπων καθηγητης:

Δρ. Μηνας Δασυγενης

Kozanh, Ιουλιος 2017

# Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο

"Σχεδιασμός και Υλοποίηση Επιταχυντή Υλικού για τον Αλγόριθμο Εκτίμησης Κίνησης Εξαντλητικής Αναζήτησης"

καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Μηνά Δασυγένη, αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

# Σχεδιασμός και Υλοποίηση Επιταχυντή Υλικού για τον Αλγόριθμο Εκτίμησης Κίνησης Εξαντλητικής Αναζήτησης


## Σύνοψη

Οι αλγόριθμοι εκτίμησης κίνησης (motion estimation algorithms) αποτελούν αναπόσπαστο κομμάτι όλων των σύγχρονων τεχνικών συμπίεσης/κωδικοποίησης video. Λόγω των εντατικών υπολογισμών που γίνονται για τον καθορισμό των διανυσμάτων κίνησης, κατά τη φάση αυτή απαιτείται μεγάλη επεξεργαστική ισχύς . Η συγκεκριμένη διεργασία είναι ιδιαίτερα δαπανηρή από πλευράς χρόνου και ενέργειας, ειδικά όταν γίνεται σε επίπεδο λογισμικού και ανατίθεται σε "μη-εξειδικευμένες" μονάδες όπως για παράδειγμα σ'εναν επεξεργαστή γενικής χρήσης (general purpose CPU). Η συγκεκριμένη διπλωματική εργασία, περιγράφει τον σχεδιασμό και την ανάπτυξη ενός περιφερειακού που λειτουργεί ως συνεπεξεργαστής-επιταχυντής και είναι εξειδικευμένο στην εκτέλεση του Αλγορίθμου Εκτίμησης Κίνησης Εξαντλητικής Αναζήτησης (Full-Search Motion Estimation Algorithm). Παράλληλα, περιγράφεται ο σχεδιασμός και η δημιουργία ενος πλήρους περιβάλλοντος διασύνδεσης με έναν επεξεργαστή ARM, χρησιμοποιόντας την πλατφόρμα Zynq-7000. Στόχος μας ήταν η δημιουργία ενός αξιόπιστου συστήματος που θα επιτρέπει την εκτέλεση του FSME με πολύ μεγαλύτερη ταχύτητα και πολύ μικρότερη κατανάλωση ενέργειας σε σχέση με συμβατικούς επεξεργαστές.

# Design and Implementation of a Hardware Accelerator for the Full Search Motion Estimation Algorithm


## Abstract

Motion estimation algorithms are an integral part of all modern video compression / coding techniques. Due to the intensive calculations required for the determination of Motion Vectors, the procedure requires great processing power. This specific process may be extremely costly in terms of time and energy, especially when it is performed at software level and assigned to ”non-specialized” units such as a general purpose CPU. The current diploma thesis, describes the design and development of a peripheral which acts as a co-processor/accelerator and it is specialized in executing the Full-Search Motion Estimation Algorithm. At the same time, we are describing the design and implementation of a complete interfacing environment with an ARM processor, based on the platform Zynq-7000. Our goal was to create a reliable system that would allow it to run the FSME algorithm faster and with much lower power consumption than conventional processors.

Στην οικογενεια και τους φιλους μου

# Περιεχόμενα

# Ευχαριστίες

Κάπου εδώ λοιπόν έφτασε το τέλος αυτού του κύκλου. Για να είμαι ειλικρινής, φαινόταν πάντα πολύ μακριά η ώρα που θα έγραφα επιτέλους αυτές τις γραμμές. Γεγονός βέβαια είναι πως δεν τις γράφω τυχαία, ούτε επειδή "έτσι είθισται". Μιας και είναι το πρώτο ολοκληρωμένο έργο μου, η πρώτη ακαδημαϊκή εργασία με κάποια ουσιαστική αξία, θεωρώ πως είναι ευκαιρία να αφιερώσω λίγο χρόνο και χώρο για να ευχαριστήσω όλους εκείνους που βοήθησαν με τον έναν ή τον άλλο τρόπο στο να βρίσκομαι σήμερα εδώ.

Καταρχάς, θα ήθελα να ευχαριστήσω την οικογένεια μου, τον πατέρα μου Γιάννη και τη μητέρα μου Γεωργία, καθώς και τον αδερφό μου Κωνσταντίνο, για όλη τη συμπαράσταση, τη στήριξη, την πίστη αλλά και τη γκρίνια (που ήταν απαραίτητη ορισμένες φορές, το ομολογώ) κατά τη διάρκεια των σπουδών μου. Κατόπιν θα ήθελα να ευχαριστήσω ξεχωριστά τους φίλους μου Πέτρο, Κώστα, Λεβόν και Μάκη για όλη εκείνη τη βοήθεια, την εμψύχωση αλλά και την κατανόηση που έδειξαν όταν τα πράγματα πήγαιναν στραβά, αλλά και τη χαρά και τον ενθουσιασμό τους μπροστά σε κάθε μικρή ή μεγάλη επιτυχία.

Οφείλω να ευχαριστήσω ξεχωριστά δύο ανθρώπους από το ακαδημαϊκό μου περιβάλλον. Ο πρώτος είναι ο Δρ. Μηνάς Δασυγένης, με τον οποίο συνεργαστήκαμε πολλές φορές κατά τη διάρκεια των σπουδών μου και η επιτυχία αυτής της εργασίας οφείλεται αναμφισβήτητα, στις συμβουλές και την καθοδήγησή του. Ο δεύτερος είναι ο Δρ. Γιώργος Δημητρακόπουλος, ο οποίος αν και πλέον δε βρίσκεται στο τμήμα μας, η παλαιότερη συνεργασία μαζί του έδρασε καταλυτικά στην απόφασή μου να ασχοληθώ με τον τομέα της Ψηφιακής Σχεδίασης.

Για το τέλος, άφησα ορισμένους ανθρώπους που αν και η ζωή θέλησε να πάρουμε χωριστούς δρόμους, στάθηκαν δίπλα μου για πολλά χρόνια. Δεν μπορώ και πιθανότατα δε θα μπορέσω ποτέ να παραβλέψω τη συνεισφορά τους. Τους ευχαριστώ ειλικρινά και τους εύχομαι κάθε επιτυχία στη ζωή τους, ωστόσο δε θα τους αναφέρω ονομαστικά, καθώς δεν είμαι σε θέση να γνωρίζω εαν και κατά πόσο θα το ήθελαν αυτό. Εκείνοι οι λίγοι που θα το διαβάσουν ελπίζω πως θα καταλάβουν.

# Κατάλογος Σχημάτων

# Κατάλογος Πινάκων

# 1 Εισαγωγή

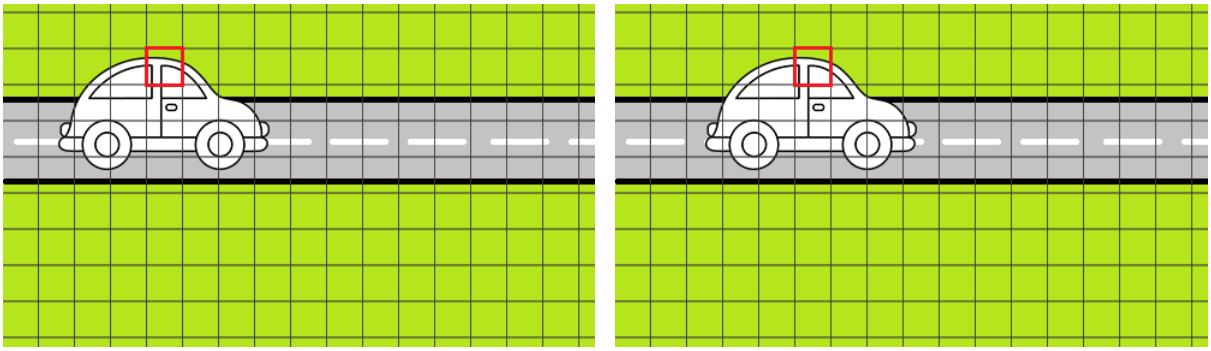
## 1.1 Περιγραφη του προβληματος

Με τη ραγδαία εξάπλωση του Internet, εξίσου ραγδαία ήταν και η εξάπλωση των πολυμέσων, μεταξύ των οποίων οι στατικές αλλά και οι κινούμενες εικόνες (βίντεο). Δυστυχώς, με την πάροδο του χρόνου, το bandwidth παρέμενε περιορισμένο, ενώ οι απαιτήσεις για ολοένα και υψηλότερη ποιότητα εικόνας αύξαναν συνεχώς. Το πρόβλημα είχε ήδη παρατηρηθεί από τα προηγούμενα χρόνια, και η αποτελεσματικότερη λύση ήταν η συμπίεση των αρχείων εικόνας/βίντεο, με σκοπό τη μείωση του μεγέθους τους και κατα συνέπεια της μνήμης και του εύρους ζώνης που απαιτούνταν. Η χρήση αλγορίθμων συμπίεσης ξεκίνησε με την εφαρμογή διαφόρων τεχνικών συμπίεσης σε στατικές εικόνες, αρχικά σχετικά απλών, όπως στην περίπτωση του JPEG με τη χρήση του Διακριτού Μετασχηματισμού Συνημιτόνου (Discrete Cosine Transform - DCT), και στη συνέχεια πολυπλοκότερων όπως η συμπίεση με Fractals ή η συμπίεση πολλαπλών επιπέδων όπως στο πρότυπο PNG.

Το ίδιο μοτίβο ακολουθήθηκε και στην περίπτωση των βίντεο, τα οποία ως γνωστόν, είναι αλληλουχίες στατικών εικόνων που ονομάζονται **καρε (frames)**. Μια από τις πρώτες τεχνικές που εφαρμόστηκαν ήταν η συμπίεση κάθε καρέ με το πρότυπο JPEG κάτι που έγινε γνωστό ως MJPEG (Motion-JPEG). Φυσικά, η σχέση ποιότητας/μεγέθους ήταν πολύ κακή και πλέον έπρεπε να εφευρεθούν νέες τεχνικές που θα επέτρεπαν μεγάλη συμπίεση κρατώντας όσο το δυνατόν σε υψηλότερα επίπεδα την

ποιότητα.

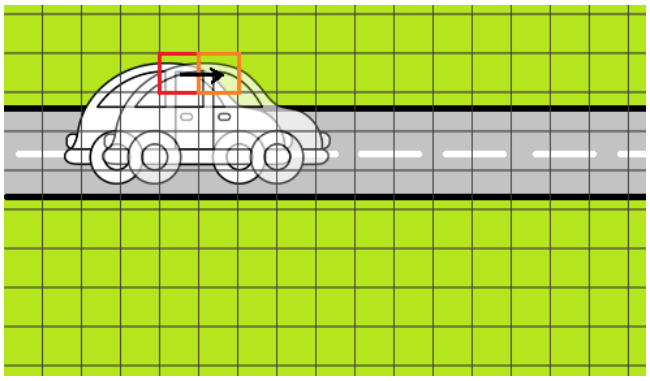
Αυτό που παρατηρήθηκε πολύ γρήγορα ήταν το γεγονός πως, εφόσον πρόκειται για αλληλουχίες καρέ, το κάθε καρέ σε σχέση με τα γειτονικά του παρουσίαζε τεράστιες ομοιότητες ως προς την πληροφορία που έφερε. Αυτή η παρατήρηση οδήγησε στη δημιουργία ενός νέου όρου, αυτού του **χρονικού πλεονασμού (temporal redundancy)**. Ήδη ήταν γνωστή η έννοια του **χωρικού πλεονασμού (spatial redundancy)** η οποία αναφέρεται στην πληροφορία που μένει αναλλοίωτη μέσα σε μια συγκεκριμένη περιοχή της εικόνας, π.χ. οι τιμές των pixel ενός κόκκινου τετραγώνου διαστάσεων 32 × 32 pixel. Η έννοια του χρονικού πλεονασμού αφορά στην πληροφορία που μένει αναλλοίωτη μεταξύ των διαδοχικών καρέ. Για παράδειγμα, δύο διαδοχικά frames από ένα τμήμα βίντεο που δείχνουν την κίνηση ενός αυτοκινήτου σε ένα δρόμο έχουν εκ των πραγμάτων πολύ μικρές διαφορές μεταξύ τους (σχήμα 1.1).



**Σχήμα 1.1**: Δυο διαδοχικά frames

Όπως γίνεται εύκολα αντιληπτό, είναι δυνατόν να επιτευχθεί μεγάλος βαθμός συμπίεσης αν εκμεταλλευτούμε αυτα τα είδη των πλεονασμών. Πιο συγκεκριμένα, και σε ότι αφορά στον χρονικό πλεονασμό, είναι δυνατόν να συμπιέσουμε ένα καρε χωρίζοντάς το σε **μπλοκ (blocks)**, υπολογίζοντας την κίνηση τους και απεικονίζοντας την χρησιμοποιώντας ένα **διάνυσμα κίνησης (motion vector)**. Υπολογίζοντας τα διανύσματα κίνησης για όλα τα μπλοκ του καρέ έχουμε μια **εκτίμηση κίνησης**, και πρόκειται στην ουσία για έναν τρόπο να κωδικοποιήσουμε την πλεονάζουσα πληροφορία.

Κατα τη διαδικασία της συμπίεσης, αυτό που συμβαίνει είναι ο διαχωρισμός κάθε καρέ σε **περιοχές αναζήτησης** που με τη σειρά τους διαχωρίζονται σε μικρότερα μπλοκ (macroblocks). Κάθε macroblock της περιοχής αναζήτησης **Π** ενός καρέ **ν**,



**Σχήμα 1.2**: Το διάνυσμα κίνησης του σημειωμένου macroblock του σχήματος 1.1

γίνεται προσπάθεια να αντιστοιχιθεί με κάποιο από τα macroblocks της περιοχής αναζήτησης **Π** ενός καρέ **ν+1** (του επόμενου ή των επόμενων ανάλογα με την τεχνική). Η καλύτερη δυνατή αντιστοίχιση, είναι και αυτή που θα δώσει τελικά το διάνυσμα κίνησης του συγκεκριμένου macroblock.

Υπάρχουν πολλοί αλγόριθμοι που υλοποιούν μια τέτοια διαδικασία αντιστοίχισης, όπως η **δισδιάστατη λογαριθμική αναζήτηση (2-D Logarithmic Search)** [11], η **αναζήτηση τριών βημάτων (Three-Step Search)** [12], ο αλγόριθμος **Diamond Search** [37] και η **εξαντλητική αναζήτηση**. Οι αλγόριθμοι αυτοί ονομάζονται **block matching algorithms** επειδή προσπαθούν να υπολογίσουν τα διανύσματα κίνησης με βάση την αντιστοίχιση macroblocks. Όλοι τους φυσικά έχουν τόσο πλεονεκτήματα όσο και μειονεκτήματα, όπως η ταχύτητα εκτέλεσης ή η ακρίβεια της αντιστοίχισης. Ειδικά όμως στην περίπτωση της εξαντλητικής αναζήτησης ισχύουν δύο πράγματα: παρέχει το καλύτερο δυνατό αποτέλεσμα απο πλευράς ακρίβειας, δυστυχώς όμως υστερεί σε ταχύτητα και κατανάλωση πόρων, διότι πραγματοποιεί πληθώρα υπολογισμών αφού επεξεργάζεται pixel προς pixel κάθε macroblock.

Ακριβώς αυτό ήταν το πρόβλημα που έπρεπε να λύσουμε. Ο αλγόριθμος της εξαντλητικής αναζήτησης βρίσκει σημαντικές εφαρμογές στις σύγχρονες τεχνικές κωδικοποίησης/συμπίεσης βίντεο όπως το πρότυπο H.265/HEVC, όμως είναι ιδιαίτερα δαπανηρός απο πλευράς υπολογιστικής ισχύος και κατα συνέπεια, ενέργειας. Οι περισσότερες υλοποιήσεις του σε επίπεδο λογισμικού δεν ανταπεξέρχονται ικανοποιητικά και οπωσδήποτε δεν μπορούν να χρησιμοποιηθούν για την κωδικοποίηση βίντεο σε πραγματικό χρόνο (για παράδειγμα σε φορητές κάμερες, κινητών τηλεφώνων, κλπ) αφού η εκτέλεση τους γίνεται χρησιμοποιώντας τα σετ εντολών επεξερ-

γαστών γενικής χρήσης (General-Purpose CPUs) ακόμα και όταν στους τελευταίους έχουν ενσωματωθεί ειδικά σετ εντολών για πολυμέσα.

Η λύση στο πρόβλημα μπορεί να δοθεί με το σχεδιασμό και την υλοποίηση ενός κυκλώματος ειδικού σκοπού, το οποίο λειτουργεί σε συνεργασία με την κεντρική CPU και είναι εξειδικευμένο στην εκτέλεση του αλγορίθμου εξαντλητικής αναζήτησης. Το κύκλωμα αυτό, ένας συνεπεξεργαστής-επιταχυντής, μπορεί να βελτιώσει σημαντικά την απόδοση του αλγορίθμου και να τον καταστήσει μια λογική λύση ακόμα και σε περιπτώσεις όπου απαιτείται real-time κωδικοποίηση.

## 1.2 Κινητρα και Στοχοι Υλοποιησης

Τα βασικά κίνητρα αναφέρθηκαν και στην προηγούμενη παράγραφο. Ο FSBMA (Full-Search Block Matching Algorithm) είναι μια πολύ καλή προσέγγιση στο πρόβλημα της εκτίμησης κίνησης για εφαρμογές κωδικοποίησης βίντεο, που ωστόσο έχει τεράστιες απαιτήσεις σε επεξεργαστική ισχύ. Είναι επίσης σαφές πως αυτή η επεξεργαστική ισχύς που απαιτεί, μεταφράζεται σε μεγάλη κατανάλωση ενέργειας απο τις υπολογιστικές μονάδες, κάτι που καθιστά ασύμφορη την επιλογή του σε ενσωματωμένα συστήματα και φορητές συσκευές, που λειτουργούν κυρίως με μπαταρίες. Επιπλέον, η έλευση νέων προτύπων για την κωδικοποίηση βίντεο UHD (Ultra-High Definition) απαιτεί τη χρήση όσο το δυνατόν πιο αξιόπιστων αλγορίθμων, με ελάχιστες απώλειες στην ποιότητα.

Οι στόχοι της υλοποίησης ήταν τρεις. Πρώτον, θα έπρεπε η υλοποίηση στο hardware να είναι τάξεις μεγέθους ταχύτερη στην εκτέλεση από τις υλοποιήσεις σε software. Δεύτερον, θα έπρεπε η κατανάλωση ενέργειας να είναι όσο το δυνατόν πιο περιορισμένη, σε επίπεδα όπου θα δικαιολογείται η χρήση του κυκλώματος σε φορητές συσκευές. Τρίτος στόχος της υλοποίησης ήταν η κατασκευή ενός ολοκληρωμένου περιβάλλοντος που θα περιλαμβάνει το κύκλωμα του επιταχυντή, τον επεξεργαστή και τα κυκλώματα διασύνδεσης, αλλά και το λειτουργικό σύστημα μαζί με το λογισμικό ελέγχου (driver) του επιταχυντή. Έτσι, μπορούμε να ελέγξουμε και να διαπιστώσουμε στην πράξη την απόδοση του συστήματος.

Αξίζει να σημειώσουμε ότι τα αποτελέσματα αυτής της εργασίας έχουν παρουσιαστεί σε διεθνή συνέδρια ύστερα από κρίση: SouthEast European Design Automation, Computer Engineering, Computer Networks and Social Media Conference [15] και 6th Conference on Modern Circuits and Technologies [14].

## 1.3 Περιπτωσεις παρομοιας ερευνας

Παρόμοιο έργο έχει εκπονηθεί από πολλούς ερευνητές ανα τον κόσμο. Χαρακτηριστική είναι η δουλειά του Y. Ismail [9], [10], [8], αλλά και των Goel, Bayoumi [7], όπως και των Redkar και Kuwelkar [21], όπου αναφέρονται με λεπτομέρειες στις διαδικασίες σχεδιασμού και υλοποίησης ενός κυκλώματος επιτάχυνσης για τον αλγόριθμο FSBMA. Φυσικά κάθε ομάδα εστιάζει σε διαφορετικά στοιχεία, όπως για παράδειγμα η επαναχρησιμοποίηση δεδομένων έτσι όπως προτάθηκε από τον Ismail και τους συνεργάτες του.

Ο Olivares και οι συνεργάτες του [19] σχεδίασαν επίσης ένα κύκλωμα υλοποίησης του αλγορίθμου FSBMA και το συνέκριναν με παλαιότερες υλοποιήσεις των Ryszko [27], Wong [32], Roma [26] και Loukil [13], επιτυγχάνοντας σημαντική αύξηση της απόδοσης.

## 1.4 Διαρθρωση κειμενου

Τα επόμενα κεφάλαια οργανώνονται ως εξής:

Στο δεύτερο κεφάλαιο γίνεται μια αναλυτική επεξήγηση της διαδικασίας συμπίεσης, καθώς και επιγραμματική αναφορά σε διάφορους αλγορίθμους και τεχνικές συμπίεσης εικόνας και βίντεο. Δίνεται επίσης όλο το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση του Αλγορίθμου Εκτίμησης Κίνησης Εξαντλητικής Αναζήτησης.

Το τρίτο κεφάλαιο αφορά το επίπεδο του hardware. Στο κεφάλαιο αυτό περιγράφεται αναλυτικά η αρχιτεκτονική του επιταχυντή, και οι βασικές συστατικές του μονάδες. Περιγράφονται επίσης οι διαδικασίες που ακολουθήθηκαν στο σχεδιασμό του, αλλά και οι προκλήσεις και τα σφάλματα που έπρεπε να αντιμετωπιστούν. Αναλύεται επίσης η διασύνδεση του επιταχυντή (μονάδες DMA, διασύνδεση AXI) με το υπόλοιπο υπολογιστικό σύστημα: τον επεξεργαστή ARM και την κεντρική μνήμη.

Στο τέταρτο κεφάλαιο περιγράφεται η υλοποίηση του software. Σχολιάζονται θέματα του λειτουργικού συστήματος (Linux), και αναλύεται η διαδικασία συγγραφής του driver και του testbench για τον έλεγχο και το verification του συστήματος.

Τέλος, στο πέμπτο κεφάλαιο παρατίθενται τα αποτελέσματα, συνοδευόμενα από στοιχεία μετρικών και συμπεράσματα για τη λειτουργικότητα και αξιοπιστία του συστήματος.

Στα παραρτήματα βρίσκονται τόσο τμήματα κώδικα όσο και σχεδιαγράμματα που περιγράφουν αναλυτικότερα τη λειτουργία του επιταχυντή.

# 2
# Θεωρητικό Υπόβαθρο

## 2.1 Η Κωδικοποιηση Εικονας

Οι ψηφιακές εικόνες ήταν από τις πρώτες μορφές δεδομένων που απαιτήθηκε να συμπιεστούν. Ο λόγος είναι ότι πρόκειται για πολύπλοκες μορφές δεδομένων, συχνά με πολύ μεγάλη λεπτομέρεια στην απεικόνιση και που ωστόσο, είναι αποδεκτό ορισμένες φορές να μειώσουμε τη λεπτομέρεια μιας εικόνας με σκοπό πολύ μεγάλη μείωση του μεγέθους της. Κάτι τέτοιο είναι ιδιαίτερα χρήσιμο, τόσο για την αποθήκευση όσο και τη μετάδοση εικόνων, ειδικά όταν η διαθέσιμη μνήμη ή το διαθέσιμο έυρος ζώνης είναι περιορισμένα. Η ανάγκη αυτή έγινε ακόμα μεγαλύτερη με την ταχεία εξάπλωση του διαδικτύου και των νέων εφαρμογών που βασίζονται σε αυτό. Υπάρχουν δύο μορφές συμπίεσης, η **συμπίεση με απώλειες (lossy)** και η **συμπίεση χωρίς απώλειες (lossless)**.

Στη συμπίεση με απώλειες προσπαθούμε να κωδικοποιήσουμε μέρος της πλεονάζουσας πληροφορίας με τεχνικές οι οποίες εκ των πραγμάτων αλλοιώνουν το οπτικό αποτέλεσμα. Αυτό συμβαίνει γιατί η βασική αρχή κάθε μεθόδου απωλεστικής συμπίεσης είναι οι μείωση του συσχετισμού μεταξύ των τιμών των pixel μιας εικόνας. Ο κύριος λόγος για τον οποίο είναι εφικτή η συμπίεση μιας εικόνας είναι ο πολύ μεγάλος συσχετισμός που υπάρχει ανάμεσα σ' ένα pixel και τα γειτονικά του. Αυτό πρακτικά μεταφράζεται στο ότι οι τιμές ενός pixel και των γειτονικών του είναι παρόμοιες. Αν όμως μπορέσουμε να μειώσουμε τον συσχετισμό αυτό, είναι σχετικά

εύκολο να εκμεταλλευτούμε τα στατιστικά χαρακτηριστικά που παρουσιάζονται και τελικά να μειώσουμε το μέγεθος που καταλαμβάνει η εικόνα. Υπάρχουν πολλές τεχνικές που το πετυχαίνουν αυτό, όπως ο **διακριτός μετασχηματισμός συνημιτόνου (DCT)** που χρησιμοποιείται στο πρότυπο JPEG. Άλλες μέθοδοι απωλεστικής συμπίεσης είναι η **υποδειγματοληψία χρώματος (chrominance downsampling)** και η συμπίεση με **fractals**.

Σε ότι αφορά στη μη-απωλεστική συμπίεση, οι μέθοδοι που χρησιμοποιούνται είναι λιγότερο "επιθετικές" και πετυχαίνουν μικρότερου βαθμού συμπίεση. Παρόλα αυτά, η ποιότητα της εικόνας παραμένει άθικτη. Τέτοιες τεχνικές περιλαμβάνουν τη **διαφορική παλμοκωδική διαμόρφωση (DPCM)** και τον αλγόριθμο **DEFLATE**. Ειδικά ο τελευταίος αποτελεί το βασικό αλγόριθμο συμπίεσης του προτύπου PNG.

Όλα τα παραπάνω που αναφέρθηκαν αφορούν όπως είπαμε στην κωδικοποίηση της **πλεονάζουσας πληροφορίας**. Ως εκ τούτου, εισάγεται ένας νέος όρος, αυτός του **χωρικού πλεονασμού (spatial redundancy)** και ο οποίος αφορά στην επανάληψη συγκεκριμένης πληροφορίας (τιμές pixel στη συγκεκριμένη περίπτωση) μέσα σε μια δομή δεδομένων (την εικόνα).

## 2.2 Η Κωδικοποιηση Βιντεο

Συχνά, όταν αναφερόμαστε στο θέμα του βίντεο αμελούμε το γεγονός πως στην ουσία πρόκειται για αλληλουχία στατικών εικόνων. Οι εικόνες αυτές ονομάζονται **καρέ** ή **frames** και διαθέτουν όλα τα χαρακτηριστικά μιας κοινής στατικής εικόνας όπως ανάλυση, φωτεινότητα (luminance) και χρώμα (chrominance). Είναι λοιπόν προφανές, πως όσα αναφέραμε προηγουμένως για την κωδικοποίηση/συμπίεση εικόνας, μπορούν πολύ έυκολα να εφαρμοστούν στο βίντεο. Δυστυχώς, το βίντεο εμπεριέχει ορισμένες ιδιαιτερότητες που δυσκολεύουν την αποθήκευση και την επεξεργασία του ακόμα και μετά την εφαρμογή των προαναφερθεισών τεχνικών.

Σε ότι αφορά το βίντεο ως μορφή σήματος, υπάρχουν σημαντικές διαφορές σε σχέση με τη στατική εικόνα. Η βασικότερη όμως διαφορά εντοπίζεται στο ότι η στατική εικόνα περιλαμβάνει πληροφορία η οποία είναι σταθερή και αμετάβλητη ως προς το χρόνο, σε σχέση με το βίντεο που διαθέτει επιπλέον μια χρονική διάσταση. Το σήμα του βίντεο μεταβάλλεται ως προς το χρόνο με κάποιο συγκεκριμένο ρυθμό **καρέ ανά δευτερόλεπτο (frames per second ή FPS)**. Προκειμένου να είναι ορατό στο ανθρώπινο μάτι ως ομαλή συνεχής κίνηση και όχι ως διακοπτόμενη ακολουθία

εικόνων, θα πρέπει ο ρυθμός αυτός να είναι όσο το δυνατόν μεγαλύτερος. Παρακάτω παραθέτουμε έναν πίνακας (2.1) με διάφορα framerates και την αντίληψη κίνησης όπως προκύπτει από αυτά.

**Πίνακας 2.1**: Διάφορα framerates και το αποτέλεσμά τους

| **Framerate (FPS)** | **Αντίληψη κίνησης** |
|---|---|
| 10-12 | Το απόλυτο ελάχιστο για την αντίληψη συνεχόμενης κίνησης. Οτιδήποτε μικρότερο, γίνεται αντιληπτό ως ξεχωριστές εικόνες. |
| <16 | Η αλληλουχία των frames είναι ορατή. Μπορεί να ενοχλήσει πολλούς θεατές. |
| 24 | Το ελάχιστο ανεκτό για την αντίληψη μιας ομαλής κίνησης. Αποτελεί το κινηματογραφικό πρότυπο. |
| 30 | Πολύ καλύτερο σήμα σε σχέση με τα 24 FPS. Αποτελεί την καθιερωμένη πρακτική για το πρότυπο NTSC λόγω της συχνότητας του εναλλασσόμενου ρεύματος στις ΗΠΑ (60Hz), καθώς και την πλειοψηφία των ψηφιακών βίντεο. |
| 48 | Αυξημένη ποιότητα, που πλησιάζει την αντίληψη της κίνησης ως φυσική. |
| 60 | Ιδιαίτερα αυξημένη ποιότητα σε σχέση με τα προηγούμενα framerates. Οι περισσότεροι άνθρωποι αντιλαμβάνονται την κίνηση ως φυσική. |

Η περίπτωση των 30 FPS αποτελεί την πιο διαδεδομένη, ειδικά σε ότι αφορά το ψηφιακό βίντεο και τις υπηρεσίες που το προσφέρουν (YouTube), καθώς και τις συσκευές που το καταγράφουν/αναπαράγουν. Σε αυτήν την περίπτωση αυτό που ισχύει πρακτικά είναι ότι ανα 33ms περίπου, θα πρέπει να λαμβάνεται ένα καινούργιο frame.

Το γεγονός αυτό μας οδηγεί σε ενα πολύ σημαντικό συμπέρασμα: είναι πολύ περιορισμένη η κίνηση που μπορεί να υπάρξει και να καταγραφεί, είτε της κάμερας είτε του θέματος, μέσα σ' αυτό το παράθυρο των 33ms (ή ακόμα μικρότερο για μεγαλύτερα framerates). Επομένως, σε μεγάλο βαθμό υπάρχει πλεονάζουσα πληροφορία ανάμεσα σε γειτονικά frames. Με βάση την παρατήρηση αυτή, μπορούμε να εισάγουμε τον όρο του **χρονικού πλεονασμού (temporal redundancy)**, που υποδηλώνει τον πλεονασμό της πληροφορίας σε ένα σήμα που μεταβάλλεται ώς προς το χρόνο.

Στην περίπτωση του προτύπου Full-HD (Full High-Definition) κάθε καρέ αποτε-

λεί μια στατική εικόνα ανάλυσης 1920 × 1080 pixel. Με μερικούς απλούς υπολογισμούς διαπιστώνουμε ότι:

$$1920 \cdot 1080 \cdot 32\text{bpp} \approx 8\text{MBytes/frame}$$

και τελικά:

$$8 \cdot 30 \approx 240\text{MBytes/second}$$

Δηλαδή, κάθε δευτερόλεπτο βίντεο σε μορφή Full HD με ρυθμό 30 FPS, απαιτεί 240 MBytes μνήμης εφόσον είναι πλήρως ασυμπίεστο (σε μορφή RAW). Ακόμα και με "επιθετικούς" αλγόριθμους συμπίεσης εικόνας, το μέγεθος παραμένει πολύ μεγάλο και μας περιορίζει σημαντικά στην περίπτωση που θέλουμε να μεταδώσουμε το βίντεο στο Internet ή να αυξήσουμε την ποιότητά του. Ειδικά σε ότι αφορά το τελευταίο, τα πρότυπα 4K και 8K που τείνουν να καθιερωθούν απαιτούν αντίστοιχα τετραπλάσια και δεκαεξαπλάσια μνήμη ανα δευτερόλεπτο! Είναι προφανές λοιπόν, πως είναι απαραίτητο να εφαρμοστούν νέες τεχνικές συμπίεσης εξειδικευμένες πάνω στις αλληλουχίες εικόνων, που θα εκμεταλλεύονται τα ιδιαίτερα χαρακτηριστικά που αυτές παρουσιάζουν.

Όπως αναφέρθηκε και στην προηγούμενη ενότητα, στην περίπτωση των στατικών εικόνων εκμεταλλευόμαστε το χαρακτηριστικό του χωρικού πλεονασμού προκειμένου να επιτύχουμε συμπίεση. Ένας τρόπος για να το πετύχουμε αυτό είναι να μεταφέρουμε το σήμα της εικόνας από το πεδίο του χρόνου, στο πεδίο της συχνότητας χρησιμοποιώντας είτε το διακριτό μετασχηματισμό συνημιτόνου (DCT) είτε το διακριτό μετασχηματισμό wavelet (DWT). Με τις δυο αυτές μεθόδους αυτό που πετυχαίνουμε είναι να μειώσουμε τον συσχετισμό μεταξύ των τιμών των pixel μιας εικόνας, διατηρώντας την ενέργεια στις χαμηλές συχνότητες (η ενέργεια του σήματος για τις περισσότερες εικόνες βρίσκεται στις χαμηλότερες συχνότητες) και παραλείποντας τις υψηλές, με πολύ μικρή οπτική παραμόρφωση.

Αυτό που πρέπει πλέον να φροντίσουμε είναι η αποτελεσματική εκμετάλλευση του χρονικού πλεονασμού. Στο σημείο αυτό, ένα στοιχείο που μπορεί να μας βοηθήσει σημαντικά είναι η εμπειρία μας στην κωδικοποίηση του ήχου. Μια πολύ διαδεδομένη τεχνική είναι η διαφορική παλμοκωδική διαμόρφωση (DPCM) η οποία αποτελεί στην ουσία τεχνική πρόβλεψης. Με την DPCM καταφέρνουμε να προβλέψουμε την τιμή που θα έχει το σήμα μας σε μια δεδομένη χρονική στιγμή, βασιζόμενοι στις τιμές του σήματος σε προηγούμενες χρονικές στιγμές. Παρομοίως, αυτό που μπο-

ρούμε να κάνουμε στην περίπτωση του βίντεο είναι να προβλέψουμε την τιμή που έχουν τα pixel ενός frame βασιζόμενοι στις τιμές που είχαν τα γειτονικά τους frames (είτε προηγούμενα, είτε επόμενα).

Εφόσον διαθέτουμε δυο τεχνικές που η καθεμιά είναι εξειδικευμένη σε συγκεκριμένες μορφές σημάτων και εφόσον το σήμα του βίντεο μπορεί να θεωρηθεί ότι αποτελείται από δύο συνιστώσες, είναι φανερό ότι μπορούμε να δημιουργήσουμε ένα **υβριδικό σύστημα κωδικοποίησης/αποκωδικοποίησης (hybrid codec)** με το οποίο μπορούμε τελικά να πετύχουμε - θεωρητικά τουλάχιστον - ικανοποιητική κωδικοποίηση και συμπίεση ενός σήματος βίντεο. Εν κατακλείδι, θα έχουμε ένα σύστημα το οποίο θα χρησιμοποιεί τον μετασχηματισμό στο πεδίο των συχνοτήτων για την κωδικοποίηση του χωρικού πλεονασμου και μια τεχνική πρόβλεψης (όπως η DPCM) για την κωδικοποίηση του χρονικού πλεονασμού.

## 2.3 Το Υβριδικο Μοντελο

Οι τεχνικές πρόβλεψης όπως η DPCM μπορούν να χρησιμοποιηθούν τόσο για μεταβαλλόμενα όσο και για αμετάβλητα ως προς το χρόνο σήματα. Είναι ωστόσο αποδεδειγμένο ότι ο χωρικός πλεονασμός είναι πολύ πιο εύκολο να αξιοποιηθεί με τεχνικές μεταφοράς στο πεδίο των συχνοτήτων. Από την άλλη, ο χρονικός πλεονασμός είναι πολύ πιο έυκολα αξιοποιήσιμος από τεχνικές κωδικοποίησης με χρήση πρόβλεψης. Το ερώτημα που φυσικά γεννάται είναι, πως εφαρμόζεται η πρόβλεψη στα σήματα βίντεο;

Αν θεωρήσουμε πως δεχόμαστε το σήμα βίντεο ως είσοδο διαδοχικών frames, τότε αυτά τα δεχόμαστε με κάποια σειρά και αναγκαστικά θα καταφθάνουν σε χρονικές στιγμές **t**, **t+1**, **t+2**, κλπ. Αν το σύστημά μας διαθέτει μνήμη ωστε να είναι δυνατόν να αποθηκεύεται ένα frame **t**, τότε είναι εφικτό να το χρησιμοποιήσουμε ωστε να προβλέψουμε ποιο θα είναι το επόμενο frame, **t+1**. Είναι σαφές ότι η πρόβλεψη, με όποιον μηχανισμό και αν γίνει, επειδή βασίζεται αποκλειστικά σε "παρελθοντικό" frame δεν πρόκειται να είναι απόλυτα ακριβής, επομένως θα περιέχει κάποιο σφάλμα. Αυτό το **σφάλμα πρόβλεψης**, όπως ονομάζεται, παρουσιάζει επίσης ένα βαθμό χωρικού πλεονασμού. Αυτό συμβαίνει γιατί όπως θα διαπιστώσουμε στη συνέχεια, σε ότι αφορά το σήμα του σφάλματος πρόβλεψης, τα διαδοχικά pixels στο χώρο παρουσιάζουν μεγάλο βαθμό ομοιότητας αφού όπως ήδη εξηγήσαμε, οι αλλαγές στο frame είναι πολύ μικρές. Αυτός ο πλεονασμός λοιπόν, είναι δυνατόν να

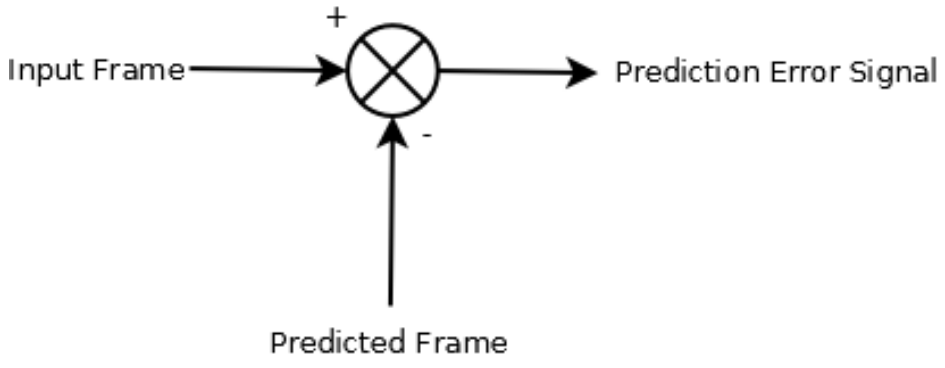
αξιοποιηθεί με κάποια τεχνική μεταφοράς στο πεδίο των συχνοτήτων όπως η DCT.

Τελικά το υβριδικό μοντέλο που περιγράψαμε χρησιμοποιεί:

- **Κωδικοποίηση πρόβλεψης (predictive coding)**, για την αξιοποίηση του χρονικού πλεονασμού
- **Κωδικοποίηση με μεταφορά στο πεδίο συχνοτήτων (transform-domain coding)**, για την αξιοποίηση του χωρικού πλεονασμού του σφάλματος πρόβλεψης

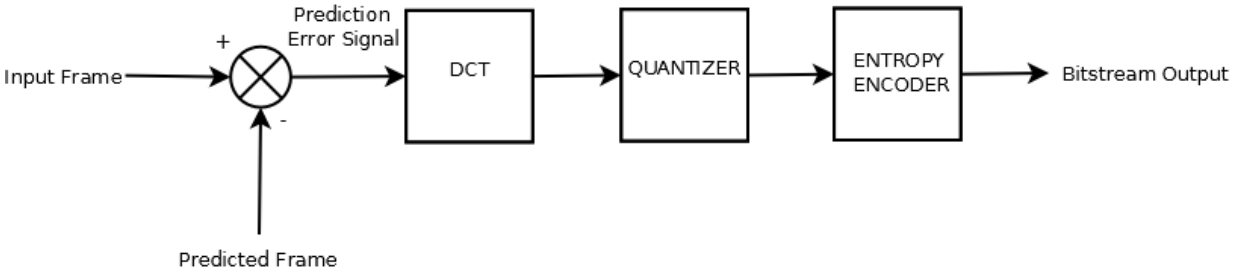
## 2.4 Σχεδιασμος του Συστηματος Κωδικοποιησης

Σύμφωνα με τα όσα περιγράψαμε παραπάνω είναι σχετικά εύκολο να υποθέσουμε τη δομή του συστήματος που θα αποτελεί τον κωδικοποιητή του σήματος βίντεο. Τα απολύτως απαραίτητα συστατικά μέρη περιλαμβάνουν πέρα από την είσοδο/έξοδο του συστήματος, έναν αφαιρέτη, έναν μηχανισμό μεταφοράς στο πεδίο της συχνότητας, ένα μηχανισμό πρόβλεψης και μια μνήμη, στην οποία θα αποθηκεύεται κάποιο ή κάποια frames. Αρχικά, η είσοδος του συστήματός μας οδηγείται στον αφαιρέτη, μαζί με το frame που έχουμε προβλέψει. Το τελευταίο αφαιρείται από το σήμα εισόδου και το αποτέλεσμα που εξάγεται αποτελεί το σφάλμα πρόβλεψης ή **υπόλοιπο**. Το σχήμα 2.1 δείχνει ακριβώς αυτή τη δομή. Στο σημείο αυτό αξίζει να σημειώσουμε πως εσκεμμένα δεν αναφερόμαστε περαιτέρω στη διαδικασία της πρόβλεψης, καθώς θα ασχοληθούμε εκτενώς με αυτήν σε επόμενη ενότητα.




**Σχήμα 2.1**: Το διάγραμμα του αφαιρέτη

Όπως αναφέραμε και προηγουμένως, το σφάλμα πρόβλεψης είναι αυτό το οποίο θα κωδικοποιηθεί χρησιμοποιώντας τη μεταφορά στο πεδίο των συχνοτήτων. Επομένως, στο επόμενο στάδιο, θα πρέπει να τροφοδοτήσουμε το σφάλμα εισόδου σε μια μονάδα DCT (διακριτού μετασχηματισμού συνημιτόνου). Ο λόγος που χρησιμοποιούμε τον DCT και όχι κάποια άλλη τεχνική, όπως η DWT, είναι ότι είναι αρκετά διαδεδομένη σε πολλά σύγχρονα πρότυπα συμπίεσης (ως DCT-II). Αφού το σήμα μεταφερθεί στο πεδίο των συχνοτήτων θα πρέπει στη συνέχεια να **κβαντιστεί** και να εφαρμοστεί **κωδικοποίηση εντροπίας** έπειτα από την οποία θα λάβουμε το τελικό bitstream. Το σχήμα 2.2 δείχνει το σύστημα μέχρι τώρα με τα συστατικά του στοιχεία.




**Σχήμα 2.2**: Το διάγραμμα του συστήματος μετά την κωδικοποίηση του σφάλματος πρόβλεψης

Είναι απαραίτητο να τονίσουμε ότι με τον τρόπο αυτό δεν κωδικοποιούμε το πραγματικό σήμα, αλλά το υπόλοιπο, το σφάλμα πρόβλεψης που είναι μόνο μια συνιστώσα του "χρήσιμου" σήματος, αυτού δηλαδή που θα εμφανίσουμε τελικά στην οθόνη. Αυτό που πρέπει να γίνει προκειμένου να πάρουμε ξανά ένα ολοκληρωμένο frame είναι να προστεθεί το σφάλμα πρόβλεψης με το predicted frame, κάτι που είναι αδύνατον σε επίπεδο αποκωδικοποιητή. Ο λόγος για τον οποίο συμβαίνει αυτό είναι ότι ο αποκωδικοποιητής δεν έχει γνώση κανενός άλλου σήματος πέραν του σφάλματος πρόβλεψης. Μπορούμε να θεωρήσουμε την είσοδο του συστήματος ως μια συνάρτηση τριών μεταβλητών:

$$s(n_1, n_2, k)$$

όπου s η ένταση (intensity) των pixel στη θέση ($n_1$, $n_2$) κατά τη χρονική στιγμή t. Είναι σημαντικό να γνωρίζουμε πως οι τρεις αυτές μεταβλητές παίρνουν ακέραιες τιμές, επομένως η μεταβλητή t συμβολίζει τον αριθμό του frame. Αντίστοιχα

μπορούμε να αναπαραστήσουμε το predicted frame ως:

$$\hat{s}(n_1, n_2, k)$$

και το σφάλμα πρόβλεψης ως:

$$e(n_1, n_2, k)$$

το οποίο μετά τον κβαντιστή αναπαριστάται ως:

$$\dot{e}(n_1, n_2, k)$$

Τελικά ο αρχικός ισχυρισμός μας για την ανακατασκευή ενός ολοκληρωμένου frame εκφράζεται ως:

$$\dot{e}(n_1, n_2, k) + \hat{s}(n_1, n_2, k) = \dot{s}(n_1, n_2, k)$$

Αξίζει να σημειώσουμε ότι $\dot{s} \neq s$ όπως επίσης $\dot{e} \neq e$. Οι τελείες εκφράζουν το σήμα αφού έχει υποστεί απώλειες λόγω της δειγματοληψίας. Ως εκ τούτου, και το frame το οποίο ανακατασκευάζεται με άθροιση του κβαντισμένου σφάλματος πρόβλεψης και του predicted frame, δεν είναι το ίδιο με εκείνο που δεχτήκαμε στην είσοδο του συστήματος.

Αναφερθήκαμε ήδη στο μηχανισμό πρόβλεψης χωρις ιδιαίτερες λεπτομέρειες, πέραν του ότι θα αναπαράγει το predicted frame βασιζόμενος σε παρελθοντικά frames. Τα παρελθοντικά frames γενικά τη μορφή: $s(n_1, n_2, k - L)$, όπου $L > 0$. Αυτό που δεν αναλύσαμε ακόμη είναι το πώς χρησιμοποιούμε τα παρελθοντικά frames προκειμένου να παράξουμε τα predicted frames.

Είναι απλό κάποιος να ισχυριστεί ότι η παραγωγή του predicted frame μπορεί να γίνει σχετικά εύκολα, αρκεί να χρησιμοποιήσουμε την εξής διαδικασία:

- Έστω ότι λαμβάνουμε ένα frame t-1 στην είσοδο του συστήματός μας, το οποίο αποθηκεύουμε σε κάποια μνήμη.
- Στη συνέχεια λαμβάνουμε ένα νέο frame t.
- Συγκρίνουμε τα δύο frames μεταξύ τους υπολογίζοντας την μετατόπιση του κάθε block στα οποία έχουμε διαχωρίσει τα frame (εκτίμηση κίνησης).

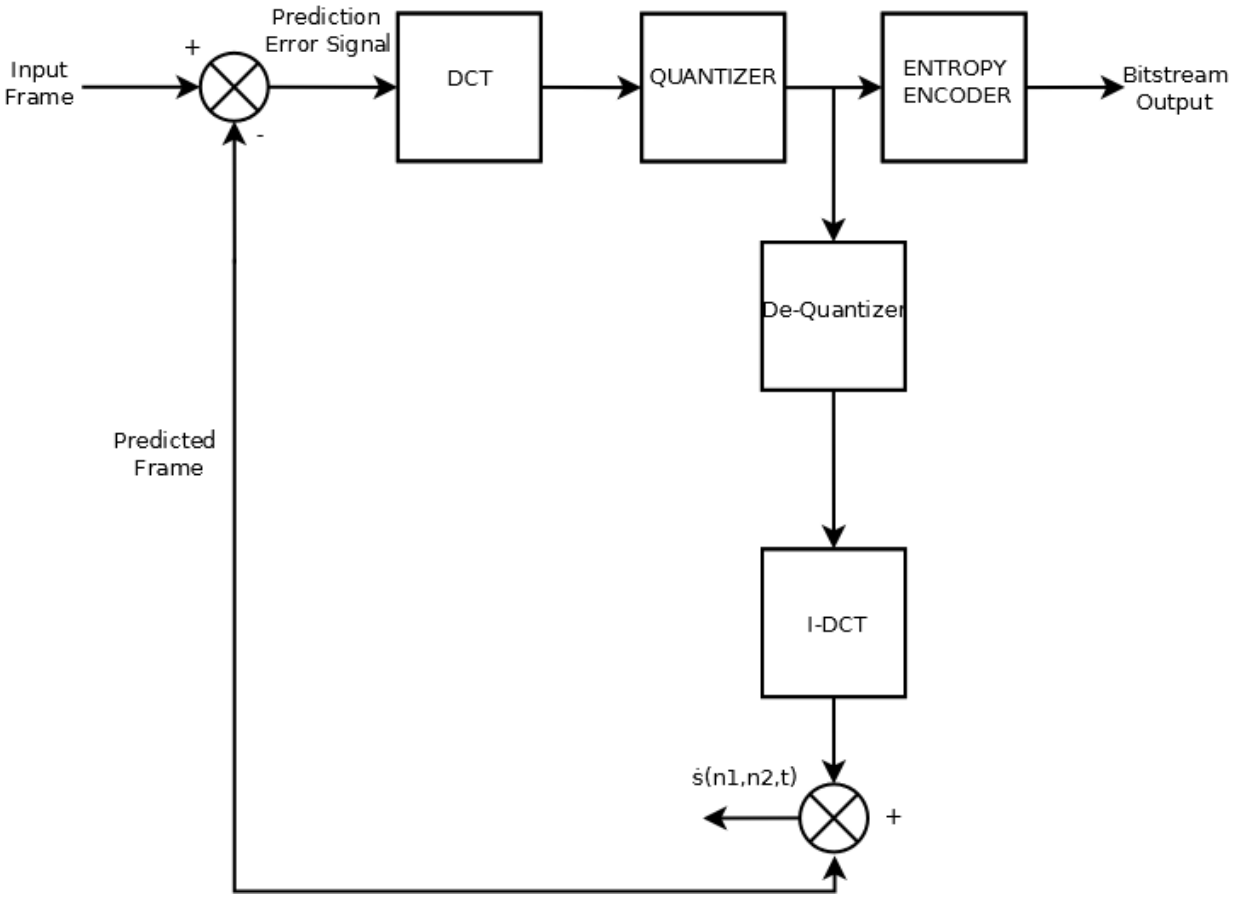
- Τελικά με κάποιο μηχανισμό (που τον θεωρούμε ως ''μαύρο κουτί'' για λόγους απλότητας) κάνουμε την πρόβλεψη κίνησης και παίρνουμε το predicted frame.

Η προσέγγιση αυτή, αν και αρκετά απλή και εφικτή στα πλαίσια του κωδικοποιητή, περιέχει μια εγγενή αδυναμία. Ο μηχανισμός πρόβλεψης πρέπει να είναι **ο ίδιος** στον κωδικοποιητή και τον αποκωδικοποιητή. Με αυτόν τον τρόπο και μόνο, μπορούμε να είμαστε σίγουροι ότι ο αποκωδικοποιητής θα μπορέσει να αναπαράγει την ακριβή πληροφορία που του δώσαμε. Στη διαδικασία που περιγράψαμε παραπάνω έχουμε υπολογίσει λανθασμένα το residue, αφού τελικά το predicted frame δεν περιλαμβάνει τις απώλειες του κβαντιστή. Ακριβώς λοιπόν επειδή ο αποκωδικοποιητής δεν θα έχει στη διάθεσή του άλλα σήματα, παρά μόνον το κβαντισμένο σφάλμα πρόβλεψης, δεν θα είναι σε θέση να αναπαράγει σωστά το ζητούμενο frame.

Άμεση απόρροια αυτής της παρατήρησης είναι η αλλαγή της πηγής του σήματος με βάση το οποίο θα γίνει η πρόβλεψη. Αυτό σημαίνει, ότι το παρελθοντικό frame δε θα το πάρουμε απευθείας από την είσοδο του συστήματος, αλλά θα πρέπει να το ανακατασκευάσουμε, κάνοντας χρήση του σφάλματος πρόβλεψης. Το σήμα που αφορά το σφάλμα πρόβλεψης θα προέρχεται από την έξοδο του κβαντιστή ($\dot{e}(n_1, n_2, t)$) και θα τροφοδοτείται απευθείας σε έναν **απο-κβαντιστή (de-quantizer)** ενώ στη συνέχεια, θα υποστεί **Αντίστροφο Διακριτό Μετασχηματισμό Συνημιτόνου (Inverse DCT - IDCT)**. Τελικά θα προστεθεί με το αντίστοιχο predicted frame ($\hat{s}(n_1, n_2, t)$), ώστε να μας δώσει το ανακατασκευασμένο frame $\dot{s}(n_1, n_2, t)$ (Σχήμα 2.3).

Αν το ανακατασκευασμένο frame το αποθηκεύσουμε σε μια μνήμη για μετέπειτα χρήση του, τότε αυτομάτως αυτό γίνεται παρελθοντικό και επομένως μπορούμε να το χρησιμοποιήσουμε για να προβλέψουμε ποιο θα είναι το επόμενο frame. Στην περίπτωση αυτή και για να είμαστε σύμφωνοι με τη σημειολογία που εισάγαμε προηγουμένως, θα αναφερόμαστε πλέον στο frame αυτό ως $\dot{s}(n_1, n_2, t-1)$ ενώ στο frame που μόλις δεχτήκαμε στην είσοδο ως $s(n_1, n_2, t)$.

Το τελευταίο κομμάτι του κωδικοποιητή είναι ο **μηχανισμός πρόβλεψης**. Είναι ίσως το σημαντικότερο κομμάτι του κωδικοποιητή αφού είναι αυτό που εκμεταλλεύεται τον χρονικό πλεονασμό μεταξύ των frames προκειμένου να πετύχουμε περαιτέρω συμπίεση. Ο μηχανισμός πρόβλεψης αποτελείται από τρία επιμέρους τμήματα: τη μνήμη, η οποία περιέχει τα παρελθοντικά frames, τη μονάδα εκτίμησης κίνησης (motion estimation) και τη μονάδα πρόβλεψης (motion prediction). Για τη μνήμη

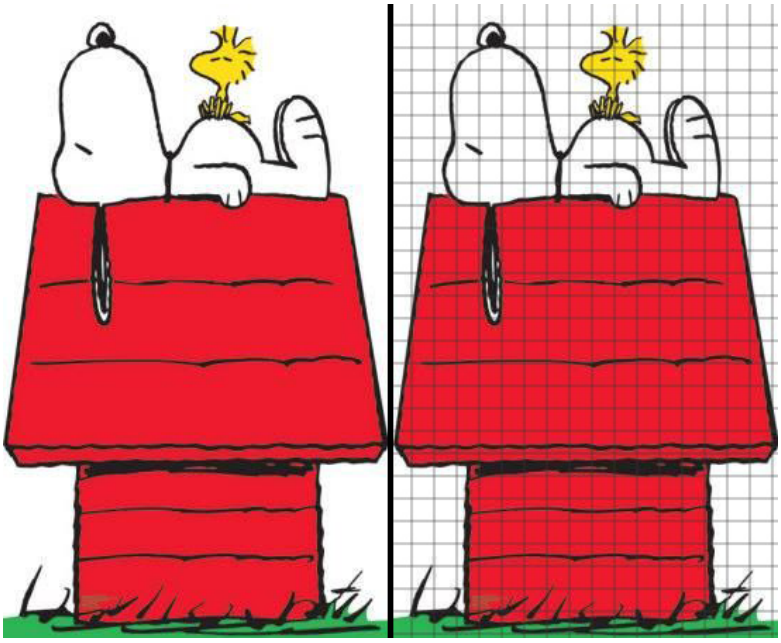



**Σχήμα 2.3**: Το διάγραμμα του συστήματος μετά την ανακατασκευή του frame

έχουμε ήδη αναφέρει ορισμένα πράγματα και η ανάλυσή της μπορεί να ολοκληρωθεί εδώ αφού δεν εκτελεί κάποια περίπλοκη εργασία.

Στο σημείο αυτό μπορεί να γίνει μια αναφορά στα blocks μιας εικόνας (ή ενός frame). Πρόκειται ουσιαστικά για έναν ευρύτερο διαχωρισμό της εικόνας σε μεγαλύτερες περιοχές που περιλαμβάνουν (και ομαδοποιούν) πολλά pixel. Η λογική τους είναι παρόμοια με αυτή των οικοδομικών τετράγωνων (blocks) στις πόλεις, αφού έχουν ακριβώς την ίδια δομή πλέγματος. Συνήθως οι διαστάσεις των block (σε pixels) είναι δυνάμεις του 2 όπως, 4x4, 8x8, 8x4, 16x16, 16x8 κλπ. Στην εικόνα 2.4 φαίνεται ο διαχωρισμός μιας εικόνας σε block.

Η μόναδα εκτίμησης κίνησης έχει δύο εισόδους και μια έξοδο. Πρακτικά, η δουλειά της είναι να συγκρίνει το παρελθοντικό frame σε σχέση με το τρέχον που

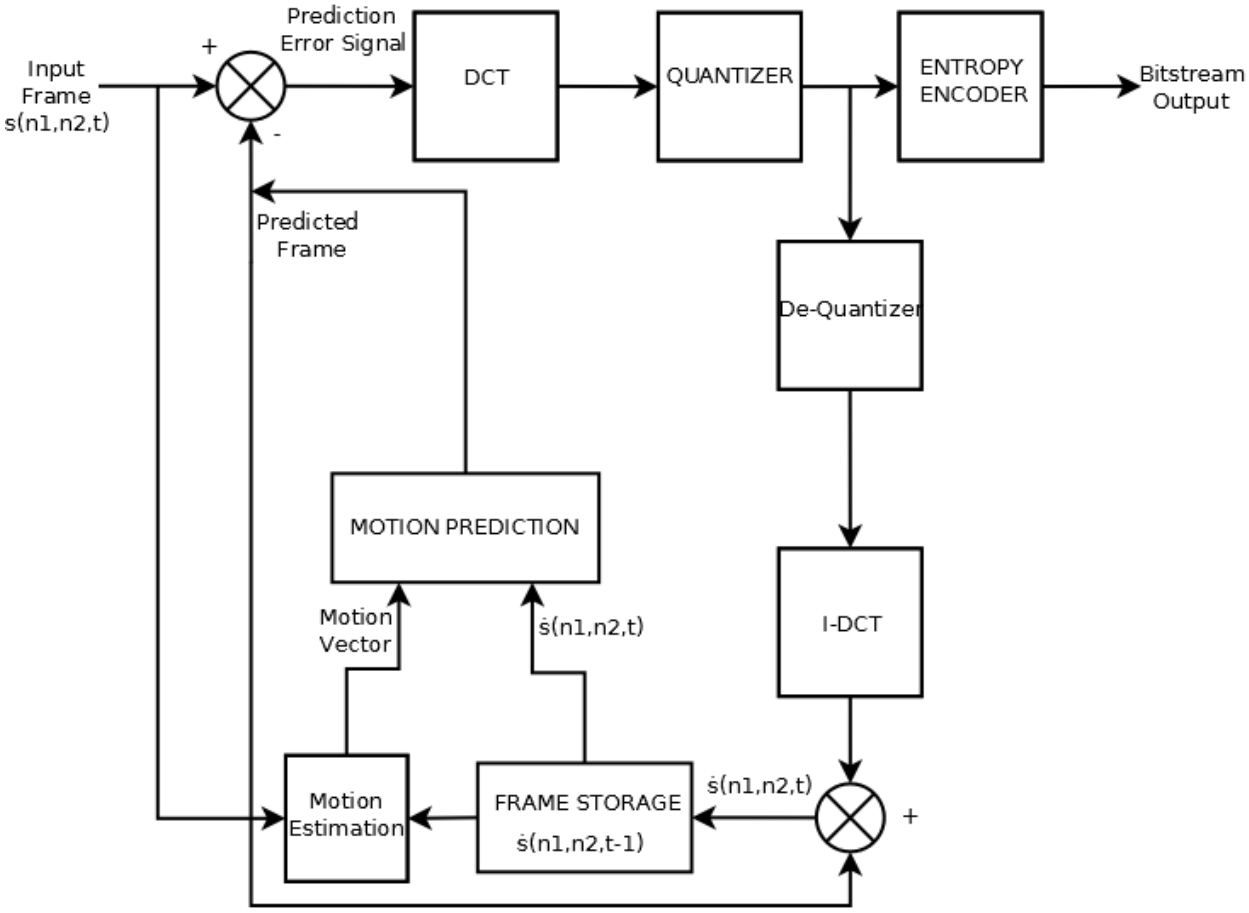


**Σχήμα 2.4**: Διαχωρισμός εικόνας σε block

λαμβάνουμε στην είσοδο του συστήματος και να υπολογίζει τη μετατόπιση που υπέστησαν τα blocks από το ένα frame στο επόμενο. Το αποτέλεσμα αυτής της σύγκρισης είναι ένα πλήθος διανυσμάτων που υποδεικνύουν τη μετατόπιση των blocks και ονομάζονται **διανύσματα μετατόπισης (displacement vectors)** ή **διανύσματα κίνησης (motion vectors)**. Όταν τα διανύσματα κίνησης εφαρμοστούν πάνω σ' ένα παρελθοντικό frame, τότε αυτό που προκύπτει είναι μια **πρόβλεψη κίνησης** που στην περίπτωσή μας είναι το predicted frame. Τα διανύσματα κίνησης θα τα δούμε αναλυτικά και θα μας απασχολήσουν στην επόμενη ενότητα αλλά και σε όλη την υπόλοιπη έκταση της εργασίας. Λάμβάνοντας υπόψην τα όσα αναφέραμε περί του μηχανισμού πρόβλεψης, το τελικό σχεδιάγραμμα του κωδικοποιητή έχει ως εξής (Σχήμα 2.5):

## 2.5 Εκτιμηση Κινησης (Motion Estimation)

Προηγουμένως αναφερθήκαμε στην εκτίμηση κίνησης και τη μονάδα που την υλοποιεί, σ' έναν τυπικό κωδικοποιητή βίντεο. Παραλείψαμε την αναφορά σε λεπτομέρειες και τη σε βάθος ανάλυση για δύο λόγους: πρώτον, διότι πρόκειται για ένα πολύ



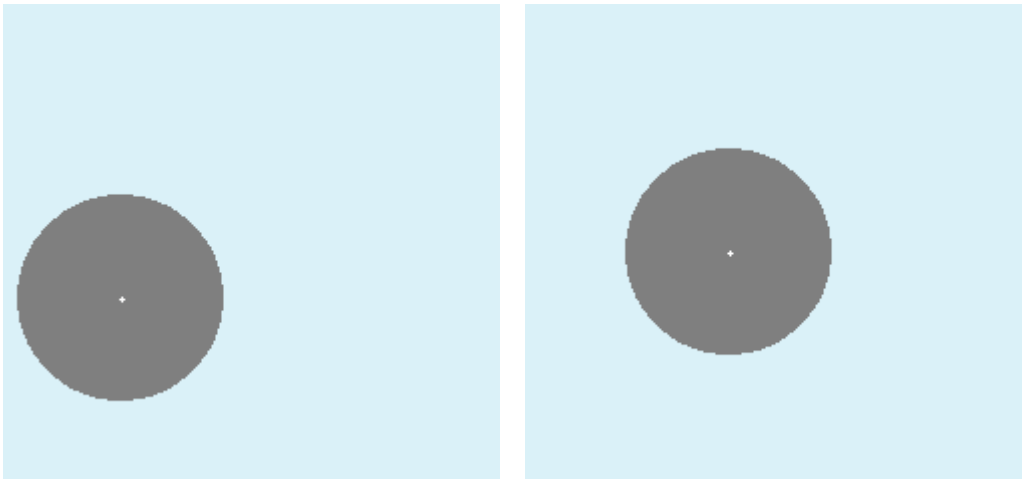

**Σχήμα 2.5**: Τελικό διάγραμμα του κωδικοποιητή

σημαντικό κομμάτι το οποίο είναι σκόπιμο να αναλυθεί ξεχωριστά και να καλυφθεί πλήρως, και δεύτερον, επειδή πρόκειται για τον πυρήνα της εργασίας μας, για τον οποίο μάλιστα έγινε η εισαγωγή στις βασικές αρχές της κωδικοποίησης βίντεο.

Αναφέραμε πολλές φορές ότι μεταξύ διαδοχικών frames υπάρχει πολύ μεγάλος συσχετισμός, διότι είναι εκ των πραγμάτων πολύ περιορισμένος ο αριθμός των συμβάντων που μπορούν να καταγραφούν μέσα στο χρονικό παράθυρο των 33ms (για τυπικούς ρυθμούς 30 καρέ/δευτερόλεπτο). Ακριβώς γι' αυτό το λόγο αν συγκρίνουμε δύο διαδοχικά frame **t-1** και **t** θα διαπιστώσουμε ότι σε συγκεκριμένες περιοχές η ένταση (intensity) των pixel παραμένει η ίδια, ενώ σε άλλες αλλάζει. Η πρώτη περίπτωση ισχύει σε περιοχές του frame όπου δεν υπάρχει κίνηση, ή η κίνηση είναι ανεπαίσθητη, τόσο που να μην είναι ορατή από frame σε frame (για παράδειγμα

ένα στατικό φόντο). Αντίθετα, στα σημεία του frame όπου εντοπίζεται κίνηση (για παράδειγμα κινήσεις χεριών) οι αλλαγές στην ένταση μπορεί να είναι μεγάλες.

Αν υποθέσουμε ότι έχουμε δύο διαδοχικά frames στα οποία παρουσιάζεται η κίνηση ενός απλού αντικειμένου (σχήμα 2.6) η κίνηση αυτή, εφόσον είναι μεταφορική, μπορεί να περιγραφεί από δύο συνιστώσες, μία κατά τον άξονα $\times$ και μία κατά τον άξονα y. Παραμένοντας συνεπείς στην προηγούμενη σημειολογία, θεωρούμε τις δυο διαστάσεις ως $n_1$ και $n_2$, οπότε τελικά η μεταφορική αυτή κίνηση μπορεί να οριστεί ως ένα δισδιάστατο διάνυσμα $\mathbf{d} = (d_1, d_2)^T$ όπου $d_1$ η μετατόπιση ως προς $n_1$ και $d_2$ η μετατόπιση ως προς $n_2$. Το διάνυσμα αυτό ονομάζεται **διάνυσμα μετατόπισης (displacement vector)** ή **διάνυσμα κίνησης (motion vector)**. Συγκρίνοντας



**Σχήμα 2.6**: Κίνηση ενός απλού γεωμετρικού σχήματος ανάμεσα σε δυο διαδοχικά frames

τα δύο frame sto σχήμα 2.6, θα παρατηρήσουμε ότι υπάρχουν περιοχές όπου η μετατόπιση είναι μηδενική και άλλες όπου είναι μη μηδενική. Πως όμως μπορεί να γίνει μια τέτοια σύγκριση; Μια συνήθης τακτική είναι η **αντιστοίχιση (matching)** του προηγούμενου frame με το επόμενο. Υπάρχουν αρκετοί τρόποι για να γίνει αυτή η αντιστοίχιση και μια απλή σκέψη θα ήταν να γίνει pixel προς pixel. Με αυτό τον τρόπο θα αντιστοιχίζαμε την τιμή κάθε ενός pixel του πρώτου frame με το δεύτερο, μέχρι να βρούμε όλες τις αλλαγές που συνέβησαν. Κάτι τέτοιο ωστόσο εκτός από εξαιρετικά χρονοβόρο δεν είναι καθόλου πρακτικό, διότι με το να συγκρίνουμε την ένταση ενός pixel με την ένταση του αντίστοιχου pixel ενός άλλου frame δε μας εξασφαλίζει ότι όντως θα υπάρχει αντιστοίχιση μεταξύ τους, αφού λόγω σφάλματος ή τύχης μπορεί να έχουν την ίδια ακριβώς τιμή. Για το λόγο αυτό ελέγχουμε όχι

μόνο την αντιστοίχιση μεταξύ δύο pixel αλλά και μεταξύ των γειτονικών τους. Μόνο όταν δύο τέτοιες "γειτονιές" μεταξύ δύο frame ταιριάζουν μεταξύ τους, μπορούμε να ισχυριστούμε με σιγουριά ότι έγινε σωστά η αντιστοίχιση.

Ένα σημαντικό ζήτημα είναι ο τρόπος με τον οποίο θα καθοριστούν αυτές οι "γειτονιές" με βάση τα γεωμετρικά τους χαρακτηριστικά. Μπορούμε να θεωρήσουμε επί παραδείγματι, αυθαίρετα, κυκλικές, πολύγωνες ή τετράγωνες περιοχές. Η απλούστερη εκδοχή είναι αυτή των τετράγωνων ή/και ορθογώνιων περιοχών. Μάλιστα, αφού είμαστε ήδη εξοικιωμένοι με το διαμερισμό μιας εικόνας σε μικρότερα μη-αλληλοεπικαλυπτόμενα block μπορούμε να εφαρμόσουμε ακριβώς την ίδια φιλοσοφία και εδώ, να χωρίσουμε δηλαδή την εικόνα σε διακριτά block και τελικά να υπολογισουμε τη μετατόπιση που υπέστη κάθε block (και συνεπώς τα διανύσματα κίνησης). Η διαδικασία εύρεσης του βαθμού μετατόπισης είναι ουσιαστικά μια **τεχνική αναζήτησης** η οποία είναι περισσότερο γνωστή ως **εκτίμηση κίνησης (motion estimation)**. Πλεόν είναι πιο ξεκάθαρος ο τρόπος με τον οποίο λειτουργεί η μονάδα εκτίμησης κίνησης που περιγράψαμε παραπάνω.

Από την πλευρά του αποκωδικοποιητή, τα διανύσματα κίνησης είναι απολύτως απαραίτητη πληροφορία αφού θα χρησιμοποιηθούν πάνω σε παρελθοντικά (ή μελλοντικά, ανάλογα τη μορφή κωδικοποίησης) frame προκειμένου να αναπαράγουν τα ζητούμενα frames κατά την αποκωδικοποίηση του βίντεο. Συνεπώς, είναι απαραίτητο να εφαρμόσουμε την κωδικοποίηση εντροπίας και σε αυτά ώστε να συμπεριληφθούν στο τελικό bitstream (την έξοδο του κωδικοποιητή).

Όπως είναι εύκολα αντιληπτό, η εκτίμηση κίνησης οφείλει να είναι μια γρήγορη, ακριβής και αποτελεσματική διαδικασία. Ο λόγος για τον οποίο πρέπει να έχει αυτά τα χαρακτηριστικά είναι ο πολύ μεγάλος όγκος πληροφοριών που καλείται να επεξεργαστεί, συχνά σε πραγματικό χρόνο. Κατά την εκτίμηση κίνησης θα πρέπει να βρούμε τα διανύσματα κίνησης για κάθε ένα block του frame και δεν γνωρίζουμε εκ των προτέρων προς ποια κατεύθυνση έχει κινηθεί αυτό το συγκεκριμένο block. Η κατάσταση περιπλέκεται ακόμη περισσότερο αν εισάγουμε και άλλες μορφές κίνησης εκτός από τη μεταφορική, π.χ περιστροφική. Προκειμένου να κάνουμε την αντιστοίχιση θα πρέπει να προχωρήσουμε με έναν **αλγόριθμο αναζήτησης** και η αναζήτηση είναι εκ των πραγμάτων, μια υπολογιστικά πολύπλοκη διαδικασία. Στην περίπτωση ενός frame που έχει διαμεριστεί σε blocks των 8x8 pixel, για κάθε θέση αναζήτησης (block), θα πρέπει να συγκρίνουμε τις τιμές 64 pixel. Φυσικά, όπως αναφέραμε ήδη πολλές φορές, η κίνηση μεταξύ γειτονικών frames είναι αρκετά

περιορισμένη επομένως η αναζήτηση αυτή δεν είναι υποχρεωτικό να λάβει χώρα σε ολόκληρο το frame αλλά μόνο σε μια περιοχή του. Αυτή την περιοχή την ονομάζουμε **περιοχή αναζήτησης (search area)**, και αν και σημαντικά μικρότερη σε έκταση, η αναζήτηση σε αυτήν εξακολουθεί να αποτελεί υπολογιστική πρόκληση.

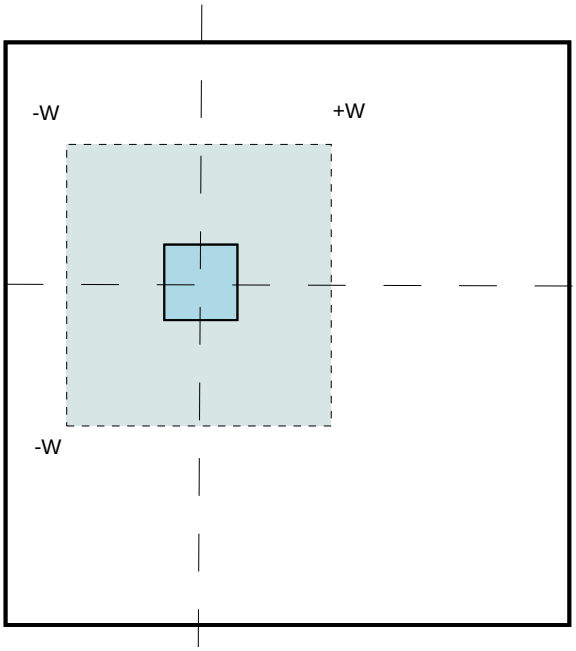
Κατα τη διαδικασία της αναζήτησης αυτό που προσπαθούμε να πετύχουμε είναι για κάθε ένα block του candidate frame (candidate block) να βρούμε με ποιο block του reference frame (reference block) έχει την καλύτερη αντιστοίχιση. Όπως αναφέραμε και προηγουμένως, η αναζήτηση δε γίνεται σε όλα τα blocks του frame αλλά σε μια μικρότερη περιοχή αναζήτησης. Αυτό που είναι προφανές είναι πως για να μπορέσουμε να είμαστε ακριβείς ως προς την αντιστοίχιση δύο block, πρέπει πρώτα να ορίσουμε ένα μέτρο, ένα **κριτήριο ταύτισης**. Στην εκτίμηση κίνησης βασισμένη σε block **(block-based motion estimation)** τα κριτήρια είναι τρία:

- Μέσο Τετραγωνικό Σφάλμα (Mean Square Error ή **MSE**)
- Μέση Απόλυτη Διαφορά (Mean of Absolute Difference ή **MAD**)
- Αριθμός Αντιστοιχισμένων Pixel (Matching Pixel Count ή **MPC**)

### 2.5.1 Μεσο Τετραγωνικο Σφαλμα

Ας υποθέσουμε και πάλι ένα candidate frame με συνάρτηση έντασης $s(n_1, n_2, t)$ και ένα reference frame με συνάρτηση έντασης $s(n_1, n_2, t-l), l > 0$. Το reference frame θα είναι παρελθοντικό, αφού το $l$ είναι μεγαλύτερο του μηδενός. Ας υποθέσουμε επίσης ένα candidate block για το οποίο θέλουμε να αναζητήσουμε ποια ακριβώς ήταν η θέση του στο reference frame. Για το λόγο αυτό ορίζουμε μια περιοχή αναζήτησης γύρω από το block, με διαστάσεις $\pm w$ pixel.

Προκειμένου να καλύψουμε ολόκληρη την περιοχή αναζήτησης, θα πρέπει να ψάξουμε συνολικά σε $(2w+1)\cdot(2w+1)$ θέσεις. Ο αριθμός $w$ πρακτικά, δεν μπορεί να είναι πολύ μικρός ούτε πολύ μεγάλος. Ας μην ξεχνάμε ότι η κίνηση των φυσικών αντικειμένων είναι πολύ περιορισμένη στο χρονικό παράθυρο των 33ms (για βίντεο 30 FPS), επομένως μια ρεαλιστική τιμή θα ήταν $w = 7$ ή $w = 8$. Αν και εκ πρώτης όψεως μοιάζει μικρός αριθμός, ο συνολικός αριθμός των θέσεων αναζήτησης που προκύπτουν είναι: $(2w+1)^2 = (2\cdot 7+1)^2 = 225$ και απαιτούνται για την εκτίμηση



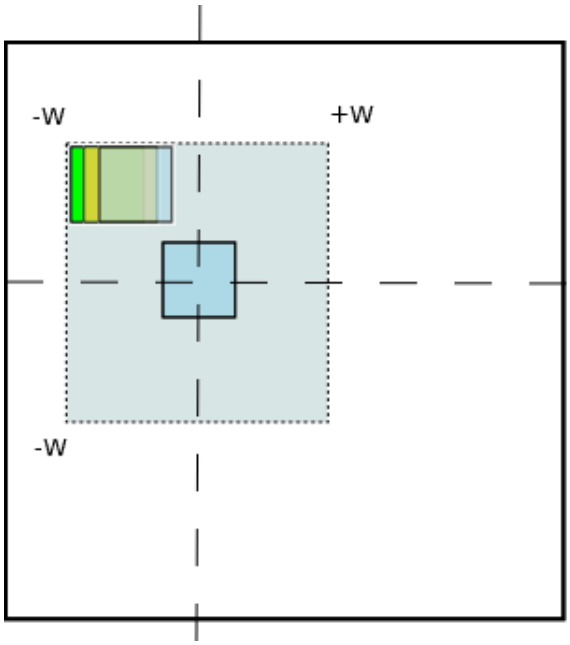

**Σχήμα 2.7**: Το candidate frame στο οποίο φαίνεται με γαλάζιο χρώμα το candidate block και η περιοχή αναζήτησης που το περιβάλλει

της κίνησης ενός μόνο block! Αν συνυπολογίσουμε το γεγονός ότι η πρόβλεψη κίνησης πρέπει να γίνεται σε πραγματικό χρόνο, τότε είναι περισσότερο από προφανής η ανάγκη για γρήγορους χρόνους επεξεργασίας.

Ο τρόπος με τον οποίο πραγματοποιούμε την αναζήτηση είναι εφαρμόζοντας μια "τεχνητή" μετατόπιση του candidate block μέσα στην περιοχή αναζήτησης και ελέγχοντας βάσει των κριτηρίων που προαναφέραμε τη συσχέτιση με την τρέχουσα θέση (το reference block), "σαρώνοντας" πρακτικά όλη την περιοχή αναζήτησης pixel-προς-pixel.

Η επεξεργαστική ισχύς καταναλώνεται σχεδόν αποκλειστικά στην εφαρμογή των κριτηρίων ταύτισης και μία περίπτωση είναι η χρήση της μεθοδολογίας του Μέσου Τετραγωνικού Σφάλματος ή MSE. Αν θεωρήσουμε ως $(i, j)$ τη μετατόπιση του candidate block μέσα στην περιοχή αναζήτησης, ο μαθηματικός ορισμός του MSE έχει ως εξής:

$$\mathrm{MSE}(i,j) = \frac{1}{N^2} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} [s(n_1, n_2, k) - s(n_1 + i, n_2 + j, k - l)]^2 \qquad (2.1)$$




**Σχήμα 2.8**: Η διαδικασία σάρωσης της περιοχής αναζήτησης. Διακρίνονται οι τρεις πρώτες θέσεις αναζήτησης.

όπου N ή διάσταση του block (στην προκειμένη περίπτωση $N \times N$ είναι οι διαστάσεις του block μας)

Κατά την αναζήτηση μέσα στην περιοχή που ορίσαμε, η θέση $(i, j)$ μεταβάλλεται συνεχώς. Για κάθε νέα θέση κάνουμε μια σύγκριση, βρίσκοντας στην πράξη τη διαφορά μεταξύ των τιμών έντασης των pixel του candidate block με κάθε ένα reference block. Επειδή όμως αυτή η διαφορά μπορεί να είναι είτε θετική είτε αρνητική, αυτό μπορεί να οδηγήσει σε απροσδόκητα αποτελέσματα όπως για παράδειγμα να εμφανιστεί η συνολική διαφορά μεταξύ των blocks ως μηδενική, αν για παράδειγμα οι επιμέρους διαφορές των pixel έχουν τέτοιες αρνητικές και θετικές τιμές που να ευνοούν ένα τέτοιο αποτέλεσμα. Αυτός είναι και ο ρόλος που παίζει το τετράγωνο στη σχέση, αφού απαλείφει ουσιαστικά τις αρνητικές τιμές. Η παραπάνω μαθηματική σχέση δηλαδή, μεταφράζεται ως η μέση τιμή του αθροίσματος των διαφορών μεταξύ του candidate και κάθε ενός reference block υψωμένο στο τετράγωνο. Αυτό που επιδιώκουμε με τη χρήση του MSE είναι να εντοπίσουμε το reference block με τη μικρότερη δυνατή διαφορά, δηλαδή, την ελάχιστη τιμή σφάλματος. Αφού εντοπίσουμε ποιο είναι αυτό το block, κατασκευάζουμε το διάνυσμα κίνησης $\mathbf{d} = (d_1, d_2)$,

με $d_1$ (αρχή) το κέντρο του candidate block και $d_2$ (πέρας) το κέντρο του reference block με το μικρότερο MSE.

Μια συνηθισμένη περίπτωση είναι να εκτελούμε την αναζήτηση με $N = 8$, επομένως τα block θα έχουν διαστάσεις 8x8 pixel. Σε αυτήν την περίπτωση ο υπολογισμός του συνολικού αθροίσματος θα περιλαμβάνει 64 προσθέσεις ($N^2$) αλλά και 64 πολλαπλασιασμούς, λόγω του τετραγώνου. Ο πολλαπλασιασμός ως υπολογιστική πράξη είναι ιδιαίτερα "προβληματικός" αφού όταν γίνεται σε επίπεδο software καταναλώνει αρκετό χρόνο ενώ σε επίπεδο hardware είναι ένα σχετικά κοστοβόρο κύκλωμα. Προκειμένου να έχουμε ένα κριτήριο ταύτισης που δεν περιλαμβάνει κάποιο πολλαπλασιασμό η ίδια φιλοσοφία μπορεί να εφαρμοστεί αντικαθιστώντας τον τετραγωνισμό με την **απόλυτη τιμή της διαφοράς**.

### 2.5.2 Απολυτη Τιμη Διαφορας

Η ελάχιστη απόλυτη τιμή της διαφοράς δύο block είναι το άλλο κριτήριο που μπορούμε να χρησιμοποιήσουμε προκειμένου να συγκρίνουμε το candidate block με κάθε ένα από τα reference blocks και να αποφανθούμε για τη βέλτιστη αντιστοίχιση. Το πλεονέκτημα αυτού του κριτηρίου είναι η "απουσία" του πολλαπλασιασμού, πράγμα που βελτιώνει σημαντικά την απόδοση της οποιασδήποτε υλοποίησης ενός αλγόριθμου εκτίμησης κίνησης.

$$\mathrm{MAD}(i,j) = \frac{1}{N^2} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \mid s(n_1, n_2, k) - s(n_1 + i, n_2 + j, k - l) \mid \qquad (2.2)$$

Η χρήση του MAD ως κριτήριο ταύτισης είναι ιδιαίτερα διαδεδομένη στους περισσότερους μηχανισμούς εκτίμησης κίνησης λόγω της σχετικά απλής υλοποίησης η οποία περιλαμβάνει μόλις 64 προσθέσεις και αφαιρέσεις.

### 2.5.3 Αριθμος Αντιστοιχισμενων Pixel

Στην περίπτωση αυτού του κριτηρίου η προσέγγιση που ακολουθείται είναι διαφορετική. Η βασική μετρική που χρησιμοποιείται είναι ο αριθμός των pixel μεταξύ δύο block που θεωρούνται αντιστοιχισμένα. Πρακτικά, ορίζουμε ένα **όριο τιμής (value threshold)** $\theta$, κάτω από το οποίο θεωρούμε ότι δύο pixel έχουν κοντινή συσχέτιση και άρα μπορούν να προσμετρηθούν ως αντοιστοιχισμένα (να αυξήσουμε δηλαδή έναν μετρητή κατά 1). Σε αντίθετη περίπτωση το τρέχον pixel που ελέγχουμε δεν

προσμετράται και άρα δεν αυξάνουμε την τιμή του μετρητή. Όπως είναι εύκολα αντιληπτό, στο συγκεκριμένο κριτήριο μας ενδιαφέρει η μέγιστη τιμή του μετρητή η οποία υποδηλώνει και τη μέγιστη ταύτιση. Η μαθηματική έκφραση του κριτηρίου συνοψίζεται στις ακόλουθες σχέσεις:

$$\text{count}(n_1, n_2) = \begin{cases} 1, & \text{αν } \mid s(n_1, n_2, k) - s(n_1 + i, n_2 + j, k - l) \mid \leq \theta \\ 0, & \text{οπουδήποτε αλλού} \end{cases} \tag{2.3}$$

$$\text{MPC}(i, j) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \text{count}(n_1, n_2) \tag{2.4}$$

Η σχέση (2.3) είναι μία δυαδική συνάρτηση η οποία παίρνει την τιμή 1 αν η διαφορά βρίσκεται εντός του ορίου τιμής που θέσαμε, ενώ σε αντίθετη περίπτωση παίρνει την τιμή 0. Η σχέση (2.4) αποτελεί το συνολικό άθροισμα των τιμών της (2.3) για κάθε pixel του block. Είναι προφανές ότι η μέγιστη τιμή καθορίζεται από τη μεταβλητή N και συνεπώς τις διαστάσεις του block. Στην περίπτωση που εξετάσαμε προηγουμένως, αν δηλαδή έχουμε block των 8x8 pixel, η μέγιστη τιμή του μετρητή θα είναι 64 ενώ η ελάχιστη 0.

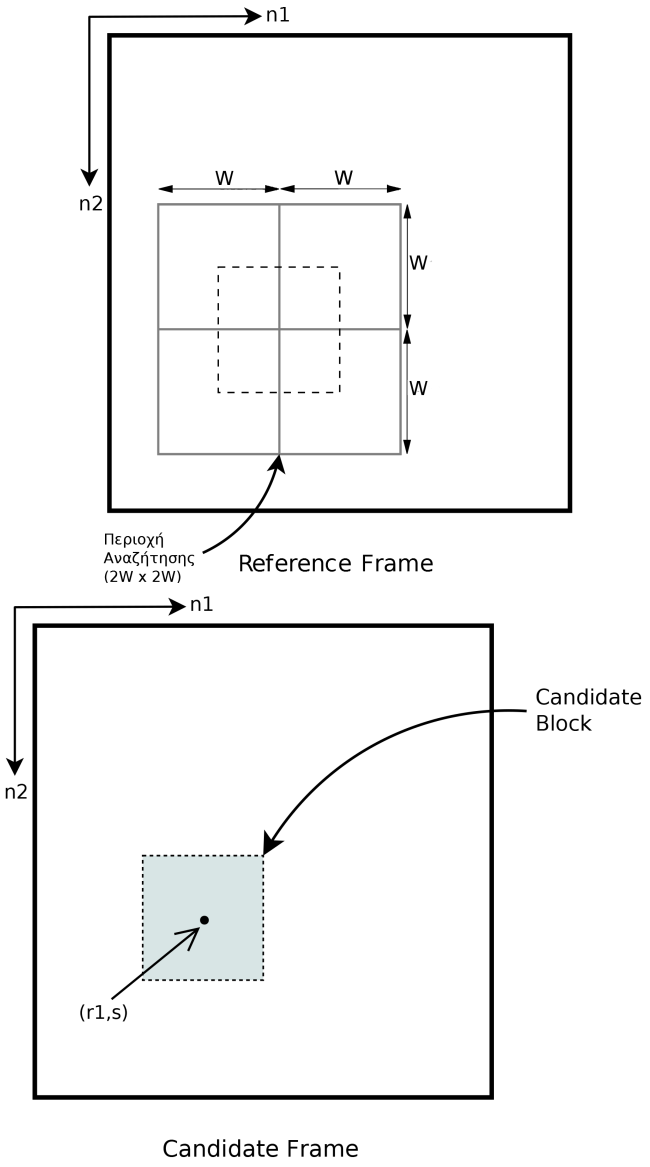
## 2.6 Αλγοριθμοι Εκτιμησης Κινησης

Η εκτίμηση κίνησης είναι ένα πρόβλημα που χαρακτηρίζεται από μεγάλη πολυπλοκότητα, ενώ η λύση του μπορεί να προσεγγιστεί με πολλές και διαφορετικές τεχνικές. Είναι επόμενο λοιπόν να έχει αναπτυχθεί ένα πλήθος τεχνικών και στρατηγικών εκτίμησης κίνησης βασισμένων σε διαφορετικά χαρακτηριστικά του προβλήματος.

### 2.6.1 Εξαντλητικη Αναζητηση (Full-Search Block Motion Estimation η FSBME)

Ο αλγόριθμος ο οποίος θα μας απασχολήσει σε όλη την υπόλοιπη έκταση της εργασίας, είναι αυτός για τον οποίο ήδη έχουμε πει κάποια πράγματα, καθώς είναι ο πιο απλός και αποτελεσματικός αλγόριθμος που μπορεί να χρησιμοποιηθεί. Το βασικό του μειονέκτημα είναι ο εξαιρετικά μεγάλος χρόνος εκτέλεσης, αφού εκτελεί υπολογισμούς πάνω σε κάθε pixel του candidate block και της περιοχής αναζήτησης.

Θεωρούμε ένα block $N \times N$ pixel στο candidate frame (το candidate block) στη θέση $(r_1, s)$ όπως φαίνεται στο σχήμα 2.9. Θεωρούμε στη συνέχεια μια περιοχή ανα-




**Σχήμα 2.9**: Δυο διαδοχικά frames

ζήτησης στο reference frame (search window) γύρω από τη θέση του αντίστοιχου block, με διαστάσεις έυρους $\pm$w pixel και προς τις δύο κατευθύνσεις $n_1$ και $n_2$. Για κάθε μία από τις $(2w + 1)^2$ θέσεις αναζήτησης το candidate block συγκρίνεται με ένα block ίδιων διαστάσεων N $\times$ N pixel, σύμφωνα με τα κριτήρια που συζητήσαμε

**Πίνακας 2.2**: Υπολογιστική πολυπλοκότητα του αλγορίθμου FSBM

| **Κριτήριο ταύτισης** | **Απαιτούμενοι υπολογισμοί** | | |
|---|---|---|---|
| | Προσθέσεις/ Αφαιρέσεις | Πολλ/σμοί | Συγκρίσεις |
| MSE | $(2N^2-1)(2w+1)^2$ | $N^2(2w+1)^2$ | $(2w+1)^2$ |
| MAD | $(2N^2-1)(2w+1)^2$ | - | $(2w+1)^2$ |
| MPC | $(2N^2-1)(2w+1)^2$ | - | $N^2(2w+1)^2$ |

στην προηγούμενη ενότητα και το block με τη μεγαλύτερη ταύτιση, μαζί με το διάνυσμα κίνησης, καθορίζονται **μόνο** αφού ολοκληρωθεί η διαδικασία της εξαντλητικής αναζήτησης και ελεγχθούν όλες οι πιθανές θέσεις.

Το ιδανικό μέγεθος για μια περιοχή αναζήτησης εξαρτάται από πολλές παραμέτρους:

- Την ανάλυση της κάθε εικόνας (ένα μεγάλο παράθυρο είναι προφανώς κατάλληλο για μια πολύ μεγάλη ανάλυση).
- Τον τύπο της σκηνής (σκηνές με γρήγορη κίνηση συνήθως οφελούνται με ένα μεγάλο παράθυρο αναζήτησης σε αντίθεση με τις σκηνές που έχουν μικρές κινήσεις).
- Τους υπολογιστικούς πόρους που είναι διαθέσιμοι, καθώς ένα μεγάλο παράθυρο αναζήτησης απαιτεί περισσότερες συγκρίσεις και άρα περισσότερους υπολογισμούς.

Ο αλγόριθμος FSBM είναι η βέλτιστη δυνατή λύση στο πρόβλημα της εκτίμησης κίνησης, κυρίως διότι, αν οριστεί σωστά η περιοχή αναζήτησης, μας εγγυάται ότι θα εντοπίσει εκείνο το block με τη μεγαλύτερη ταύτιση. Ωστόσο όπως έχουμε ήδη αναφέρει πολλές φορές είναι ιδιαίτερα δαπανηρός από πλευράς κατανάλωσης υπολογιστικής ισχύος. Για κάθε θέση αναζήτησης απαιτούνται $O(N^2)$ υπολογισμοί (προσθέσεις, αφαιρέσεις, κλπ) και εφόσον υπάρχουν $(2w+1)^2$ θέσεις ο αριθμός των υπολογισμών σε σχέση με το κριτήριο ταύτισης που χρησιμοποιείται απεικονίζεται στον πίνακα (2.2).

Ο μεγάλος αριθμός υπολογισμών που απαιτούνται κατά την εξαντλητική αναζήτηση οδήγησε στην ανάπτυξη νέων τεχνικών που προσεγγίζουν το πρόβλημα της

εκτίμησης κίνησης πολύ γρηγορότερα. Συνήθως όμως, επειδή τα αποτελέσματα αυτών των τεχνικών δεν είναι τα βέλτιστα, ανεχόμαστε κάποιο βαθμό σφάλματος. Στις επόμενες ενότητες θα εξετάσουμε επιγραμματικά μερικές από αυτές τις τεχνικές αναζήτησης.

### 2.6.2 Αναζητηση Τριων Βηματων (Three Step Search)

Στον αλγόριθμο αναζήτησης τριών βημάτων[19], η πρώτη επανάληψη αξιολογεί 9 υποψήφια σημεία συμπεριλαμβανομένου του κέντρου. Το αρχικό βήμα είναι ίσο ή λίγο μεγαλύτερο απο το μισό της περιοχής αναζήτησης. Με το πέρας της επαναλήψεως και τον υπολογισμό του σφάλματος για τα εννιά υποψήφια σημεία, το κέντρο αναζήτησης μετατοπίζεται στο σημείο με το μικρότερο σφάλμα και μειώνεται το βήμα στο μισό. Η ίδια διαδικασία επαναλαμβάνεται μέχρι το βήμα να γίνει ίσο με ένα εικονοστοιχείο. Στην περίπτωση αυτή, το σημείο με το μικρότερο σφάλμα θα επιλεγεί ως το βέλτιστο αποτέλεσμα και το διάνυσμα κίνησης που αντιστοιχεί σε αυτό επιλέγεται για το τρέχον μπλοκ. Ο αριθμός των σημείων που αξιολογούνται κατά τη διάρκεια της αναζήτησης είναι πολύ μικρότερος από αυτών της εξαντλητικής μεθόδου και καθορίζεται από το μέγεθος του βήματος που τίθεται στην πρώτη επανάληψη. Βασικά πλεονεκτήματα του αλγορίθμου είναι τα εξής:

- Μικρή πολυπλοκότητα
- Χαμηλός χρόνο αναζήτησης
- Αρκετά καλά αποτελέσματα και συνέπεια

Βασικά μειονεκτήματα του αλγορίθμου είναι τα εξής:

- Ο παράγοντας της πολυπλοκότητας μεγαλώνει ανάλογα με την περιοχή αναζήτησης
- Ακατάλληλος για μικρές κινήσεις
- Χρησιμοποιείται κυρίως σε εφαρμογές που απαιτούν χαμηλό ρυθμό bit όπως τηλεδιασκέψεις και τηλέφωνα με δυνατότητα βιντεοκλήσης.

### 2.6.3 Diamond Search

Η λογική του συγκεκριμένου αλγορίθμου[31][32] είναι αρκετά απλή και βασίζεται στην αναζήτηση με βάση την επιλογή σημείων σε μια διάταξη διαμαντιού. Πιο συγκεκριμένα, ο αλγόριθμος ξεκινάει δημιουργώντας μια διάταξη διαμαντιού επιλέγοντας 9 σημεία. Στο επόμενο βήμα, αφού βρεθεί και επιλεγεί το σημείο με το μικρότερο σφάλμα, επαναλαμβάνεται η ίδια διαδικασία με κέντρο το επιλεγμένο αυτό σημείο. Όταν το σημείο που επιλεγεί είναι το κέντρο της διάταξης, τότε ο αλγόριθμος επιλέγει 5 γειτονικά σημεία δημιουργώντας και πάλι μια διάταξη διαμαντιού, μικρότερου φυσικά από το προηγούμενο, και βρίσκει και πάλι το σημείο με το μικρότερο σφάλμα ανάμεσα σε αυτά. Αυτό, λοιπόν, το σημείο είναι που ψάχνουμε και αποτελεί το βέλτιστο ταίρι. Βασικά πλεονεκτήματα του αλγορίθμου είναι τα εξής:

- Καλύτερη απόδοση και ταχύτητα από τον αλγόριθμο αναζήτησης των τριών βημάτων
- Μεγάλη μέγιστη σήματος-προς-θόρυβο αναλογία

Βασικά μειονεκτήματα του αλγορίθμου είναι τα εξής:

- Δεν υπάρχει περιορισμός στα βήματα αναζήτησης
- Δεν υπάρχει αποτελεσματική υλοποίηση του σε hardware

Στο σχήμα φαίνεται ένα παράδειγμα αναζήτησης του αλγορίθμου οπού με κύκλο, τετράγωνο, ρόμβο και τρίγωνο, συμβολίζονται σημεία που ανήκουν σε διάταξη μεγάλου διαμαντιού αντίστοιχα για κάθε σχήμα. Με τον κύκλο χρώματος γκρι, συμβολίζεται η διάταξη μικρού διαμαντιού. Βλέπουμε ότι ο αλγόριθμος χρειάζεται πέντε βήματα για να καταλήξει σε ένα διάνυσμα κίνησης με συντεταγμένες (-4, -2). Τέσσερα από τα βήματα είναι με διάταξη αναζήτησης μεγάλου διαμαντιού και ένα από αυτά, το τελευταίο, με διάταξη αναζήτησης μικρού διαμαντιού.

### 2.6.4 Δισδιαστατη Λογαριθμικη Αναζητηση (2-D Logarithmic Search)

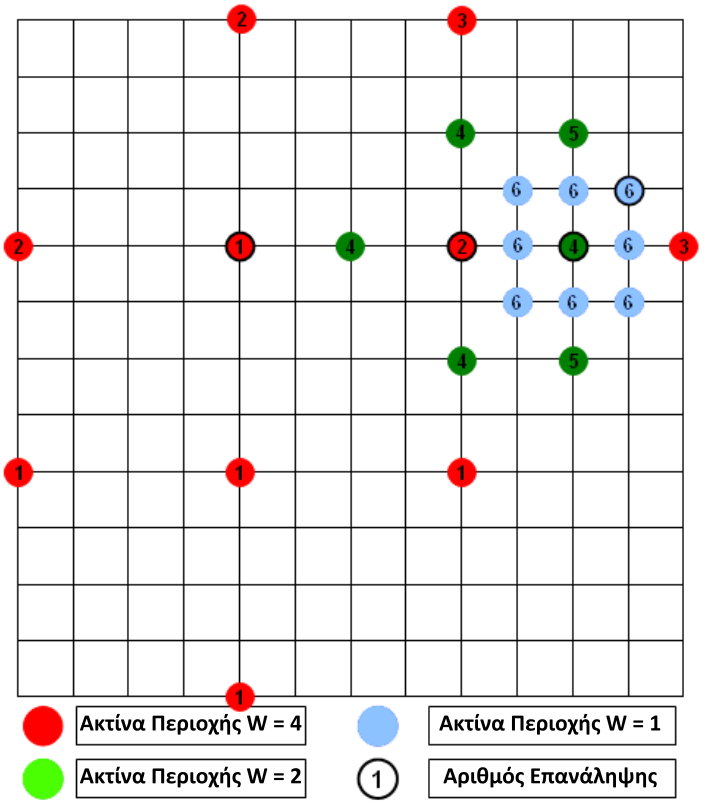
Σ' αυτήν την τεχνική που αρχικά προτάθηκε από τους Jain και Jain θεωρούμε μια αρχική περιοχή αναζήτησης με μια υποθετική βέλτιστη θέση στο κέντρο της. Η

περιοχή αναζήτησης εκτείνεται προς τις τέσσερις κατευθύνσεις κατά **p** pixel, επομένως έχει εμβαδόν $4p^2$ pixel. Αυτή η αρχική βέλτιστη θέση μπορεί να θεωρηθεί και ως η θέση του candidate block από τη σκοπιά του candidate frame. Αντί για την αναζήτηση σε όλες τις πιθανές θέσεις της περιοχής αναζήτησης, η αναζήτηση πραγματοποιείται μόνο σε πέντε θέσεις, την αρχική και τέσσερις γειτονικές της σε απόσταση **p/2** από αυτήν (μία σε κάθε πλευρά του τετραγώνου), κατα τέτοιο τρόπο ώστε τα κέντρα τους να σχηματίζουν μια δομή με σχήμα σταυρού. Το p/2 ονομάζεται και **βήμα** και στο εξής μπορούμε να το συμβολίζουμε με **s**.

Η εκτέλεση του αλγόριθμου περιλαμβάνει τα εξής βήματα:

- Στην πρώτη επανάληψη του αλγορίθμου εξετάζουμε την πρώτη "πεντάδα" θέσεων, την αρχική υποθετική βέλτιστη, και τις τέσσερις γειτονικές της.
- Αν κάποια από τις τέσσερις γειτονικές θέσεις αποτελεί βέλτιστη λύση τότε θέτουμε αυτήν ως "κεντρική" θέση και ορίζουμε ξανά τέσσερις γειτονικές της. Μοιραία, τουλάχιστον μία από τις νέες γειτονικές θέσεις θα έχει εξεταστεί και στο παρελθόν, επομένως πλέον έχουμε λιγότερες θέσεις προς εξέταση. Εκτελούμε ξανά τον έλεγχο με τις νέες θέσεις.
- Αν σε κάποια επανάληψη του αλγορίθμου εντοπίσουμε ως βέλτιστη θέση αυτήν που ορίσαμε ως κεντρική και όχι κάποια γειτονική της, τότε μειώνουμε τον αριθμό s στο μισό (κατ' επέκταση υποτετραπλασιάζουμε το μέγεθος της θέσης) και εκτελούμε την επόμενη επανάληψη του αλγόριθμου. Αυτή τη φορά θέτουμε ξανά την ίδια θέση ως κεντρική όμως ορίζουμε νέες γειτονικές θέσεις σε απόσταση $s' = s/2$ από αυτήν.
- Ο αλγόριθμος τερματίζει όταν το s γίνει ίσο με 1, οπότε πλέον ελέγχουμε και τις 9 γειτονικές θέσεις που περιβάλλουν την τελευταία βέλτιστη που βρήκαμε. Από αυτή την τελευταία αναζήτηση προκύπτει η τελική βέλτιστη θέση και τελικά το διάνυσμα κίνησης.

Στο σχήμα 2.10 φαίνεται η εκτέλεση του παραπάνω αλγορίθμου. Οι αριθμοί μέσα στους κύκλους συμβολίζουν την επανάληψη κατά την οποία ορίστηκαν αυτές οι θέσεις. Οι κυκλωμένες θέσεις υποδεικνύουν βέλτιστα. Αξίζει να σημειώσουμε ότι για τις θέσεις που δεν υπάρχει αρίθμηση σύμφωνη με την τρέχουσα επανάληψη του αλγόριθμου, ισχύει ότι έχουν ήδη υπολογιστεί σε προγούμενη επανάληψη.




**Σχήμα 2.10**: Τα βήματα εκτέλεσης του Αλγόριθμου Λογαριθμικής Αναζήτησης

# 3
# Σχεδιασμός του Υλικού

## 3.1 Εισαγωγη

Στο προηγούμενο κεφάλαιο αναφερθήκαμε στη διαδικασία της εκτίμησης κίνησης, το ρόλο που αυτή διαδραματίζει στη συμπίεση βίντεο και τους συνήθεις αλγορίθμους εκτίμησης κίνησης που χρησιμοποιούνται σε διάφορες εφαρμογές. Στην εργασία αυτή αναπτύξαμε ένα ενσωματωμένο σύστημα το οποίο υλοποιεί τον αλγόριθμο εξαντλητικής αναζήτησης (FSBM) μέσω ενός ειδικά σχεδιασμένου επιταχυντή/συνεπεξεργαστή. Στο κεφάλαιο αυτό λοιπόν θα υπεισέλθουμε σε λεπτομέρειες που αφορούν τόσο την αρχιτεκτονική του συστήματος όσο και τις μεθόδους και τεχνικές που ακολουθήθηκαν κατά το σχεδιασμό του.

## 3.2 Θεωρητικη περιγραφη

Οι περισσότεροι αλγόριθμοι εκτίμησης κίνησης που έχουν προταθεί ως τώρα παρόλο που αποτελούν βελτιώσεις του αλγορίθμου εξαντλητικής αναζήτησης ως προς την ταχύτητα, ακολουθούν μια μη-κανονική ροή δεδομένων προς επεξεργασία. Αυτό έχει ως αποτέλεσμα την εξαιρετικά δύσκολη - αν όχι αδύνατη - υλοποίηση τους σε επίπεδο υλικού. Αντίθετα, ο FSBM παρόλη τη χαμηλή του ταχύτητα, επεξεργάζεται μια σταθερή και κανονική ροή δεδομένων καθ' όλη τη διάρκεια της εκτέλεσης του και αυτό αποτελεί πλεονέκτημα για την υλοποίησή του σε VLSI. Αυτός είναι ένας

**Πίνακας 3.1**: Framerate και μέγεθος καρέ για μερικά διάσημα πρότυπα

| Format | Pixels / line | Lines / Frame | FPS |
|---|---|---|---|
| UHD 8K | 7680 | 4320 | 30 |
| UHD 4K | 3840 | 2160 | 30 |
| HDTV | 1920 | 1080 | 30 |
| SDTV | 720 | 486 | 30 |
| Τηλεδιάσκεψη (SIF) | 352 | 240 | 30 |

βασικός λόγος για τον οποίο ο FSBM εξακολουθεί να χρησιμοποιείται σε πολλές εφαρμογές.

Αναφερθήκαμε πολλές φορές στο μεγάλο πλήθος υπολογισμών που απαιτούνται κατά την εκτέλεση του FSBM, πράγμα που εξαρτάται σε μεγάλο βαθμό από το μέγεθος του frame. Στον πίνακα 3.1 αναφέρονται επιγραμματικά οι διαστάσεις των frame και το framerate για μερικά διαδεδομένα πρότυπα.

Όπως είναι εμφανές, για πρότυπα υψηλής και υπερυψηλής ευκρίνειας το μέγεθος του καρέ μεγαλώνει εκθετικά και μαζί ο αριθμός των εν δυνάμει περιοχών αναζήτησης, με τελικό αποτέλεσμα την αύξηση του υπολογιστικού φόρτου. Εκτός από αυτό όμως υπάρχει ένα ακόμη πρόβλημα, ίσως όχι άμεσα εμφανές. Ο ίδιος ο αριθμός των pixel που πρέπει να μετακινηθούν από την εξωτερική μνήμη είναι από κάθε άποψη τεράστιος, ειδικά σε πρότυπα όπως τα UHD 4/8K όπου για κάθε frame απαιτείται μεταφορά έως και 33,2 εκατομμυρίων pixel, δηλαδή περίπου 133 MByte δεδομένων!

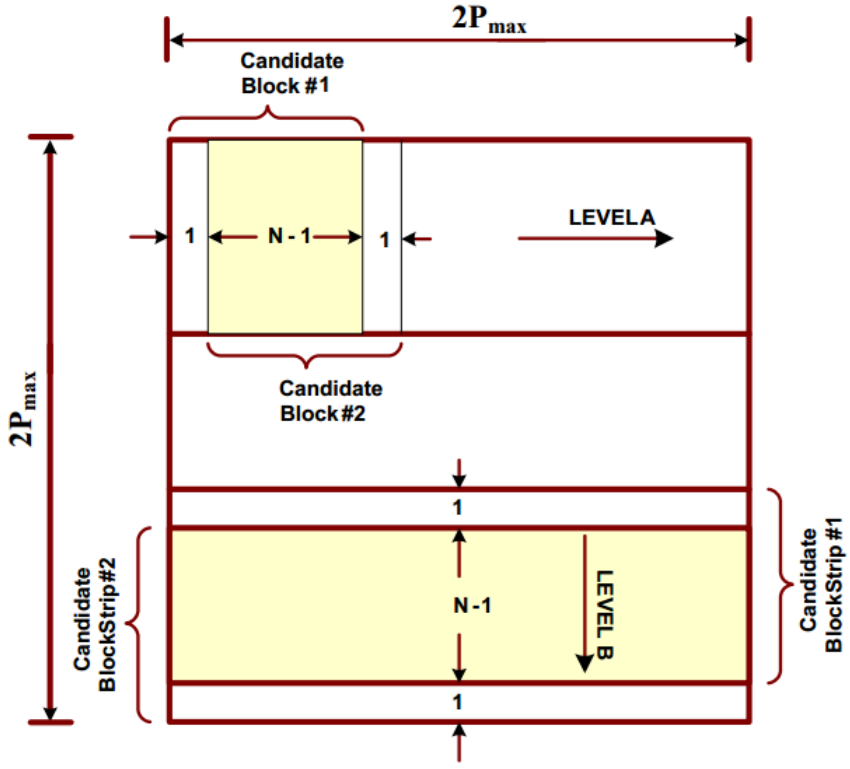
### 3.2.1 Επαναχρησιμοποιηση Δεδομενων

Το πρόβλημα του μεγάλου όγκου δεδομένων μπορεί να επιλυθεί χρησιμοποιώντας έξυπνες τεχνικές επαναχρησιμοποίησης δεδομένων. Με τον τρόπο αυτό μειώνουμε τις προσβάσεις στην εξωτερική μνήμη και μεταφέρουμε χρήσιμα δεδομένα μόνο όταν αυτό είναι απαραίτητο. Στο σχεδιασμό του επιταχυντή εφαρμόστηκε η επαναχρησιμποποίηση σε δύο επίπεδα, Α και Β.

Το επίπεδο Α αφορά στην επαναχρησιμοποίηση δεδομένων μεταξύ διαδοχικών θέσεων αναζήτησης σε μια κοινή ''λωρίδα'' της περιοχής αναζήτησης. Αν θεωρήσουμε ένα candidate block διαστάσεων N $\times$ N pixel και μια περιοχή αναζήτησης $2P_{max} \times 2P_{max}$ τότε σε μία ''λωρίδα'' της περιοχής αναζήτησης οι διαδοχικές θέσεις αναζήτησης θα αλληλοεπικαλύπτονται κατά N-1 $\times$ N pixel. Πρακτικά αυτό σημαίνει ότι κάθε επόμενο candidate block θα διαφέρει μόλις κατα μία στήλη από pixel - σε

σχέση με το προηγούμενο - μεγέθους 1 × N pixel. Αποτέλεσμα αυτής της παρατήρησης είναι ότι για να υπολογίσουμε κάθε νέα θέση απαιτείται η μεταφορά μίας και μόνο στήλης από την εξωτερική μνήμη, αφού το υπόλοιπο του candidate block μπορεί να επαναχρησιμοποιηθεί.

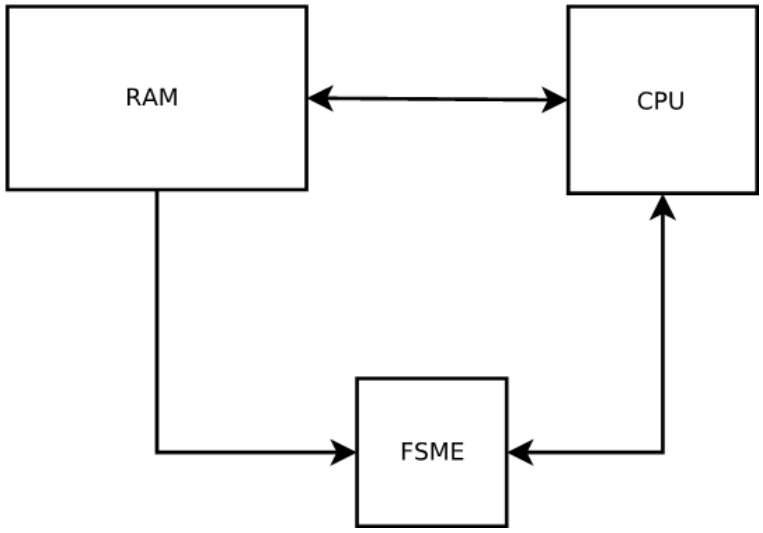
Το επίπεδο Β αφορά στην επαναχρησιμοποίηση δεδομένων μεταξύ διαδοχικών "λωρίδων" της περιοχής αναζήτησης. Η λογική είναι ή ίδια με το επίπεδο Α αφού μεταξύ διαδοχικών λωρίδων αναζήτησης υπάρχει επικάλυψη κατα N × N - 1 pixel, οπότε κατα την επεξεργασία μιας "λωρίδας" το μεγαλύτερο μέρος της προηγούμενής της μπορεί να ξαναχρησιμοποιηθεί. Στο σχήμα 3.1 μπορούμε να δούμε τα δύο επίπεδα επαναχρησιμοποίησης.




**Σχήμα 3.1**: Τα επίπεδα επαναχρησιμοποίησης Α και Β

## 3.3 Σχεδιασμος του Συστηματος

Το σύστημα του επιταχυντή αναπτύχθηκε σε δύο επίπεδα. Το πρώτο επίπεδο αφορά την εσωτερική λειτουργία του επιταχυντή, ενώ το δεύτερο τη διασύνδεση και την αλληλεπίδραση του με τα υπόλοιπα components. Στις επόμενες παραγράφους θα αναπτύξουμε και θα συζητήσουμε τις αρχιτεκτονικές που διέπουν τα δύο αυτά επίπεδα, καθώς και τις ιδιαιτερότητες και τα πλεονεκτήματα/μειονεκτήματά τους. Αρχικά, και προκειμένου να αποκτήσουμε μια γενική ιδέα, θα κάνουμε μια επιγραμματική αναφορά στον τρόπο με τον οποίο ο επιταχυντής αλληλεπιδρά με τα υπόλοιπα στοιχεία του συστήματος. Τα βασικά στοιχεία τα οποία πλαισιώνουν λοιπόν το περιφερειακό μας είναι ο επεξεργαστής και η μνήμη RAM. Ο επεξεργαστής παρέχει τα κατάλληλα σήματα στη RAM ώστε αυτή με τη σειρά της να τροφοδοτήσει με δεδομένα τον επιταχυντή, ενώ παράλληλα δέχεται τα αποτελέσματα της επεξεργασίας των δεδομένων και δίνει τον έλεγχο στον επιταχυντή όταν αυτό είναι απαραίτητο. Παρακάτω φαίνεται ένα εποπτικό σχήμα αυτής της δομής (σχήμα 3.2).




**Σχήμα 3.2**: Εποπτικό σχήμα της αρχιτεκτονικής του συστήματος

## 3.4 Εσωτερικη Αρχιτεκτονικη του Επιταχυντη

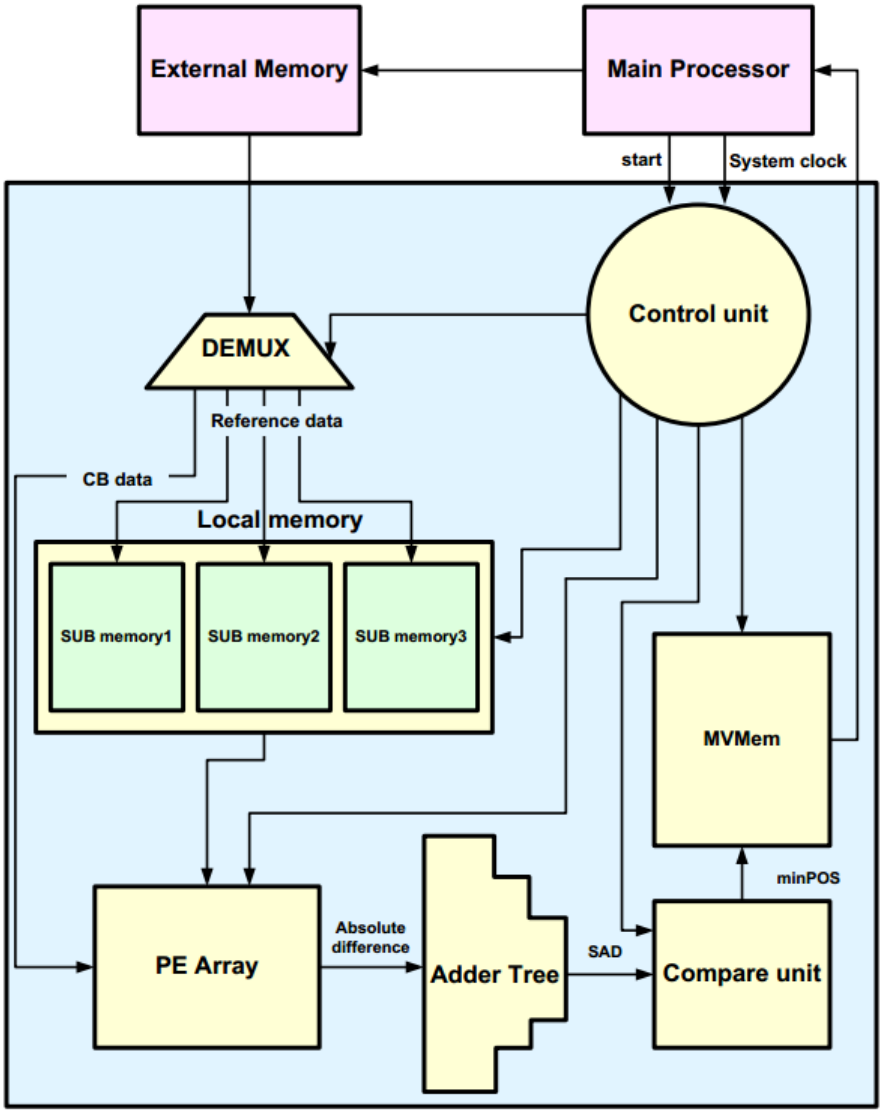
Η αρχιτεκτονική που υλοποιήθηκε για το κύκλωμα του επιταχυντή φαίνεται στο σχήμα 3.3 και χρησιμοποιείται κυρίως στο πρότυπο H.264/AVC. Η περιοχή αναζή-

τησης που ανακτάται από την εξωτερική μνήμη έχει μέγεθος $2P_{max} \times (2P_{max} + N - 1)$ και το current block έχει διαστάσεις $N \times N$. Επιλέξαμε τόσο το N όσο και το $P_{max}$ να έχουν την τιμή 16. Η διαδικασία της εκτίμησης κίνησης αρχίζει μόλις ο **αποπολυπλέκτης (demux)** λάβει τόσο τα pixel του current block (CB) όσο και τα pixel της περιοχής αναζήτησης από την εξωτερική μνήμη. Ο αποπολυπλέκτης δρομολογεί τα δεδομένα είτε προς την **τοπική μνήμη (local memory)** είτε απευθείας στη **μονάδα επεξεργασίας (PE Array)**. Η τοπική μνήμη αποτελείται από τρεις "υπομνήμες". Στη μονάδα επεξεργασίας καταλήγουν τόσο τα δεδομένα από την τοπική μνήμη (candidate blocks) όσο και το υπό εξέταση block (current block), όπου υπολογίζονται οι **απόλυτες διαφορές (absolute differences)**. Στη συνέχεια οι διαφορές αυτές θα προωθηθούν σε έναν αθροιστή δέντρου για να υπολογιστεί το άθροισμά τους και τελικά το κριτήριο **SAD (Sum of Absolute Differences)**. Το αποτέλεσμα της άθροισης τροφοδοτείται κατόπιν σε μια μονάδα σύγκρισης προκειμένου να εντοπιστεί η θέση εκείνη με το μικρότερο SAD. Μετά τη σύγκριση, η θέση του ελάχιστου τελικού SAD αποθηκεύεται με μια μνήμη διανυσμάτων κίνησης (Motion Vactor Memory), και τελικά το σύνολο των διανυσμάτων κίνησης αποστέλλεται στον επεξεργαστή στο τέλος της διαδικασίας. Η διαχείριση των σημάτων ελέγχου γίνεται από μια **μονάδα ελέγχου (control unit)** μέσα στον επιταχυντή.

Αξίζει να σημειώσουμε ότι πρόκειται για μία scalable αρχιτεκτονική η οποία μπορεί να προσαρμοστεί σε οσοδήποτε μεγάλα η μικρά frames και περιοχές αναζήτησης αρκεί να τροποποιηθεί κατάλληλα το εύρος των διαύλων, το μέγεθος των μνημών (αν απαιτείται από τις προδιαγραφές του format) και το μέγεθος της μονάδας επεξεργασίας (ώστε να έχει τη δυνατότητα παράλληλης επεξεργασίας περισσότερων δεδομένων). Είναι επομένως σχετικά εύκολο να δούμε την πρακτική της εφαρμογή σε πρότυπα όπως το H.265/HEVC, αρκεί να επεκτείνουμε τη μονάδα επεξεργασίας σε $32 \times 32$ προκειμένου να εναρμονίζεται με τις απαιτήσεις του προτύπου. Στις επόμενες παραγράφους θα αναφερθούμε εκτενώς στα επιμέρους τμήματα του επιταχυντή, την ακριβή λειτουργία τους, καθώς και τη διαδικασία σχεδιασμού τους.

### 3.4.1 Μονάδα Υπολογισμού Αθροίσματος Απολύτων Διαφορών (SAD Unit)

Η Μονάδα Υπολογισμού Αθροίσματος Απολύτων Διαφορών (στο εξής θα αναφερόμαστε σ' αυτήν ως SAD Unit για συντομία) αποτελεί την καρδιά του επιταχυντή. Πρόκειται επι της ουσίας για το μηχανισμό που εφαρμόζει το κριτήριο ταύτισης SAD και αναλαμβάνει το βάρος των απαραίτητων υπολογισμών. Το SAD Unit απο-



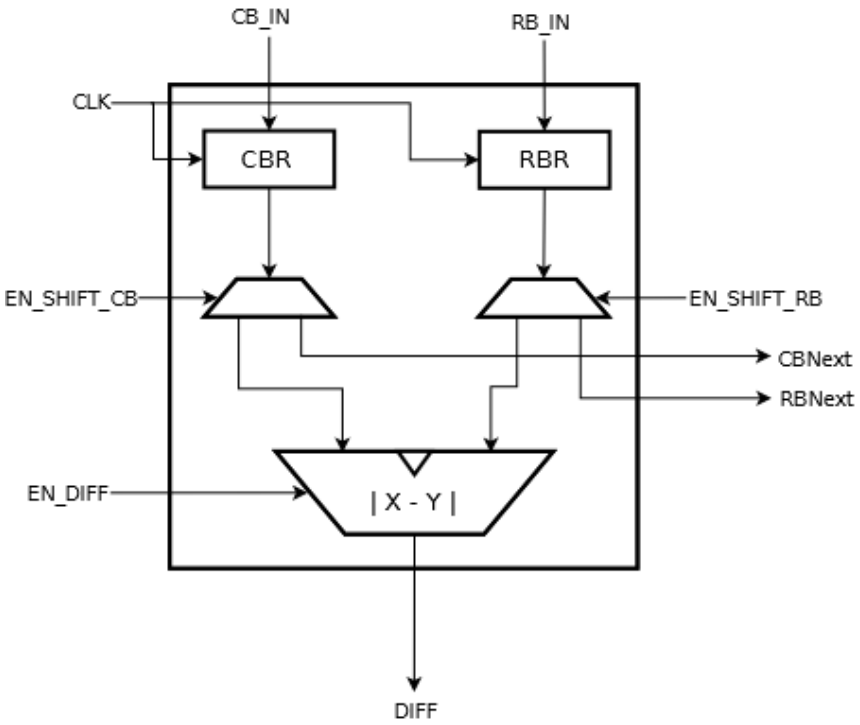

**Σχήμα 3.3**: Εποπτικό σχήμα της αρχιτεκτονικής του επιταχυντή

τελείται από δύο μικρότερα επιμέρους τμήματα, τη **Συστοιχία Επεξεργαστικών Στοιχείων (Processing Element Array - PE Array)** και το **Δέντρο Αθροιστών (Adder Tree)**. Κάθε ένα από αυτά τα τμήματα εξυπηρετεί διαφορετική διεργασία κατα την εφαρμογή του κριτηρίου SAD. Το PE Array είναι υπεύθυνο για τον υπολογισμό των διαφορών ενώ το Δέντρο Αθροιστών προσθέτει μεταξύ τους τις διαφορές που προκύπτουν στην έξοδο του PE Array.

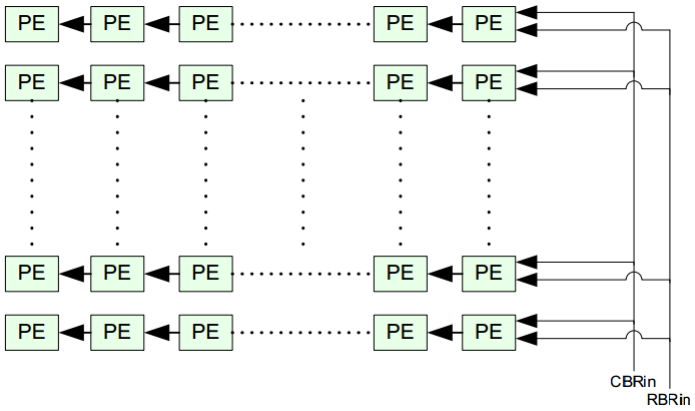
Από τη στιγμή που αποφασίσαμε να χρησιμοποιήσουμε block διαστάσεων $16 \times 16$ pixel, εξηγείται εύκολα η δομή του PE Array, το οποίο αποτελείται από 16 γραμμές επεξεργασίας, κάθε μία από τις οποίες περιλαμβάνει 16 επεξεργαστικά στοιχεία (Processing Elements - PE), συνολικά δηλαδή, 256 PE. Κάθε στοιχείο επεξεργασίας λειτουργεί βρίσκοντας την απόλυτη διαφορά μεταξύ των τιμών δύο pixel, το ένα από τα οποία προέρχεται από το candidate block και το άλλο από το reference block. Ένα PE αποτελείται από δύο καταχωρητές (registers) των 8-bit, δύο πολυπλέκτες 2-σε-1 και μια μονάδα υπολογισμού απόλυτης διαφοράς. Οι δύο καταχωρητές αποθηκεύουν τις τιμές δύο pixel, ενός προερχόμενου από το candidate block και ενός από το reference block. Οι τιμές των pixel μπορεί να είναι από 0 μέχρι 255, που αντιστοιχούν στις διαβαθμίσεις του γκρι (greyscale), επομένως 8 bit είναι αρκετά για την αναπαράσταση των τιμών. Τα δεδομένα του κάθε καταχωρητή σε κάθε κύκλο ρολογιού δρομολογούνται στον αντίστοιχο πολυπλέκτη. Αν βρισκόμαστε στη φάση της φόρτωσης των δεδομένων οι πολυπλέκτες θα στείλουν τα δεδομένα στο επόμενο PE, ενώ αν βρισκόμαστε στη φάση της επεξεργασίας τα δεδομένα θα δρομολογηθούν στη μονάδα αφαίρεσης όπου και θα υπολογιστεί η απόλυτη διαφορά τους. Από κάθε PE λαμβάνουμε ως τελική έξοδο αυτήν ακριβώς τη διαφορά ως τιμή των 8-bit. Στο σημείο αυτό θα πρέπει να σημειωθεί ότι τα PE δουλεύουν με μη-προσημασμένες τιμές (unsigned), επομένως το αποτέλεσμα θα είναι πάντα θετικό. Το σχήμα 3.4 περιγράφει την ακριβή δομή ενός processing element.

Η ροή των δεδομένων στο PE array γίνεται με παράλληλο τρόπο. Σε κάθε κύκλο ρολογιού φορτώνονται 16 pixel, από 1 σε κάθε γραμμή επεξεργασίας, πράγμα που σημαίνει ότι η φόρτωση γίνεται στήλη-προς-στήλη. Τα PE κάθε σειράς είναι συνδεδεμένα μεταξύ τους με τέτοιο τρόπο ώστε σε κάθε κύκλο ρολογιού τα δεδομένα του ενός να προωθούνται στο επόμενο σε μια λογική ''οριζόντιας στοίβας''. Μόνη εξαίρεση αποτελεί το τελευταίο PE κάθε σειράς αφού δεν χρειάζεται να προωθήσει τα δεδομένα σε κάποιο επόμενο. Ο υπολογισμός των διαφορών γίνεται **παράλληλα**, κάτι που σημαίνει ότι μόλις φορτωθούν τα δεδομένα των δύο block, σε έναν μόλις κύκλο ρολογιού έχουμε τη διαφορά τους.

Κάθε PE όπως είπαμε και προηγουμένως παράγει τελικά ως αποτέλεσμα μια λέξη 8-bit, επομένως μετά από τη σύγκριση κάθε θέσης λαμβάνουμε συνολικά 256 λέξεις των 8 bit (συνολικά 2048 bit), που κάθε μια αντιστοιχεί στη διαφορά ανάμεσα σε δύο pixel. Το επόμενο βήμα είναι η άθροιση αυτών των τιμών ωστε να βγεί το τελικό αποτέλεσμα της σύγκρισης με βάση το κριτήριο SAD. Η άθροιση



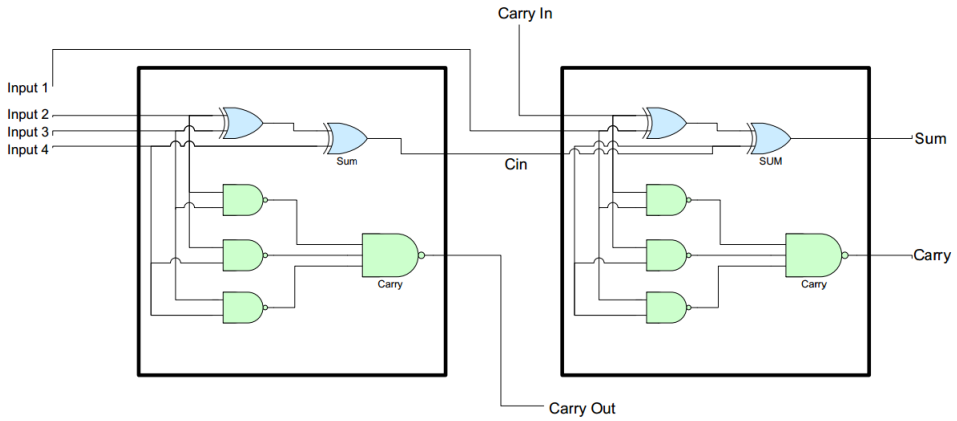

**Σχήμα 3.4**: Ένα Στοιχείο Επεξεργασίας (Processing Element)




**Σχήμα 3.5**: Η δομή του PE Array

μπορεί να γίνει με πολλούς τρόπους όπως για παράδειγμα με χρήση αθροιστών ριπής . Δυστυχώς το βασικό μειονέκτημα αυτής της πρακτικής είναι η πολύ μεγάλη καθυστέρηση που εισάγεται λόγω της μετάδοσης του κρατουμένου. Η καθυστέρηση

αυτή θα καθιστούσε απαγορευτική τη χρήση του κυκλώματος σε εφαρμογές real-time, επομένως θα πρέπει να βρεθεί ένας τρόπος ώστε η πρόσθεση να μπορεί να γίνει γρήγορα και με ένα βαθμό παραλληλίας. Μία λύση στο πρόβλημα αυτό είναι η χρήση **συμπιεστών 4:2 (4:2 compressors)** σε διάταξη δέντρου.



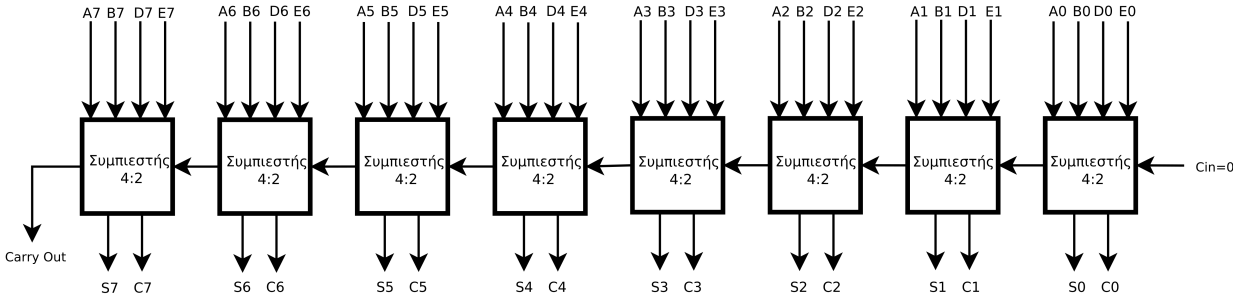

**Σχήμα 3.6**: Η δομή του 4:2 compressor

Ο συμπιεστής 4:2 αθροίζει συνολικά 4 bit τη φορά. Χρησιμοποιώντας λοιπόν 8 τέτοιους συμπιεστές είναι δυνατόν να έχουμε παράλληλη άθροιση 4 byte, με το αποτέλεσμα να είναι μια λέξη των 9 bit. Χρησιμοποιώντας αυτή τη δομή, μπορούμε να υλοποιήσουμε ένα ”δέντρο” αθροιστών. Στο πρώτο στάδιο χωρίζουμε τα 256 byte που έδωσε το PE Array σε 16 ομάδες των16 bytes η κάθε μία. Κάθε τέτοια ομάδα θα αθροιστεί από τέσσερις αθροιστές (συμπιεστές) των τεσσάρων byte ο καθένας σε συνδυασμό με έναν συμπιεστή των 10-bit. Αυτός ο δεύτερος αθροιστής, αθροίζει λέξεις των 10-bit.

**Πίνακας 3.2**: Πρόσθεση 2 byte με χρήση συμπιεστή

| **Carry Out** | $S_8$ | $S_7$ | $S_6$ | $S_5$ | $S_4$ | $S_3$ | $S_2$ | $S_1$ |
|---|---|---|---|---|---|---|---|---|
| | | | | + | | | | |
| $C_8$ | $C_7$ | $C_6$ | $C_5$ | $C_4$ | $C_3$ | $C_2$ | $C_1$ | $C_0$ |

Στη συνέχεια, ακολουθείται η ίδια λογική, τροφοδοτώντας τα αποτελέσματα του πρώτου επιπέδου στο δεύτερο που αποτελείται από τέσσερις συμπιεστές των 12-bit και του δεύτερου στο τρίτο που έχει έναν μοναδικό συμπιεστή των 14-bit. Το τελικό αποτέλεσμα στην έξοδο του αθροιστή δέντρου είναι μια ποσότητα 16-bit που αποτελεί το SAD μεταξύ του candidate και του reference block. Αυτή η ποσότητα




**Σχήμα 3.7**: Η δομή ενός 4-byte αθροιστή

δρομολογείται στη συνέχεια στη μονάδα σύγκρισης, για την οποία θα μιλήσουμε παρακάτω.

### 3.4.2 Μονάδα Τοπικής Μνήμης

Η μονάδα τοπικής μνήμης (Local Memory Unit) είναι ο βασικός μηχανισμός με τον οποίο αφ'ενός τροφοδοτούμε με δεδομένα το PE Array και αφ'ετέρου εφαρμόζουμε την επαναχρησιμοποίηση δεδομένων με τον τρόπο που περιγράψαμε προηγουμένως. Στη μονάδα αυτή περιλαμβάνονται δύο επιμέρους στοιχεία, ο **αποπολυπλέκτης εισόδου (Demux)** και η **μονάδα μνήμης (Memory Unit)**, η οποία με τη σειρά της αποτελείται από τρία μικρότερα modules μνήμης, τις Submemories 1, 2 και 3.

Ο αποπολυπλέκτης εισόδου καθορίζει την αρχική ροή των δεδομένων από την εξωτερική μνήμη (RAM) στο εσωτερικό του επιταχυντή. Λειτουργεί ως διεπαφή (interface) μεταξύ της εξωτερικής και της τοπικής μνήμης και ανάλογα με την τιμή του selection signal τα δεδομένα μπορεί να δρομολογηθούν απευθείας στο SAD Unit ή σε κάποια από τις submemories. Το σήμα ελέγχου προέρχεται από τη μονάδα ελέγχου, στην οποία θα αναφερθούμε εκτενώς αργότερα.

**Πίνακας 3.3**: Τα σήματα ελέγχου του αποπολυπλέκτη και το routing των δεδομένων

| **Σήμα Ελέγχου** | **Προορισμός Δεδομένων** |
|---|---|
| 00 | SAD Unit |
| 01 | Submemory #1 |
| 10 | Submemory #2 |
| 11 | Submemory #3 |

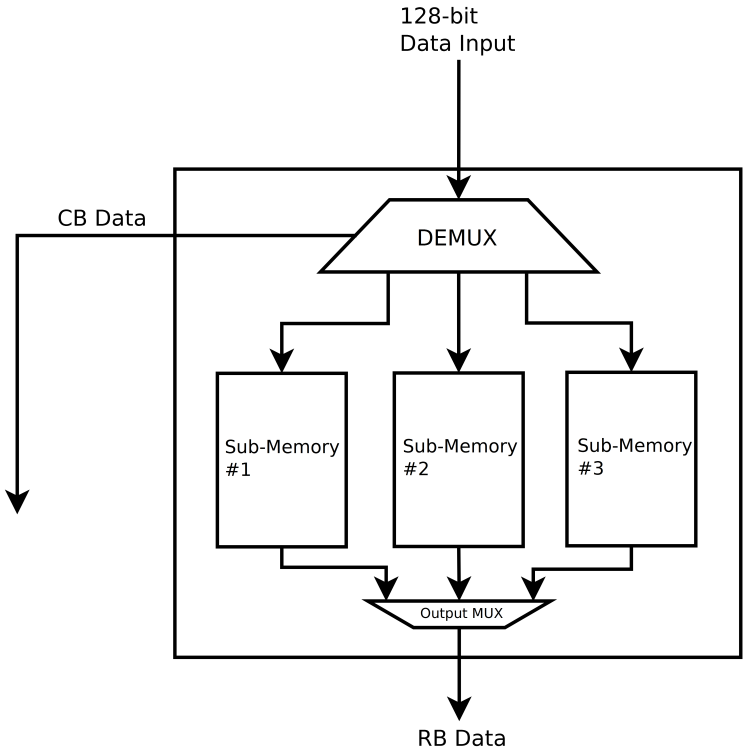
Η Μονάδα Τοπικής Μνήμης δέχεται ως είσοδο ένα δίαυλο δεδομένων των 128-bit.

Με δεδομένο ότι ο επιταχυντής σχεδιάστηκε για να εφαρμόσει τεχνικές συμπίεσης σε εικόνες greyscale, το κάθε pixel της εικόνας αποτελείται από 8-bit. Ως εκ τούτου, η μονάδα μπορεί να δεχτεί κάθε φορά 16 pixel. Στο σημείο αυτό μπορούμε να αναφερθούμε στη δομή των submemory modules (υπο-μνήμες). Κάθε submemory λοιπόν αποτελείται από μια συστοιχία 256 καταχωρητών των 8-bit. Η συστοιχία αυτή έχει δομή 16×16 και κάθε καταχωρητής μπορεί να αποθηκεύσει την τιμή ενός pixel. Κατά συνέπεια, μια submemory μπορεί να αποθηκεύσει ένα τμήμα εικόνας μεγέθους 16 ×16 pixel. Η εγγραφή των δεδομένων γίνεται γραμμή-προς-γραμμή, ενώ η ανάγνωση στήλη-προς-στήλη. Ο βασικός λόγος για τον οποίο δε χρησιμοποιήθηκε κάποια άλλη δομή μνήμης, είναι η εξαιρετική πολυπλοκότητα της διευθυνσιοδότησης. Πέρα από το γεγονός ότι δε θα είχε κάποια πρακτική χρησιμότητα, η συστηματική διευθυνσιοδότηση θα απαιτούσε interfaces τα οποία θα εισήγαν καθυστέρηση και περιορισμούς στη συχνότητα ρολογιού. Στη συγκεκριμένη περίπτωση τη διευθυνσιοδότηση αναλαμβάνει ένας απλός **μετρητής**, ο οποίος ελέγχεται από τη μονάδα ελέγχου, και αυξάνεται σε κάθε κύκλο ρολογιού. Με τη χρήση του μετρητή μπορεί να γίνει απευθείας προσπέλαση σε μια στήλη της τοπικής μνήμης. Η αρχιτεκτονική αυτή εκτός των άλλων εξυπηρετεί φυσικά τη λογική της επαναχρησιμοποίσης δεδομένων. Το σχήμα 3.8 περιγράφει τη δομή της τοπικής μνήμης.

Σε προηγούμενη ενότητα αναφερθήκαμε στο πώς γίνεται η σύγκριση του candidate block με κάθε μία δυνητική θέση της περιοχής αναζήτησης (candidate blocks). Αυτός ο τρόπος σάρωσης της περιοχής αναζήτησης ονομάζεται **raster scan**. Ο τρόπος με τον οποίο πραγματοποιούμε αυτή τη σάρωση φαίνεται στο σχήμα 3.9, όπου απεικονίζεται η απαιτούμενη δομή δεδομένων για μια περιοχή αναζήτησης 32×32 pixel, με current block 16×16 pixel.

Η περιοχή αναζήτησης χωρίζεται σε τέσσερα μέρη, το καθένα από τα οποία έχει μέγεθος 16×16 pixel. Προκειμένου να υπολογίσουμε τα τελευταία pixel των θέσεων 2 και 4 θα χρειαστούμε ακόμα 16×15 pixel και αυτός είναι και ο ρόλος της γραμμοσκιασμένης περιοχής. Για το λόγο αυτό χρησιμοποιείται η τρίτη submemory (submemory #3) στην οποία φορτώνονται αυτά τα επιπλέον pixel. Για τις θέσεις 3 και 4 τα επιπλέον pixel (15×16) μπορούν να φορτωθούν με διαδοχικές προσβάσεις στις τρείς υπομνήμες. Πρακτικά ο ''αλγόριθμος'' με τον οποίο φορτώνεται μια περιοχή αναζήτησης και γίνεται αναζήτηση ενός CB είναι η εξής:

- Αρχικά, κατά τους πρώτους 16 κύκλους ρολογιού, φορτώνεται το CB απευθείας στο PE Array, στήλη-προς-στήλη.

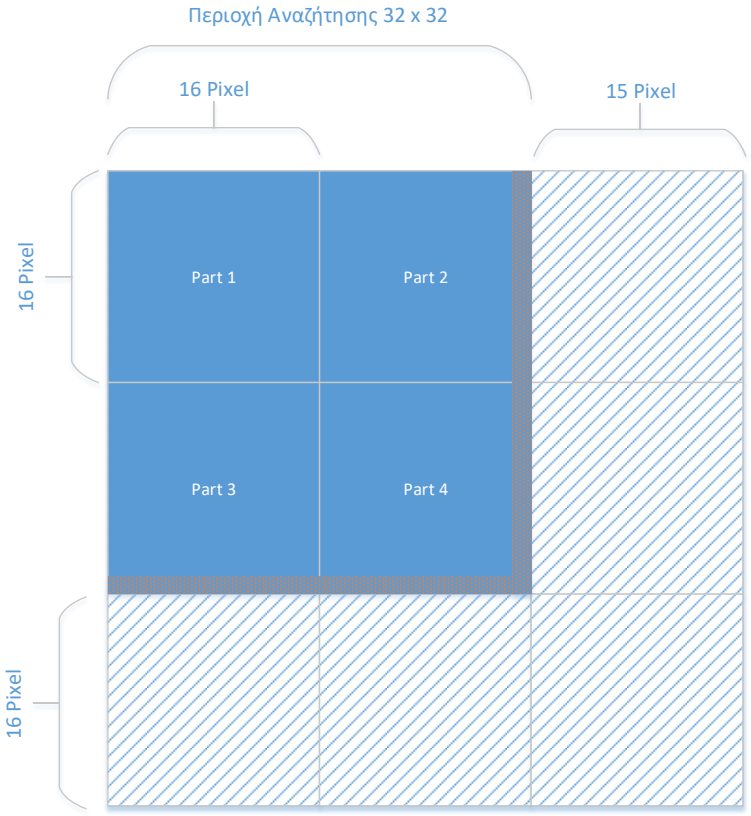



**Σχήμα 3.8**: Η αρχιτεκτονική της τοπικής μνήμης

- Στους επόμενους 16 κύκλους φορτώνεται το πρώτο τμήμα της περιοχής αναζήτησης στην πρώτη υπο-μνήμη σειρά-προς-σειρά.

- Μόλις φορτωθεί το πρώτο τμήμα της περιοχής αναζήτησης στη μνήμη ξεκινάει αμέσως η φόρτωσή του στο PE Array, στήλη προς-στήλη. Αυτό όπως θα δούμε στη συνέχεια εξυπηρετεί τους σκοπούς της επαναχρησιμοποίησης δεδομένων. Ταυτόχρονα, ξεκινάει η φόρτωση του δεύτερου τμήματος της περιοχής αναζήτησης στην δεύτερη υπο-μνήμη, η οποία και διαρκεί επίσης 16 κύκλους ρολογιού.

- Μόλις ολοκληρωθεί η φόρτωση του δεύτερου τμήματος της περιοχής αναζήτησης, ξεκινάει η προώθησή του στο PE Array, επίσης στήλη-προς-στήλη. Αξίζει να σημειώσουμε, ότι κάθε φορά που μια καινούργια στήλη της περιοχής αναζήτησης εισέρχεται στο PE Array, οι προηγούμενες που βρίσκονται ήδη σε αυτό ολισθαίνουν προς τα αριστερά (left-shifting). Εδώ, εμφανίζεται και το επίπεδο




**Σχήμα 3.9**: Απεικόνιση της περιοχής αναζήτησης για ένα CB 16×16 pixel

επαναχρησιμοποίησης A.

- Όσο προχωράει η ανάγνωση της submemory #2, η submemory #3 γεμίζει με δεδομένα. Με το τέλος των εγγραφών, μία μόνο σειρά από 16 pixel του τρίτου τμήματος της περιοχής αναζήτησης εγγράφεται στην υπο-μνήμη #1, ετοιμάζοντας την ανάγνωση από την επόμενη ”λωρίδα” της περιοχής αναζήτησης. Εδώ εμφανίζεται το επίπεδο επαναχρησιμοποίησης B.

- Στους επόμενους 2 κύκλους ρολογιού θα εγγραφεί μια σειρά από 16 pixel στις υπομνήμες #2 και #3, που θα αντιστοιχούν στο τέταρτο τμήμα της περιοχής αναζήτησης, και του γραμμοσκιασμένου χωρίου αντίστοιχα.

- Με το τέλος της ανάγνωσης της υπο-μνήμης #3, ενεργοποιείται εκ νέου η

ανάγνωση από την υπο-μνήμη #1 και έτσι εισάγεται ξανά ένα νέο RB στο PE Array, από την επόμενη ''λωρίδα'' της περιοχής αναζήτησης.

Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να εξαντληθούν όλες οι πιθανές θέσεις της περιοχής αναζήτησης.

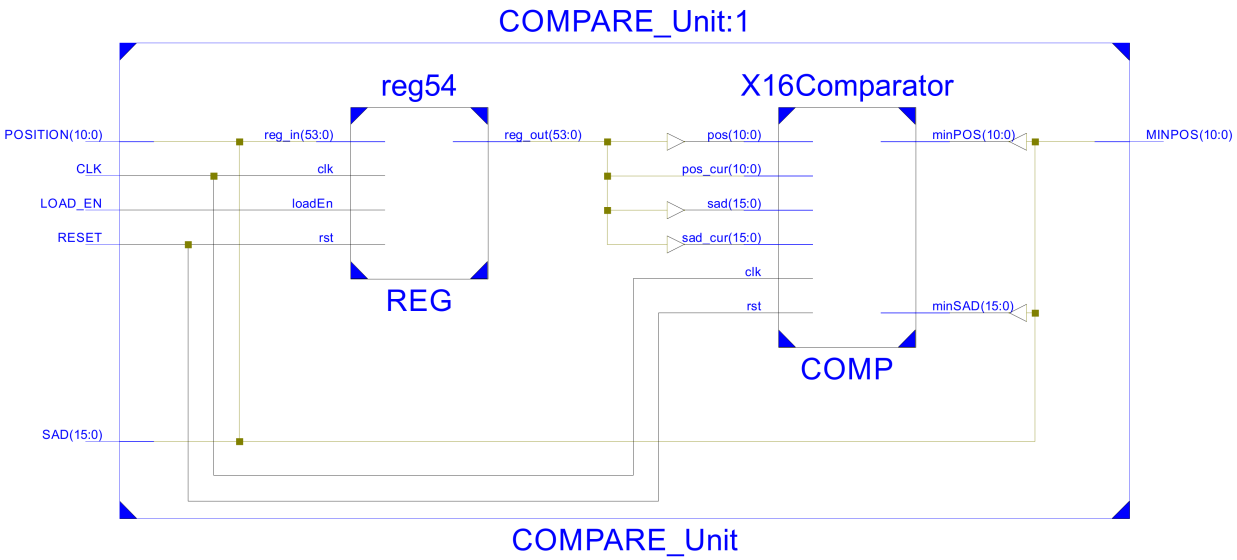
### 3.4.3 Μονάδα Συγκρισης

Η **Μονάδα Σύγκρισης (Compare Unit)** είναι ο μηχανισμός με τον οποίο αξιολογείται το υπολογισμένο SAD σχετικά με το αν αποτελεί βέλτιστο ή όχι κατα την αναζήτηση ενός CB. Η διαδικασία προκειμένου να βρούμε το μικρότερο δυνατό SAD μέσα σε μια περιοχή αναζήτησης είναι σχετικά απλή: η μονάδα σύγκρισης περιέχει έναν καταχωρητή που αποθηκεύει το ελάχιστο SAD που έχει υπολογιστεί μέχρι εκείνη τη δεδομένη χρονική στιγμή καθώς και τη θέση του, σε ποιο σημείο δηλαδή εντοπίστηκε. Αρχικά ο καταχωρητής έχει την τιμή 65.535 (και τα 16 bit είναι μονάδες), όμως καθώς η διαδικασία της εκτίμησης κίνησης προχωράει η τιμή του καταχωρητή αντικαθίσταται κάθε φορά που εντοπίζεται μία μικρότερη. Πρακτικά δηλαδή, κάθε φορά που υπολογίζουμε ένα νέο SAD το συγκρίνουμε με το SAD που είναι αποθηκευμένο στον καταχωρητή της μονάδας σύγκρισης. Αν είναι μικρότερο, παίρνει τη θέση του παλιού, αλλιώς απορρίπτεται. Με τον τρόπο αυτό είμαστε σίγουροι ότι για κάθε CB έχουμε υπολογίσει την ελάχιστη διαφορά και κατ'επέκταση το βέλτιστο διάνυσμα κίνησης.

Εκτός από την τιμή του SAD ο συγκριτής διατηρεί αποθηκευμένη και την αντίστοιχη θέση στην οποία εντοπίστηκε. Η όλη διαδικασία της εκτίμησης κίνησης όπως θα δούμε και παρακάτω ελέγχεται αυστηρά από έναν μετρητή στην κεντρική μονάδα ελέγχου, ο οποίος αυξάνεται σε κάθε κύκλο ρολογιού. Εκτός των άλλων, η τιμή αυτού του μετρητή αντιπροσωπεύει παράλληλα τη θέση στην οποία βρισκόμαστε μέσα στην περιοχή αναζήτησης (και κατ' επέκταση το τρέχον RB), οπότε μπορούμε να χρησιμοποιήσουμε αυτήν την τιμή για τοποθετήσουμε χωρικά το SAD που μόλις υπολογίσαμε.

Σχεδιαστικά, τόσο το ''παλιό'' ζεύγος SAD-θέσης όσο και το νέο, αποθηκεύονται στον ίδιο καταχωρητή ως ενα συνενωμένο (concatenated) διάνυσμα 54-bit. Στην πορεία διαμοιράζονται τα δεδομένα, με τα SAD να τροφοδοτούνται ταυτόχρονα σ' έναν συγκριτή και έναν πολυπλέκτη, ενώ οι τιμές των θέσεων απευθείας σ'ενα δεύτερο πολυπλέκτη. Στους πολυπλέκτες διαμοιράζεται ένα κοινό σήμα ελέγχου το

οποίο οδηγείται από την έξοδο του συγκριτή. Το τελικό ζεύγος SAD-θέσης συνενώνεται ξανά σ' ένα διάνυσμα 27-bit και τροφοδοτείται ξανά στον καταχωρητή της μονάδας σύγκρισης, ενώ ως τελική έξοδο λαμβάνουμε μόνο το 11-bit διάνυσμα της θέσης.



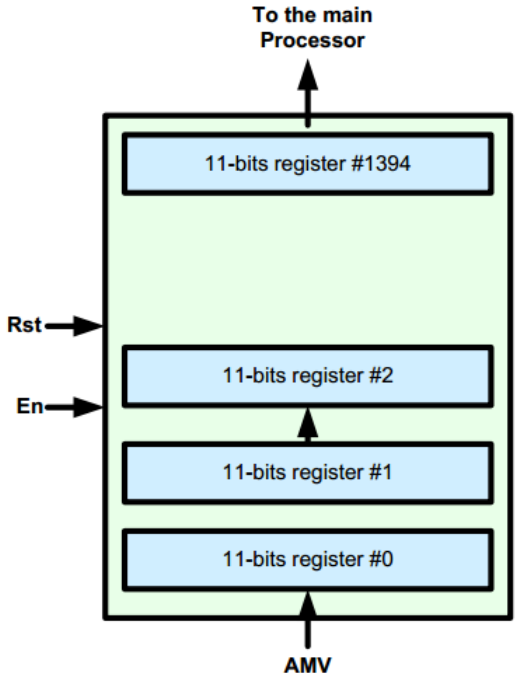

**Σχήμα 3.10**: Σχεδιάγραμμα του συγκριτή

### 3.4.4 Μνημη Διανυσματων Κινησης (Motion Vector Memory)

Η Μνήμη Διανυσμάτων Κίνησης (Motion Vector Memory ή MVMEMORY) είναι η μνήμη στην οποία αποθηκεύονται τα διανύσματα κίνησης των blocks ολόκληρου του frame. Προκειται επί της ουσίας για μια ουρά FIFO αποτελούμενη από 1395 καταχωρητές των 11-bit ο καθένας (σχήμα 3.11). Η θέση που υπολογίζεται στο τέλος της διερεύνησης ενός CB, αποτελεί το πέρας ενος διανύσματος με αρχή την αρχική θέση του CB μέσα στο frame. Ένα frame του προτύπου SDTV (Standard Definition TeleVision) έχει διαστάσεις 720×486 pixel, επομένως με block size 16×16 pixel, κάθε frame αποτελείται από συνολικά 1395 block. Καθώς η σάρωση του frame γίνεται με τη μέθοδο raster, μπορούμε να αποθηκεύσουμε τη νέα θέση που υπολογίσαμε για κάθε block στον αντίστοιχο καταχωρητή της FIFO, αφού κάθε νέα θέση που θα αποθηκεύεται θα προκαλεί ολίσθηση όλων των προηγούμενων προς τα επάνω.

Τελικά το η θέση του πρώτου block θα είναι αποθηκευμένη στο τέλος της ουράς, ενώ του τελευταίου στην αρχή.

Στο τέλος της αναζήτησης κάθε frame, η μνήμη στέλνει τα δεδομένα της στον κεντρικό επεξεργαστή και κατόπιν κάνει reset τους καταχωρητές προκειμένου να είναι έτοιμοι για τα διανύσματα του επόμενου frame.




**Σχήμα 3.11**: Motion Vector Memory

### 3.4.5 Μονάδα Ελεγχου

Η Μονάδα Ελέγχου (Control Unit) είναι το σημαντικότερο και πιο περίπλοκο στοιχείο του επιταχυντή. Οδηγεί όλα τα απαραίτητα σήματα ελέγχου για τις επιμέρους μονάδες του συστήματος και αποτελεί το συνδετικό κρίκο μεταξύ του κεντρικού επεξεργαστή και του επιταχυντή. Η μονάδα ελέγχου αποτελείται από δύο τμήματα, έναν αύξοντα μετρητή και έναν controller που παράγει τα απαραίτητα σήματα. Η μονάδα ελέγχου δέχεται τρία σήματα ως είσοδο: enable, reset, και clk. Το σήμα enable ενεργοποιεί τη διαδικασία της εκτίμησης κίνησης, το reset μηδενίζει όλους τους καταχωρητές του συστήματος, ενώ το clk τροφοδοτείται από ένα σήμα ρολογιού προερχόμενο από τον επεξεργαστή. Τα σήματα εξόδου που παράγει η μονάδα

ελέγχου είναι όλα όσα χρειάζονται για τον έλεγχο των επιμέρους μονάδων (Μονάδα Τοπικής Μνήμης, PE Array, κ.λπ.). Σε επόμενη ενότητα θα αναφερθούμε εκτενώς στην υλοποίηση της μονάδας ελέγχου και τα σήματα που αυτή παράγει.

Ο μετρητής της μονάδας ελέγχου αυξάνει σε κάθε κύκλο ρολογιού και χρησιμοποιείται για τη μέτρηση των κύκλων ρολογιού που απαιτούνται για την εκτίμηση της κίνησης κάθε block από το πρώτο (πάνω αριστερά) pixel μέχρι το τελευταίο (κάτω δεξιά). Έτσι, σε μια περιοχή αναζήτησης 32×32 pixel, οι τιμές που παίρνει ο μετρητής είναι από 000'h μέχρι 400'h. Προκειμένου να ξεκινήσει η μέτρηση θα πρέπει να ενεργοποιηθούν και τα τρία σήματα εισόδου της μονάδας ελέγχου, οπότε είναι προφανές ότι ο μετρητής μηδενίζεται με κάθε νέα αναζήτηση. Η τιμή του μετρητή αναπαριστά τη θέση του τρέχοντος CB μέσα στην περιοχή αναζήτησης και όπως είναι φυσικό, αυτή η θέση θα πρέπει στο τέλος να αντιστοιχηθεί στο συνολικό frame.

Ο controller της μονάδας ελέγχου δέχεται ως είσοδο την τιμή του μετρητή και παράγει τα ανάλογα σήματα που απαιτούνται στο συγκεκριμένο κύκλο ρολογιού. Στην προκειμένη περίπτωση ο μετρητής της μονάδας ελέγχου χρησιμεύει ως **μηχανή πεπερασμένων καταστάσεων (Finite State Machine)** με τα σήματα ελέγχου να αποκωδικοποιούν την κάθε κατάσταση. Συνολικά η μηχανή αποτελείται από 291 καταστάσεις οι οποίες ελέγχουν τη ροή της διαδικασίας και την είσοδο/έξοδο των δεδομένων.

Παρακάτω, θα αναφερθούμε σε ένα ακόμη συστατικό στοιχείο του περιφερειακού, το οποίο όμως δεν επηρεάζει τη βασική λειτουργία του και χρησιμεύει μόνο για τη διαμεταγωγή δεδομένων πάνω από τον δίαυλο επικοινωνίας AXI. Ωστόσο είναι σκόπιμο πρώτα να περιγράψουμε τη λειτουργία και την αρχιτεκτονική του πρωτοκόλλου και στην πορεία να συμπληρώσουμε το τμήμα αυτό με μεγαλύτερη ευκολία στην κατανόηση.

## 3.5 Αρχιτεκτονικη του Συστηματος

Ο επιταχυντής που περιγράψαμε παραπάνω δεν μπορεί να λειτουργήσει μόνος του. Στην πραγματικότητα αποτελεί μόνο ένα τμήμα ενός μεγαλύτερου και πολυπλοκότερου ενσωματωμένου συστήματος χτισμένου γύρω από την πλατφόρμα ZYNQ-7000 της Xilinx. Προκειμένου να υπάρξει κάποια χρησιμότητα στον επιταχυντή που υλοποιήσαμε θα πρέπει αυτός να δέχεται δεδομένα από μία κεντρική

μνήμη, και η λειτουργία του να ελέγχεται από έναν κεντρικό επεξεργαστή. Αυτά τα δεδομένα μας οδηγούν στην ανάπτυξη ενός πλήρους ενσωματωμένου συστήματος το οποίο περιλαμβάνει πολλά επιπλεόν υποσυστήματα όπως memory interfaces, clock generators και Direct Memory Access (DMA) Controllers. Το ενσωματωμένο σύστημα που υλοποιήσαμε περιλαμβάνει μια πληθώρα στοιχείων και βασίζεται σε τεχνολογίες και πρότυπα που θα είναι απαραίτητο να αναλυθούν προκειμένου να κατανοηθεί πλήρως ο τρόπος λειτουργίας του και οι δυνατότητες του.

### 3.5.1 Βασικη Δομη

Η βασική δομή του συστήματος απαρτίζεται από τρία στοιχεία: την εξωτερική μνήμη (RAM), τον κεντρικό επεξεργαστή (ARM CPU) και τον επιταχυντή. Ο τελευταίος έχει το ρόλο περιφερειακού στο σύστημα, αφού δέχεται εντολές από τον επεξεργαστή και δεδομένα από την εξωτερική μνήμη. Τα δεδομένα που αποστέλλονται στον επιταχυντή είναι λέξεις των 128 bit που αντιστοιχούν στα pixel της εκάστοτε περιοχής αναζήτησης ή του εκάστοτε CB.

Είναι προφανές ότι σε καμία περίπτωση δεν πρέπει να καθυστερεί η αποστολή των δεδομένων στον επιταχυντή, γιατί σε αντίθετη περίπτωση θα έχουμε ”data starvation” οπότε είτε θα πρέπει να εισάγουμε καθυστέρηση στον επιταχυντή είτε τελικά θα πάρουμε λανθασμένα αποτελέσματα. Η πρώτη σκέψη από πλευράς αρχιτεκτονικής προκειμένου να ελαχιστοποιήσουμε τον κίνδυνο του data starvation, είναι η απευθείας διασύνδεση της μνήμης RAM με το περιφερειακό. Αυτή η μέθοδος είναι γνωστή ως **Direct Memory Access (DMA)** και μας επιτρέπει να προσπελαύνουμε τη μνήμη χωρίς την προηγούμενη διαμεσολάβηση της CPU.

### 3.5.2 Άμεση Προσπελαση Μνημης - DMA

Οι δύο απλούστεροι τρόποι για τη μεταφορά δεδομένων μεταξύ της μνήμης και ενός περιφερειακού είναι το polling* και οι Διακοπές I/O (I/O Interrupts). Και οι δύο αυτές τεχνικές λειτουργούν καλύτερα με περιφερειακά που απαιτούν χαμηλό σχετικά εύρος ζώνης και δεν χρειάζονται αμεσότητα στην επικοινωνία τους με τη μνήμη. Επίσης, και οι δύο τεχνικές καθιστούν υπεύθυνο τον επεξεργαστή για τη διαχείριση της μεταφοράς των δεδομένων. Όπως είναι φυσικό χρειαζόμαστε έναν

*Polling: Ο ενεργός έλεγχος ανά τακτά χρονικά διαστήματα για το αν μια συσκευή έχει έτοιμα δεδομένα προς αποστολή.

καλύτερο τρόπο για επικοινωνία με περιφερειακά που που χρειάζονται μεγάλο bandwidth ή real-time δεδομένα.

Αυτός ο εναλλακτικός μηχανισμός ονομάζεται **Άμεση Προσπέλαση Μνήμης (Direct Memory Access - DMA)** και χρησιμοποιεί τις διακοπές (interrupts) μόνο για να ενημερώσει τον επεξεργαστή για το τέλος μιας μεταφοράς δεδομένων ή σε περίπτωση που υπάρξει σφάλμα. Το DMA υλοποιείται από έναν εξειδικευμένο ελεγκτή (DMA Controller - DMAC) ο οποίος οργανώνει τη μεταφορά δεδομένων από και προς το εκάστοτε περιφερειακό και τη μνήμη, ανεξάρτητα από τον επεξεργαστή. Πρακτικά, στο μοντέλο master-slave που υλοποιείται στους περισσότερους διαύλους διασύνδεσης, ο DMAC γίνεται bus master και κατευθύνει τις αναγνώσεις/εγγραφές ανάμεσα στον ίδιο και τη μνήμη RAM. Υπάρχουν τρία βήματα σε μία μεταφορά DMA:

- Ο επεξεργαστής αρχικοποιεί το DMA παρέχοντας την ταυτότητα της συσκευής, τη λειτουργία που πρόκειται να πραγματοποιηθεί, τη διύθυνση μνήμης του προορισμού ή της πηγής των δεδομένων και των αριθμό των byte που πρόκειται να μεταφερθούν.

- Το DMA ξεκινάει τη λειτουργία στο περιφερειακό αναλαμβάνοντας το arbitration του διαύλου και μεταφέροντας τα δεδομένα όποτε αυτά είναι διαθέσιμα προς μετάδοση. Επιπλέον, το DMA παρέχει τις διευθύνσεις μνήμης από και προς τις οποίες θα γίνει η ανάγνωση/εγγραφή. Αν το request απαιτεί περισσότερες από μία διαμεταγωγές το DMA ορίζει τη νέα διεύθυνση μνήμης και ξεκινάει την επόμενη διαμεταγωγή. Χρησιμοποιώντας το μηχανισμό αυτό, μία συσκευή DMA μπορεί να ολοκληρώσει μία μεταφορά δεδομένων ακόμα και πολλών kBytes χωρίς να ενοχλήσει τον επεξεργαστή. Πολλοί DMA controllers περιέχουν εσωτερική μνήμη προκειμένου να διαχειριστούν τυχόν καθυστερήσεις είτε κατά τη μεταφορά, είτε κατά το διάστημα που ανέμεναν να πάρουν τον δίαυλο στον έλεγχο τους.

- Μόλις ολοκληρωθεί η μεταφορά ο ελεγκτής στέλνει ένα interrupt στον επεξεργαστή, ώστε ο τελευταίος να ελέγξει εάν και κατά πόσο ολοκληρώθηκε σωστά η διαδικασία.

Αξίζει να σημειώσουμε πως σ' ένα σύστημα μπορούν να υπάρχουν περισσότερες της μίας συσκευές DMA. Επίσης όταν υπάρχουν πολλαπλοί δίαυλοι I/O συνήθως κάθε ένας από αυτούς φέρει έναν ελεγκτή DMA που διαχειρίζεται τις μεταφορές

δεδομένων από και προς τη μνήμη για το περιφερειακό που βρίσκεται συνδεδεμένο πάνω σ 'αυτόν τον δίαυλο. Τέλος, ιεραρχικά, μια συσκευή DMA έχει πάντα προτεραιότητα στην πρόσβαση στη μνήμη, ακόμα και σε σχέση με τον επεξεργαστή.

### 3.5.3 Το πρωτοκολλο AXI

Στο σημείο αυτό αξίζει να αφιερώσουμε μερικές παραγράφους προκειμένου να συζητήσουμε για το πρωτόκολλο **AXI**. Στη συνέχεια αυτής της ενότητας θα αναφερθούμε επίσης σε λεπτομέρειες που διέπουν το πρωτόκολλο αυτό και θα εξηγήσουμε τη σημασία που έχει για την εσωτερική αρχιτεκτονική του συστήματός μας.

Το πρότυπο **AXI (Advanced eXtensible Interface)** αποτελεί μέλος της οικογένειας πρωτοκόλλων **AMBA (Advanced Microcontroller Bus Architecture)** που σχεδιάστηκε και υλοποιήθηκε από την ARM και περιλαμβάνει συστήματα διαύλων διασύνδεσης για μικροελεγκτές. Το AMBA αποτελεί ένα ανοιχτό πρότυποαρχιτεκτονική για on-chip δίκτυα διασύνδεσης, το οποίο χρησιμοποιείται για να διασυνδέει και να διαχειρίζεται διαφορετικές λειτουργικές μονάδες σε ένα SoC. Η έκδοση 4 του προτύπου AXI που χρησιμοποιήθηκε στο σύστημα αποτελεί την τρίτη γενιά του προτύπου, που στοχεύει σε κυκλώματα υψηλών επιδόσεων και συχνοτήτων ρολογιού και εισήχθη το 2010 με την τέταρτη έκδοση του προτύπου AMBA (AMBA 4.0). Η οικογένεια διαύλων AMBA χρησιμοποιείται εκτενώς στα προϊόντα της ARM. Το σύστημα που αναπτύξαμε βασίζεται στην πλατφόρμα της Xilinx Zynq-7000 η οποία ως βασικό συστατικό της στοιχείο έχει έναν επεξεργαστή ARM και χρησιμοποιεί το πρότυπο AXI προκειμένου να πετύχει επικοινωνία μεταξύ του επεξεργαστή ARM (Processing System - PS) και της επαναπρογραμματιζόμενης λογικής (Programmable Logic - PL). Επίσης πρέπει να τονίσουμε πως η Xilinx χρησιμοποιεί αυτό το πρωτόκολλο ως την εξ' ορισμού επιλογή της για τα σύγχρονα SoC που παράγει. Ως εκ τούτου, είναι προφανές ότι οποιοδήποτε νέο περιφερειακό δημιουργούμε και σκοπεύουμε να το διασυνδέσουμε με την πλατφόρμα Zynq, θα πρέπει να επικοινωνεί πανω από το δίαυλο AXI.

Συνολικά το AXI διαθέτει τρεις τύπους interface:

- **AXI4**: Έκδοση για χρήση σε IP υψηλών επιδόσεων τα οποία χαρτογραφούνται στη μνήμη του συστήματος.
- **AXI4-Lite**: Αποτελεί υποσύνολο του AXI4. Χρησιμοποιείται για απλές, μονές διαμεταγωγές μεταξύ χαρτογραφημένων IP.

- **AXI4-Stream**: Χρησιμοποιείται σε μη χαρτογραφημένα στη μνήμη περιφερειακά, όπου απαιτείται υψηλής ταχύτητας συνεχής ροή δεδομένων.

Κάθε τύπος έχει διαφορετικές χρήσεις και πλεονεκτήματα στα οποία θα αναφερθούμε επιγραμματικά παρακάτω.

### Λειτουργια του AXI

Οι προδιαγραφές του προτύπου ορίζουν το AXI ως διεπαφή μεταξύ ενός AXI Master και ενός AXI Slave (Μοντέλο "Αφέντη-Σκλάβου") τα οποία λογίζονται ως δύο IP τα οποία ανταλλάσσουν μεταξύ τους δεδομένα. Τα AXI4 και AXI4-Lite χρησιμοποιούν πέντε διαφορετικά κανάλια:

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

Τα δεδομένα μπορούν να μεταδίδονται ταυτόχρονα και προς τις δύο κατευθύνσεις, ενώ και τα μεγέθη τους μπορούν να διαφέρουν. Αυτό υποστηρίζεται πλήρως από την ύπαρξη διαφορετικών καναλιών για διευθύνσεις και δεδομένα τόσο στις εγγραφές όσο και στις αναγνώσεις. Το όριο για το AXI4 μία ριπή μέχρι και 256 μεταδόσεων δεδομένων, ενώ το AXI4-Lite επιτρέπει μία μόνο μετάδοση ανά μεταφορά.

Σε επίπεδο υλικού, το AXI4 επιτρέπει διαφορετικά ρολόγια στον master και τον slave ενώ επιτρέπει επίσης τη χρήση **καταχωρητών διασωλήνωσης (pipeline registers)** για την εξομάλυνση των χρονικών αποκλίσεων. Το AXI4-Lite λειτουργεί κατά κανόνα όμοια με την πλήρη εκδοχή του προτύπου, με τη μοναδική διαφορά ότι δεν υποστηρίζει τη λειτουργία ριπής. Το AXI4-Stream ορίζει ένα μοναδικό κανάλι για τη μετάδοση ροής δεδομένων, το οποίο βασίζεται στο κανάλι εγγραφής δεδομένων του AXI4. Σε αντίθεση με το AXI4, το AXI4-Stream μπορεί να αποστείλει

ριπές απεριόριστων δεδομένων, με το μειονέκτημα όμως ότι οι μεταδόσεις του δεν είναι δυνατό να αναδιαταχθούν.

Είναι σημαντικό να τονίσουμε ότι το πρωτόκολλο AXI4 δεν αναγνωρίζει ούτε υπαγορεύει μια συγκεκριμένη μορφή δεδομένων (πχ συγκεκριμένη μορφή πακέτων). Αυτό πρακτικά σημαίνει ότι τα εκάστοτε IP είναι υπεύθυνα για τη σωστή κωδικοποίηση-αποκωδικοποίηση των δεδομένων.

#### IP Υποδομης (Infrastructure IPs)

Πρόκειται για IP που χρησιμοποιούνται στη σύνθεση πολύπλοκων συστημάτων ωστε να διαμορφώσουν την υποδομή των τελευταίων. Χρησιμεύουν στη μεταφορά-μετατροπή των δεδομένων εσωτερικά του συστήματος χρησιμοποιώντας διεπαφές AXI4 γενικού σκοπού. Αυτά τα IP μπορεί να περιλαμβάνουν:

- Register Slices (για pipelining)
- AXI FIFOs (buffering/μετατροπή ρολογιού)
- AXI Interconnect IP (διασύνδεση χαρτογραφημένων IP μεταξύ τους)
- AXI DMA (Direct Memory Access - μετατροπή δεδομένων από memory-mapped σε stream)

Τα παραπάνω δεν αποτελούν τερματικά δεδομένων, αλλά χρησιμοποιούνται στη διασύνδεση διαφορετικών IP σ'ενα σύστημα.

#### Υλοποιηση της Διασυνδεσης AXI στο συστημα μας

Όπως ήδη αναφέραμε, το πρότυπο AXI υποστηρίζει τόσο memory-mapped I/O όσο και streaming I/O (non-memory-mapped). Στην πρώτη περίτπωση η πρόσβαση στα περιφερειακά γίνεται χρησιμοποιώντας την ιδέα της προσπέλασης μιας συγκεκριμένης διεύθυνσης μνήμης. Αυτή η μεθοδολογία έχει ορισμένα πλεονεκτήματα, όπως η ομοιογένεια του συστήματος, δηλαδή, η πρόσβαση σε κάθε περιφερειακό μπορεί να γίνει απευθείας με μια απλή προσπέλαση μιας διεύθυνσης μνήμης. Στην περίπτωσή μας ωστόσο, αυτό το μοντέλο μας είναι δύσχρηστο για δύο λόγους:

- Πρώτον, χρειαζόμαστε άμεση επικοινωνία του επιταχυντή με τη μνήμη, λόγω μεγάλων απαιτήσεων σε ταχύτητα

- Δεύτερον, δεν υπάρχει κάποια απαίτηση για διευθυνσιοδότηση, αφου το περιφερειακό μας (ο επιταχυντής) απλώς έχει ανάγκη από δεδομένα ανεξαρτήτου σειράς, μορφής ή προέλευσης. Εν ολίγοις δεν ενδιαφερόμαστε για τη φύση των δεδομένων, παρά μόνο για την κανονικότητα στη ροή τους.

Το AXI4-Stream είναι σχεδιασμένο για τέτοιου είδους εφαρμογές, οδηγούμενες από τα δεδομένα και τη ροή τους, αγνοώντας παντελώς την ιδέα της διευθυνσιοδότησης.

Φυσικά, δεν είναι από μόνο του αρκετό για το συγκεκριμένο ενσωματωμένο σύστημα που σχεδιάσαμε. Η απαίτηση πρόσβασης στον επιταχυντή από λογισμικό, καθιστά επιβεβλημένη τη χρήση ενός **υβριδικού μοντέλου**, το οποίο συνδυάζει τη λογική των χαρτογραφημένων περιφερειακών με τη λογική της ροής δεδομένων. Αυτό ακριβώς το μοντέλο μπορεί να υλοποιηθεί με τη χρήση της AXI DMA Engine, μιας μηχανής DMA για το πρωτόκολλο AXI η οποία παρέχεται ως IP από τη Xilinx. Δουλειά της DMA engine είναι να επικοινωνεί με τον επεξεργαστή όποτε αυτό είναι απαραίτητο, προκειμένου να διαιτητεύει τη μεταφορά των δεδομένων από και προς τη μνήμη RAM.

### 3.5.4 Υλοποιηση της Αρχιτεκτονικης

Το σύστημά μας όπως αναφέραμε και προηγουμένως, χρησιμοποιεί ένα μηχανισμό DMA για τη μετακίνηση των δεδομένων, ο οποίος μάλιστα οφείλει να συμμορφώνεται με το πρωτόκολλο επικοινωνίας AXI-Stream. Το γεγονός αυτό προσθέτει ένα νέο επίπεδο πολυπλοκότητας στην υλοποίηση της αρχιτεκτονικής, καθώς, σε συνδυασμό με άλλες επιλογές που κάναμε απαιτεί ειδική προσέγγιση τόσο σε επίπεδο υλικού όσο και λογισμικού.

#### Μεθοδοι Διαμεταγωγης Δεδομενων

Η μετακίνηση δεδομένων με DMA μπορεί να γίνει με διαφορετικούς τρόπους και καθορίζεται από μια πληθώρα παραμέτρων. Αυτές μπορεί να είναι το εύρος του διαύλου δεδομένων, ο αριθμός των καναλιών επικοινωνίας και ο τρόπος με τον οποίο πραγματοποιούνται οι μεταδόσεις. Σχετικά με τις δύο πρώτες παραμέτρους η επιλογή ήταν εύκολη καθώς σε μεγάλο βαθμό είχε ήδη καθοριστεί από την αρχιτεκτονική του επιταχυντή. Επομένως η μηχανή DMA επικοινωνεί με τον επιταχυντή

πάνω από ένα κανάλι επικοινωνίας με δίαυλο εύρους 128 bit. Σαφώς και αυτό μπορεί να αλλάξει ανάλογα με την εφαρμογή. Για παράδειγμα η ύπαρξη δύο καναλιών επικοινωνίας μπορεί να ωφελήσει εφαρμογές η οποίες διαχειρίζονται μεγάλο όγκο δεδομένων ή με συγκεκριμένη μορφή, όπως για παράδειγμα 2-D πίνακες. Μια τέτοια εφαρμογή θα μπορούσε να χειρίζεται δεδομένα εικόνας ή βίντεο, οπότε να χρησιμοποιεί τα επιπλέον κανάλια για να γράφει ή να διαβάζει ένα ολόκληρο frame σε κάθε μεταφορά.

Σε ότι αφορά την τρίτη παράμετρο τα πράγματα είναι λίγο πιο περίπλοκα. Ο μηχανισμός DMA μας δίνει τη δυνατότητα να μεταφέρουμε δεδομένα με δύο τρόπους: με απευθείας προσπέλαση των καταχωρητών του ελεγκτή DMA και με τη χρήση Scatter-Gather descriptors. Στην πρώτη περίπτωση για κάθε μεταφορά εφαρμόζεται η εξής διαδικασία:

- Γίνεται αίτηση στον επεξεργαστή (interrupt request) για μία μεταφορά δεδομένων
- Ο επεξεργαστής περνάει τον έλεγχο στον ελεγκτή DMA προκειμένου να εκτελέσει τη μεταφορά
- Με την ολοκλήρωση της μεταφοράς ο ελεγκτής DMA ενημερώνει τον επεξεργαστή για την ολοκλήρωση με νέο interrupt
- Η διαδικασία επαναλαμβάνεται για κάθε μεταφορά δεδομένων

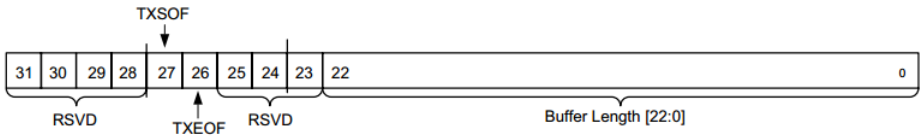
Στο σημείο αυτό αξίζει να επισημάνουμε δύο πράγματα: με τον όρο "μεταφορά" εννοούμε μία μετακίνηση δεδομένων, είτε από την κεντρική μνήμη, είτε προς αυτήν. Το μέγεθος των δεδομένων είναι άσχετο και μπορεί να φτάνει τα αρκετά Mbyte. Το δεύτερο πράγμα έχει να κάνει με τον τρόπο με τον οποίο δίνεται η εντολή στον ελεγκτή DMA και αυτό συμβαίνει συνήθως με απευθείας προσπέλαση των καταχωρητών ελέγχου του τελευταίου από τον επεξεργαστή, γι' αυτό το λόγο και στη βιβλιογραφία συναντάται ως **Direct Register Access**. Αυτός ο τρόπος μεταφοράς δεδομένων είναι ιδιαίτερα χρήσιμος και εφαρμόζεται σε μικρές διαμεταγωγές όπου τα δεδομένα βρίσκονται "συγκεντρωμένα" με μία περιοχή μνήμης. Πολλές φορές ωστόσο, στην πράξη τα δεδομένα είναι κατακερματισμένα σε αρκετές περιοχές της μνήμης, χωρίς συνέχεια. Σε αυτήν την περίπτωση είμαστε υποχρεωμένοι να

χρησιμοποιήσουμε πολλαπλές αιτήσεις μεταφοράς δεδομένων, κάτι το οποίο μπορεί τελικά να επηρεάσει αρνητικά την απόδοση του συστήματός μας.

Προκειμένου να λυθεί αυτό το πρόβλημα καταφεύγουμε σε μια λύση που είναι γνωστή ως **Scatter-Gather (SG - Διασπορά-Συγκέντρωση)**. Σ'αυτή τη μέθοδο, χρησιμοποιείται η λογική των **descriptors (περιγραφέων)** προκειμένου να καθοδηγηθεί ο ελεγκτής DMA σχετικά με τις μεταφορές δεδομένων που πρέπει να εκτελέσει. Κάθε descriptor περιλαμβάνει οδηγίες σχετικά με τη διεύθυνση από την οποία πρέπει να αναγνωστούν (ή να εγγραφούν) τα δεδομένα, το μέγεθος της μεταφοράς που πρόκειται να γίνει, καθώς επίσης και μια διεύθυνση μνήμης η οποία δείχνει που βρίσκεται ο επόμενος descriptor. Με τον τρόπο αυτό δημιουργείται μια "αλυσίδα" από descriptors με τον καθένα να περιέχει τις οδηγίες προς το μηχανισμό DMA για μία μεταφορά δεδομένων. Ο πρώτος descriptor της αλυσίδας ονομάζεται **head descriptor** ενώ ο τελευταίος **tail descriptor**. Οι descriptor αποθηκεύονται είτε στη μνήμη RAM είτε σε ειδική εξωτερική μνήμη που προορίζεται γι'αυτόν το σκοπό. Συνεπώς η διαδικασία η οποία εφαρμόζεται στη μέθοδο Scatter-Gather είναι η εξής:

- Γίνεται αίτηση στον επεξεργαστή (interrupt request) για μία μεταφορά δεδομένων
- Ο επεξεργαστής περνάει τον έλεγχο στον ελεγκτή DMA προκειμένου να εκτελέσει τη μεταφορά δίνοντας τη διεύθυνση του head descriptor
- Ο ελεγκτής DMA ενεργοποιεί τον μηχανισμό Scatter-Gather και ξεκινάει τη μεταφορά σύμφωνα με τις οδηγίες του head descriptor
- Στη συνέχεια ακολουθεί τις οδηγίες του επόμενου descriptor. Ο τελευταίος βρίσκεται στη θέση μνήμης που καθόρισε ο δείκτης του head descriptor
- Η διαδικασία εκτελείται για κάθε descriptor, μέχρι να φτάσουμε στον tail descriptor, οπότε και σημαίνει το τέλος της μετάδοσης

Όπως είναι προφανές, η χρήση του μηχανισμού Scatter-Gather δίνει έναν μεγαλύτερο βαθμό αυτονομίας στη μηχανή DMA, αφού αρχικά ο επεξεργαστής μπορεί να καθορίσει μέσω λογισμικού την αλληλουχία των μεταφορών δεδομένων που πρέπει να πραγματοποιηθούν με τους descriptors, τους οποίους χρησιμοποιεί κατόπιν η μηχανή DMA για να πραγματοποιήσει όσες μεταφορές χρειάζονται. Η γενική μορφή ενός descriptor φαίνεται στον πίνακα 3.4




**Σχήμα 3.12**: AXI MM2S Control Field

Οι descriptors αποτελούνται από 13 πεδία των 4-byte ή 32-bit. Μπορούν να διαχειριστούν δεδομένα που προέρχονται τόσο από διευθύνσεις μνήμης των 32-bit όσο και από διευθύνσεις μνήμης των 64-bit. Αυτός είναι και ο λόγος για τον οποίο σε ορισμένες εντολές του descriptor υπάρχουν δύο πεδία, με το ένα να καθορίζει τα 32 περισσότερο σημαντικά bit (MSB) και το άλλο τα 32 λιγότερο σημαντικά bit (LSB). Στην περίπτωση που δουλεύουμε με διευθύνσεις των 32 bit, το σκέλος που καθορίζει τα 32 MSB έχει τη δεκαεξαδική τιμή 0x00000000. Στο σημείο αυτό αξίζει να αναφερθούμε ειδικά στο control field του descriptor και τη δομή του. Όπως φαίνεται και στο σχήμα 3.12 υπάρχουν συνολικά 7 bit (#23-#25, #28-#31) τα οποία είναι κατειλημμένα (reserved) ενώ τα bit #26 και #27 σημαίνουν την αρχή και το τέλος ενός frame. Τα bit #0 - #22 χρησιμοποιούνται για να αναπαραστήσουν το μέγεθος της μετάδοσης σε bytes. Αυτό σημαίνει ότι πρακτικά κάθε descriptor μπορεί να μεταφέρει μέχρι και 8 Mbytes δεδομένων.

Οι descriptors στο σύστημά μας αποθηκεύονται σε μία μικρή τοπική μνήμη, ξεχωριστή από την κύρια RAM, μεγέθους 8 kb. Αυτή η μνήμη είναι τύπου Block-RAM (BRAM) και αποτελεί τμήμα της επαναπρογραμματιζόμενης λογικής του Zynq. Αυτή η προσέγγιση έχει δύο βασικά πλεονεκτήματα: το πρώτο είναι η πλήρης απομόνωση των δεδομένων των descriptor από τα υπόλοιπα δεδομένα της μνήμης του συστήματος και το δεύτερο ότι στη φάση της προτοτυποποίησης προσδίδει ευελιξία για πειραματισμούς με διάφορες συχνότητες ρολογιών (κατά κανόνα υψηλές) και πλήθος descriptor. Φυσικά για πιο ''συμπαγή'' συστήματα μπορεί να χρησιμοποιηθεί η κύρια μνήμη μειώνοντας την πολυπλοκότητα και το απαιτούμενο υλικό.

### AXI Stream

Σε προηγούμενες παραγράφους αναφερθήκαμε στο πρωτόκολλο AXI Stream, ως το βασικό πρωτόκολλο που διέπει την επικοινωνία του περιφερειακού μας με το

**Πίνακας 3.4**: Δομή ενός SG descriptor

| Addr. Space Offset | Όνομα | Περιγραφή |
|---|---|---|
| 00h | NXTDESC | Δείκτης του επόμενου descriptor - 32 LSB |
| 04h | NXTDESC_MSB | Δείκτης του επόμενου descriptor - 32 MSB |
| 08h | BUFFER_ADDRESS | Θέση μνήμης από όπου θα γίνει ανάγνωση ή εγγραφή δεδομένων - 32 LSB |
| 0Ch | BUFFER_ADDRESS_MSB | Θέση μνήμης από όπου θα γίνει ανάγνωση ή εγγραφή δεδομένων - 32 MSB |
| 10h | RESERVED | - |
| 14h | RESERVED | - |
| 18h | CONTROL | Δεδομένα που θα εγγραφούν στον control register της μηχανής AXI DMA |
| 1Ch | STATUS | Δεδομένα που θα εγγραφούν στον status register της μηχανής AXI DMA |
| 20h | APP0 | User Application Field 0 |
| 24h | APP1 | User Application Field 1 |
| 28h | APP2 | User Application Field 2 |
| 2Ch | APP3 | User Application Field 3 |
| 30h | APP4 | User Application Field 4 |

υπόλοιπο σύστημα. Ωστόσο θα πρέπει να εξηγήσουμε τον τρόπο με τον οποίο λειτουργεί αυτή η συγκεκριμένη μορφή του πρωτύπου AXI, την αρχιτεκτονική της, τα σήματα που χρησιμοποιεί και πώς αυτά τα χαρακτηριστικά επηρεάζουν τον τρόπο με τον οποίο αλληλεπιδρά το σύστημα με τον επιταχυντή.

Το πρωτόκολλο AXI Stream χρησιμοποιείται για τη μετάδοση δεδομένων μεταξύ δύο περιφερειακών τα οποία ωστόσο δεν είναι απαραίτητο να είναι χαρτογραφημένα στο σύστημα. Αυτό σημαίνει ότι οι μεταδόσεις δεν διέπονται από κάποιας μορφής διευθυνσιοδότηση, αλλά από έναν άλλο μηχανισμό χειραψίας (handshaking) ο οποίος είναι χαρακτηριστικός του πρωτοκόλλου. Οι μεταδόσεις δεδομένων στο AXI Stream χωρίζονται σε επίπεδα:

- **Μετάδοση (Transfer)**: Μία μονή μετάδοση δεδομένων μεταξύ δύο περιφερειακών. Καθορίζεται από ένα συμβάν χειραψίας (TVALID-TREADY handshaking)
- **Πακέτο (Packet)**: Το πακέτο συμβολίζει τη μαζική μεταφορά ενός αριθμού bytes. Μπορεί να περιλαμβάνει μία ή περισσότερες μεταδόσεις. Είναι ένας εύκολος τρόπος για τα περιφερειακά υποδομή να διαχειρίζονται τη ροή των δεδομένων.
- **Πλαίσιο (Frame)**: Η ανώτερη βαθμίδα ενθυλάκωσης στο πρωτόκολλο AXI Stream. Κάθε frame περιέχει ακέραιο αριθμό πακέτων και μεταφέρει μεγάλο όγκο δεδομένων.
- **Ροή (Stream)**: Ως ροή αναφέρουμε τη μεταφορά δεδομένων από μία πηγή σε έναν τελικό προορισμό.

Πριν αναλύσουμε το μηχανισμό της χειραψίας (handshaking) και της μετάδοσης δεδομένων παραθέτουμε στον πίνακα 3.5 τη λίστα με τα σήματα που χρησιμοποιεί το πρωτόκολλο.

Η διαδικασία του handshaking γίνεται με την ενεργοποίηση δύο σημάτων, του TVALID και του TREADY. Τα σήματα αυτά πρέπει να ενεργοποιηθούν με την κατάλληλη σειρά και να μείνουν ενεργοποιημένα για συγκεκριμένο χρονικό διάστημα προκειμένου μια μετάδοση να είναι επιτυχής. Ένας master οδηγεί το σήμα TVALID και το ενεργοποιεί όταν έχει έτοιμα δεδομένα προς μετάδοση. Από την άλλη πλευρά, ένας slave οδηγεί το σήμα TREADY και το ενεργοποιεί μόνο όταν είναι έτοιμος να

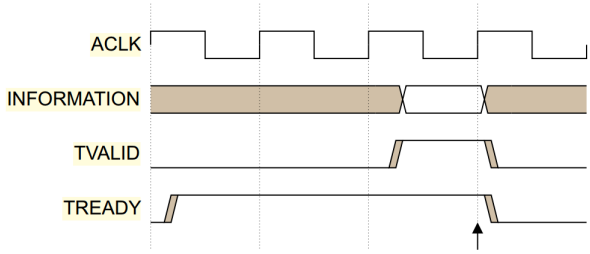
**Πίνακας 3.5**: Τα σήματα του AXI Stream

| **Σήμα** | **Πηγή** | **Περιγραφή** |
|---|---|---|
| ACLK | Clock | Καθολικό σήμα ρολογιού. Όλα τα σήματα δειγματοληπτούνται κατά τη θετική ακμή του. |
| ARESETn | Reset | Καθολικό σήμα επαναφοράς. Είναι active-LOW. |
| TVALID | Master | Υποδεικνύει ότι υπάρχει μια έγκυρη μετάδοση σε εξέλιξη. Μια μετάδοση αρχίζει όταν ενεργοποιηθεί τόσο το TVALID όσο και το TREADY. |
| TREADY | Slave | Υποδεικνύει ότι ο slave είναι έτοιμος να δεχτεί δεδομένα στον ίδιο κύκλο ρολογιού. |
| TDATA[(8n-1):0] | Master | Πρόκειται για το σήμα που φέρει τα χρήσιμα δεδομένα της μετάδοσης (payload) |
| TSTRB[(n-1):0] | Master | Το σήμα χρησιμοποιείται για τη στοίχιση των δεδομένων και υποδεικνύει αν συγκεκριμένα byte θα αξιολογηθούν ως bytes δεδομένων ή bytes θέσης |
| TKEEP[(n-1):0] | Master | Χρησιμεύει στο να καθοριστεί ποια bytes αποτελούν χρήσιμο τμήμα της ροής δεδομένων και ποια όχι. Τα bytes που έχουν το TKEEP στο 0 μπορούν να αφαιρεθούν από τη ροή των δεδομένων |
| TLAST | Master | Υποδεικνύει το τέλος ενός πακέτου |
| TID[(i-1):0] | Master | Χρησιμεύει για την ταυτοποίηση μια ροής δεδομένων, και μπορεί να υποδεικνύει διαφορετικές ροές |
| TDEST[(d-1):0] | Master | Περιλαμβάνει δεδομένα δρομολόγησης των δεδομένων |
| TUSER[(u-1):0] | Master | Επιπλέον παράλληλο σήμα το οποίο μπορεί να μεταδοθεί μαζί με την πληροφορία του TDATA και περιλαμβάνει πληροφορίες που ορίζονται από τον χρήστη |

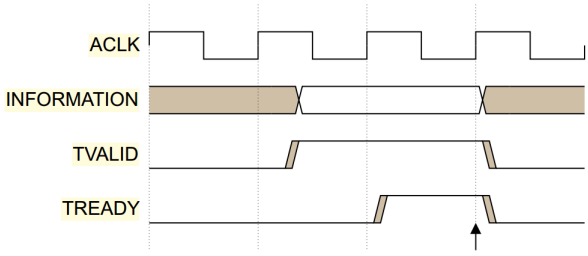
λάβει δεδομένα. Μόνο όταν ο master λάβει το TREADY και εφόσον έχει **ήδη** ενεργοποιημένο το TVALID μπορεί να ξεκινήσει η μετάδοση. Γενικά το handshaking διέπεται από τους εξής κανόνες:

- Μια συσκευή που παίζει το ρόλο του master **απαγορεύεται** να περιμένει το σήμα TREADY του slave για να ενεργοποιήσει το TVALID
- Μια συσκευή που παίζει το ρόλο του slave επιτρέπεται να περιμένει το σήμα TVALID πριν ενεργοποιήσει το TREADY
- Αν μια συσκευή που παίζει το ρόλο του slave ενεργοποιήσει το TREADY μπορεί να το απενεργοποιήσει **μόνο** αν δεν έχει ενεργοποιηθεί ακόμα το TVALID
- Γενικά, πρώτα ενεργοποιείται το TVALID και μετά το TREADY, διαφορετικά μπορούν να ενεργοποιηθούν ταυτόχρονα, στον ίδιο κύκλο ρολογιού

Στα σχήματα παρακάτω φαίνονται τα διαφορετικά σενάρια χειραψίας. Σε κάθε σχήμα το βέλος υποδεικνύει πότε ξεκινάει η μετάδοση των δεδομένων.

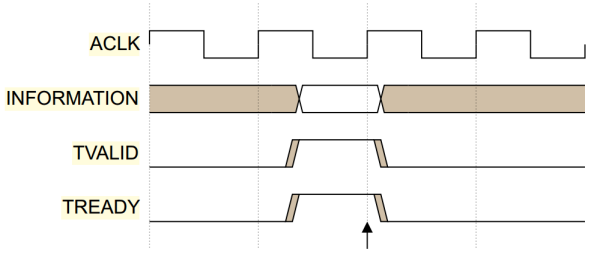



**Σχήμα 3.13**: Ενεργοποίηση του TREADY πριν το TVALID




**Σχήμα 3.14**: Ενεργοποίηση του TVALID πριν το TREADY

Προκειμένου να μπορέσει ο επιταχυντής να επικοινωνήσει πάνω από το πρωτόκολλο AXI Stream έπρεπε να τον μετατρέψουμε κατάλληλα ωστε να ενσωματώνει

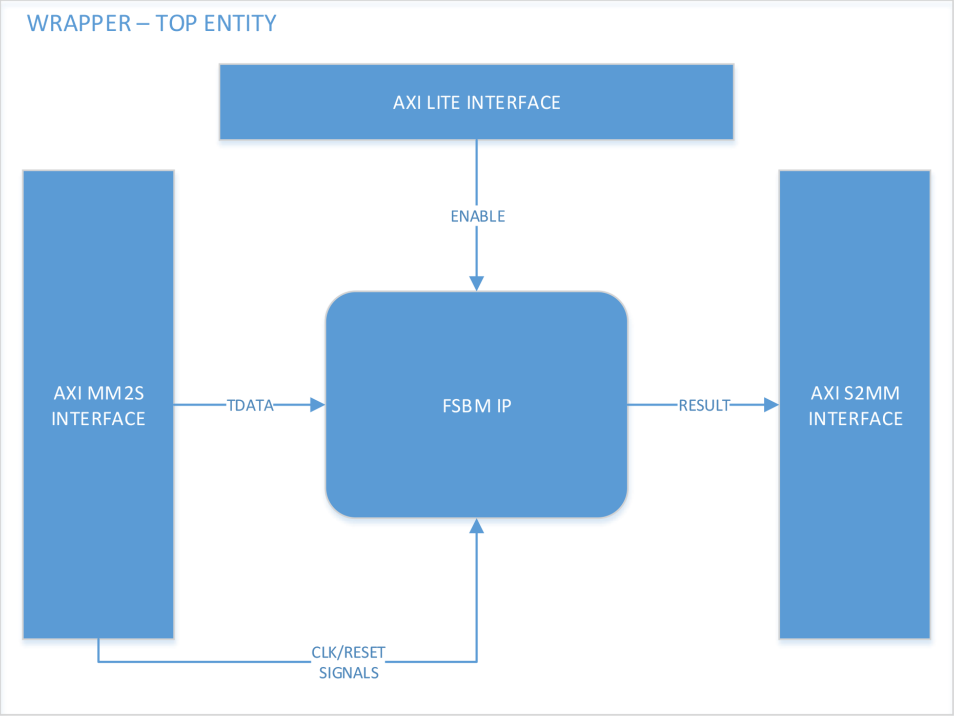



**Σχήμα 3.15**: Ταυτόχρονη ενεργοποίηση του TVALID και του TREADY

τα παραπάνω σήματα ελέγχου. Για να το πετύχουμε αυτό δημιουργήσαμε έναν wrapper, δηλαδή ένα πρόσθετο κύκλωμα το οποίο περιγράψαμε σε VHDL και το οποίο περιλαμβάνει τον επιταχυντή αλλά και τρία AXI interfaces. Τα δύο από αυτά είναι AXI Stream και χρησιμοποιούνται για τη μετακίνηση δεδομένων από και προς τον επιταχυντή. Το ένα εξυπηρετεί το κανάλι **Memory-Mapped to Stream (MM2S)**, χρησιμεύει στη μεταφορά δεδομένων από την κύρια μνήμη στον επιταχυντή και παίζει το ρόλο του AXI slave. Το δεύτερο interface εξυπηρετεί το κανάλι **Stream to Memory-Mapped (S2MM)**, χρησιμεύει στη μετάδοση δεδομένων από τον επιταχυντή στην κύρια μνήμη και παίζει το ρόλο του AXI master. Το τρίτο interface είναι ένα AXI Lite interface και χρησιμεύει στην ενεργοποίηση-απενεργοποίηση του επιταχυντή μέσω ενός σήματος ελέγχου.

### 3.5.5 AXI Stream Controller

Στο σημείο αυτό μπορούμε να μιλήσουμε για ένα επιπλέον συστατικό στοιχείο του περιφερειακού που δεν αναφέρθηκε προηγουμένως και χρησιμεύει στη διασύνδεση του επιταχυντή με το δίαυλο AXI. Η μονάδα AXI Stream Controller δεν είναι ζωτικής σημασίας για τη διαδικασία του υπολογισμού των διανυσμάτων κίνησης, αφού χρησιμεύει μόνο στην περίπτωση που ο επιταχυντής χρησιμοποιηθεί ως περιφερειακό AXI Stream. Εν τούτοις, είναι εξαιρετικά σημαντική στην περίπτωσή μας, αφού παράγει όλα εκείνα τα σήματα ελέγχου που καθορίζουν την επικοινωνία με βάση τους κανόνες του πρωτοκόλλου AXI Stream. Πιο συγκεκριμένα, η μονάδα αυτή χρησιμεύει στην αξιολόγηση των σημάτων **M_TREADY** και **S_TVALID** που προέρχονται από τη μηχανή DMA, καθώς επίσης των σημάτων εγγραφής του register file που προέρχονται από τη Μονάδα Ελέγχου και του σήματος **FIFOFULL** που υποδεικνύει ότι η Μνήμη Διανυσμάτων Κίνησης είναι γεμάτη, ώστε να παράξει αντί-




**Σχήμα 3.16**: Αρχιτεκτονική του Επιταχυντή με τα AXI Interfaces

στοιχα σήματα S_TREADY και M_TVALID **προς** τη μηχανή DMA, αλλά και το σήμα ανάγνωσης προς τη Μνήμη Διανυσμάτων Κίνησης.

Η μονάδα αποτελείται από έναν μετρητή και μια FSM δύο καταστάσεων. Η μία είναι η κατάσταση ανάγνωσης, κατα την οποία παράγεται το σήμα ανάγνωσης των διανυσμάτων κίνησης, καθώς επίσης και το σήμα TVALID προς τη μηχανή DMA που υποδεικνύει ότι έχουμε δεδομένα έτοιμα για μεταφορά. Η άλλη κατάσταση είναι η κατάσταση αναμονής (IDLE) στην οποία ο ελεγκτής περιμένει μέχρι να λάβει σήμα FIFOFULL ώστε να μεταβεί σε κατάσταση ανάγνωσης. Από τη στιγμή που θα ξεκινήσει η ανάγνωση των διανυσμάτων κίνησης, τα σήματα ανάγνωσης και M_TVALID πρέπει να παραμείνουν ενεργοποιημένα μέχρι να διαβαστούν και να μεταφερθούν όλα τα διανύσματα κίνησης. Σε αντίθετη περίπτωση υπάρχει σοβαρός κίνδυνος να λάβουμε τελικά λανθασμένα (corrupted) αποτελέσματα. Ο χρόνος για τον οποίο τα σήματα παραμένουν ενεργοποιημένα καθορίζεται σε κύκλους ρολογιού από τον 11-bit μετρητή που περιέχει ο ελεγκτής. Στην περίπτωσή μας είναι 1395

κύκλοι ρολογιού, αλλά για μεγαλύτερα frames (όπως για παράδειγμα frames σε High-Definition) μια απλή αλλαγή του μετρητή αρκεί για να γίνει προσαρμογή του ελεγκτή.

Ο ελεγκτής AXI Stream φροντίζει ώστε τα διανύσματα κίνησης κάθε frame να μεταφέρονται σε ένα πακέτο με χρήση ενός μόνο descriptor. Το συνολικό μέγεθος των δεδομένων που μεταφέρεται είναι περίπου 1.918 byte αν τα δεδομένα μας είναι unaligned ενώ αυξάνεται στα 5.580 byte αν γίνει alignment των δεδομένων στα 32-bit.

# 4
# Λογισμικό

Το λογισμικό αποτελεί βασικό τμήμα κάθε υπολογιστικού συστήματος. Στην περίπτωσή μας αποτελεί το συνδετικό κρίκο ανάμεσα στον χρήστη και το υλικό, αν και είναι προφανές από τη φύση του συστήματος, ότι η διαμεσολάβηση κάποιου χρήστη δεν είναι απαραίτητη, αφού ο επιταχυντής μπορεί να δουλέψει στο "παρασκήνιο" αρκεί φυσικά να έχει σταθερή ροή δεδομένων. Στην τελευταία περίπτωση το λογισμικό θα έπαιζε το ρόλο της διαιτησίας της ροής των δεδομένων. Το επίπεδο του λογισμικού χωρίζεται σε τρία επιμέρους τμήματα: το **λειτουργικό σύστημα** που εκτελείται στο κύριο επεξεργαστή, το **πρόγραμμα-οδηγό (Driver)** για το περιφερειακό που δημιουργήσαμε, και την **εφαρμογή χρήστη (User Application)**. Θα αναφερθούμε ξεχωριστά στο καθένα από αυτά, αφού το καθένα είχε τις δικές του ιδιαιτερότητες και χρήζει ειδικής προσοχής.

## 4.1 Το Λειτουργικο Συστημα

Στο ενσωματωμένο σύστημα που υλοποιήσαμε θα πρέπει να εκτελείται ένα λειτουργικό σύστημα το οποίο θα φροντίζει για την εκκίνηση του συστήματος, την κατάλληλη αναγνώριση των περιφερειακών, τη φόρτωση των κατάλληλων driver και τη παροχή μιας διεπαφής στο χρήστη ώστε να αλληλεπιδράσει με τον συνεπεξεργαστή. Όπως έχουμε ήδη αναφέρει, στο σύστημα περιλαμβάνεται ένας μικροεπεξεργαστής ARM. Αυτό πρακτικά σημαίνει ότι έχουμε μια πληθώρα λειτουργικών συστημάτων

ώστε να διαλέξουμε ποιο είναι καταλληλότερο για τις ανάγκες μας. Αυτά περιλαμβάνουν Windows, Linux, FreeRTOS και πολλά άλλα. Η επιλογή που κάναμε ήταν το Linux, αφ' ενός γιατί είναι ένα δωρεάν λειτουργικό σύστημα με μεγάλη υποστήριξη από την κοινότητα των χρηστών και έτσι είναι πιο εύκολο να αντιμετωπιστούν πιθανά προβλήματα. Ο δεύτερος λόγος είναι ότι είναι ιδιαίτερα φιλικό προς τους προγραμματιστές και τους μηχανικούς, αφού δίνει δυνατότητες όπως η παραμετροποίηση του πυρήνα, η εύκολη επικοινωνία με το hardware, και οι πολλές επεκτάσεις που διευκολύνουν την ανάπτυξη εφαρμογών.

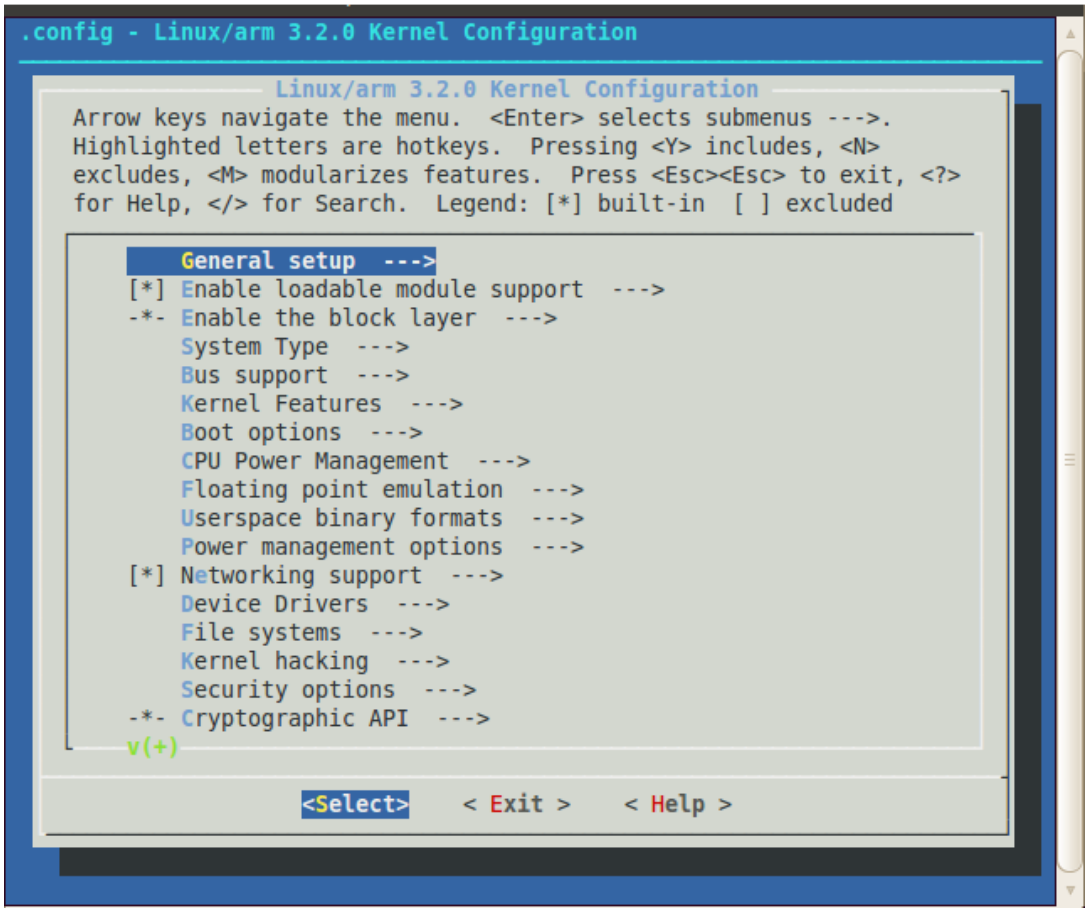
Στην προκειμένη περίπτωση χρησιμοποιήθηκε η διανομή της Xilinx, PetaLinux η οποία χρησιμοποιείται κατά κόρον σε ενσωματωμένα συστήματα. Προκειμένου να εκτελέσουμε τη συγκεκριμένη διανομή στο σύστημά μας έπρεπε να ακολουθήσουμε μια μεγάλη διαδικασία, η οποία περιλάμβανε το compile του πυρήνα (kernel), την κατάλληλη χαρτογράφηση του hardware, και τη δημιουργία ενός boot-loader ο οποίος θα αναλάμβανε την αρχικοποίηση του συστήματος και κατόπιν θα περνούσε τον έλεγχο στον kernel.

### 4.1.1 Linux Kernel

Ο πυρήνας, είναι αναμφισβήτητα το βασικότερο κομμάτι του λειτουργικού συστήματος. Πρόκειται ουσιαστικά για τον "εγκέφαλο" ο οποίος ελέγχει νευραλγικές λειτουργίες όπως η χαρτογράφηση της μνήμης και η πρόσβαση σ' αυτήν, η διαχείριση των νημάτων επεξεργασίας και η διαχείριση των I/O interrupts. Ο Linux kernel με την πάροδο του χρόνου έχει συσσωρεύσει υποστήριξη για πληθώρα τεχνολογιών και υλικού, από επεξεργαστές MIPS και απαρχαιωμένους διαύλους όπως ο ISA, μέχρι σύγχρονες μονάδες DMA και περιφερειακά Infiniband! Το χαρακτηριστικό του Linux kernel είναι ότι η αρχιτεκτονική του είναι **αρθρωτή** πράγμα που σημαίνει ότι μπορούν να προστεθούν και να αφαιρεθούν κομμάτια κατά βούληση, ανάλογα με τις απαιτήσεις μας. Προφανώς λοιπόν, όλες αυτές οι δυνατότητες είναι εντελώς περιττές για ένα - σχετικά απλό - ενσωματωμένο σύστημα, οπότε και πρέπει να αφαιρεθούν. Ταυτόχρονα, θα πρέπει να σιγουρευτούμε ότι ο πυρήνας μπορεί να υποστηρίξει τις λειτουργίες εκείνες που επιθυμούμε και που πρέπει να περιλαμβάνονται στο σύστημά μας όπως μηχανές DMA και υποστήριξη επεξεργαστών ARM.

Προκειμένου να το καταφέρουμε αυτό πρέπει να περάσουμε από μια πολύπλοκη διαδικασία παραμετροποίησης του πυρήνα κατά την οποία οφείλουμε να είμαστε πολύ προσεκτικοί αλλιώς υπάρχει μεγάλος κίνδυνος τελικά να οδηγηθούμε σε κάτι

τελείως μη-λειτουργικό. Η σουίτα εργαλείων ανάπτυξης της Xilinx για το Petalinux (Petalinux toolchain) μας δίνει τη δυνατότητα να παραμετροποιήσουμε τον πυρήνα χρησιμοποιώντας ένα απλό εργαλείο, το **menuconfig**. Το εργαλείο αυτό είναι γνωστό γενικά για τη δυνατότητα να παρέχει στον τελικό χρήστη ένα φιλικό περιβάλλον παραμετροποίησης μέσα από μενού και λίστες με χαρακτηριστικά. Στο τέλος παράγει ένα αρχείο ρυθμίσεων το οποίο τροφοδοτείται στη συνέχεια στον μηχανισμό make που αναλαμβάνει να κάνει το compile του κώδικα του πυρήνα σύμφωνα με τα χαρακτηριστικά που επιλέξαμε.




**Σχήμα 4.1**: Το περιβάλλον του menuconfig

Για το σύστημά μας δημιουργήσαμε έναν πυρήνα ο οποίος περιλαμβάνει υποχρεωτικά υποστήριξη για συγκεκριμένες τεχνολογίες. Αυτές φαίνονται αναλυτικά στον παρακάτω πίνακα:

**Πίνακας 4.1**: Οι ελάχιστες ρυθμίσεις που είναι απαραίτητο να γίνουν ωστε να είμαστε σίγουροι ότι ο kernel θα εκκινήσει στην πλακέτα

| **Ρύθμιση** | **Παράμετρος** |
|---|---|
| Target Architecture | ARM (little endian) |
| Target Architecture Variant | cortex-A9 |
| Enable NEON SIMD extension support | YES |
| Floating point strategy | VFPv3 |
| ext2/3/4 root filesystem | YES |
| ext2/3/4 variant | ext4 |

### 4.1.2 Device Tree

Προκειμένου ο πυρήνας να αναγνωρίσει το υλικό το οποίο υπάρχει στο σύστημα θα πρέπει να του παρέχουμε με κάποιο τρόπο την πληροφορία σχετικά με το τι υπάρχει και που πρόκειται να το βρει. Αυτό πρακτικά σημαίνει ότι θα πρέπει να ενημερώσουμε τον πυρήνα με ένα αρχείο περιγραφής το οποίο περιλαμβάνει όλες τις συσκευές τις οποίες θέλουμε να αναγνωρίσει, καθώς επίσης και την περιοχή της μνήμης στην οποία θέλουμε να χαρτογραφηθούν. Για να το πετύχουμε αυτό θα πρέπει να δημιουργήσουμε ένα αρχείο Device Tree με λεπτομέρειες σχετικά με τα όσα αναφέραμε. Η τυπική μορφή μια καταχώρησης σ' ένα αρχείο device tree φαίνεται στην παρακάτω απεικόνιση (Απεικόνιση 4.1)

**Τμήμα κώδικα 4.1**: Η καταχώρηση ενός απλού περιφερειακού στο Device Tree

```
FSME_Dummy_0: FSME_Dummy@43c00000 {
                        compatible = "xlnx,FSME-Dummy-1.0";
                        reg = <0x43c00000 0x10000>;
                };
```

Τα device trees είναι ένας εύκολος τρόπος να έχουμε διαφορετικά configurations συσκευών και περιφερειακών μέσα στην ίδια αρχιτεκτονική, όπως για παράδειγμα στη Zedboard. Χρησιμοποιώντας την ίδια πλατφόρμα μπορούμε με διαφορετικά device tree files να έχουμε κάθε φορά διαφορετικό σετ περιφερειακών στη διάθεσή μας. Στην καταχώρηση του περιφερειακού είναι ορατό το όνομά του (FSME_Dummy), η έκδοσή του (xlnx,FSME-Dummy-1.0) καθώς επίσης και η περιοχή μνήμης στην οποία πρέπει να χαρτογραφηθεί (reg = <0x43c00000 0x10000>). Στην προκειμένη

περίπτωση, τόσο το ενδεικτικό περιφερειακό που παρουσιάσαμε όσο και ο κανονικός επιταχυντής χαρτογραφήθηκαν στην ίδια περιοχή μνήμης με βασική διεύθυνση (base address) την 0x43c00000 και 64kByte offset (0x10000). Με αυτόν τον τρόπο ξέρουμε ακριβώς όταν εκκινήσει το λειτουργικό σύστημα, ποια περιοχή της μνήμης πρέπει να προσπελάσουμε ώστε να επικοινωνήσουμε με τον επιταχυντή.

Στο σημείο αυτό θα πρέπει να τονίσουμε ορισμένα σημαντικά πράγματα. Πρώτον, το αρχείο device tree γράφεται με μια συγκεκριμένη μορφή και συντακτικό που θυμίζει γλώσσα προγραμματισμού και στη συνέχεια γίνεται compile του αρχείου σε μια μορφή που ονομάζεται **Device Tree Blob (.dtb)** και η οποία αποτελεί εκτελέσιμο κώδικα (binary). Αυτό το αρχείο χρησιμοποιείται στη δημιουργία του bootloader και είναι το πρώτο πράγμα που διαβάζει ο kernel πριν ξεκινήσει το booting.

### 4.1.3 Διαχειριση εκκινησης - Bootloader

Ο διαχειριστής εκκίνησης ή αλλιώς **Bootloader** είναι ένα εξαιρετικά σημαντικό κομμάτι του λογισμικού του συστήματος. Ο bootloader είναι υπεύθυνος για τις εξής ενέργειες:

- Εκκίνηση του επεξεργαστή ARM
- Προγραμματισμός της FPGA με το bitstream
- Εκτέλεση του dtb
- Εκκίνηση του kernel

Ο bootloader δεν είναι παρά ένα αρχείο BOOT.BIN το οποίο για να δημιουργηθεί χρειάζονται τρία επιμέρους κομμάτια:

- Ο First-Stage Bootloader (FSBL)
- Το bitstream file (fpga.bit)
- Ο κυρίως bootloader (U-BOOT)

Ο FSBL είναι ένα εξειδικευμένο πρόγραμμα που έχει αναπτυχθεί από τη Xilinx ειδικά για την οικογένεια Zynq. Ο λόγος είναι ότι τα συγκεκριμένα SoC περιλαμβάνουν εκτός από τον επεξεργαστή και FPGA οπότε η διαδικασία εκκίνησης είναι

αρκετά πιο περίπλοκη. Η χρήση του δεν είναι προαιρετική και υπάρχουν αρκετές παραλλαγές ανάλογα με τις ανάγκες του εκάστοτε συστήματος. Η ροή εκτέλεσης του FSBL είναι αρκετά απλή και έχει ως εξής:

- Εκκίνηση του επεξεργαστή ARM (Processing System - PS) με τη ρουτίνα ps7_init
- Εντοπισμός του boot image με σάρωση όλων των μέσων αποθήκευσης
- Κάθε partition του boot image τοποθετείται στον αντίστοιχο προορισμό του. Στο σημείο αυτό γίνεται ο προγραμματισμός της FPGA.
- Φόρτωση εφαρμογής/λειτουργικού συστήματος. Στην περίπτωσή που πρόκειται για μια εφαρμογή standalone περνάει ο έλεγχος σε αυτήν. Στην περίπτωση που έχουμε κάτι μεγαλύτερο όπως ένα λειτουργικό σύστημα, ο έλεγχος περνάει στον U-BOOT (second-stage bootloader).

Ο second-stage bootloader στην περίπτωσή μας ονομάζεται U-Boot. Πρόκειται για έναν κύριο bootloader που χρησιμοποιείται κατά κόρον σε ενσωματωμένα συστήματα και υποστηρίζει πολλές αρχιτεκτονικές συμπεριλαμβανομένων των Motorola 68k, ARM, MIPS, x86 και MicroBlaze. Ο U-Boot υποστηρίζει ένα περιβάλλον γραμμής εντολών από το οποίο ο χρήστης μπορεί να επιλέξει τον kernel ή το partition που θέλει να κάνει boot. Φυσικά είναι δυνατόν να καθοριστούν default παράμετροι ώστε ο χρήστης να μην έχει απολύτως κανένα interaction με τον bootloader. Ο U-Boot είναι ένας ιδιαίτερα ευέλικτος bootloader, αφού περιλαμβάνει πλήθος εντολών μεταξύ των οποίων εντολές για φόρτωση αρχείων μέσω θύρας Ethernet ή σειριακής, εγγραφής/ανάγνωσης από διαφορετικές μνήμες και διαχείρισης των device trees.

Σε αντίθεση με πολλούς άλλους bootloader, κυρίως αυτούς που χρησιμοποιούνται στους προσωπικούς υπολογιστές, χρειάζεται μια περίπλοκη αλληλουχία εντολών προκειμένου τελικά να εκκινηθεί το λειτουργικό σύστημα. Αυτό αν και αρχικά μοιάζει μειονέκτημα, στην πράξη αποτελεί σημαντικό πλεονέκτημα του U-Boot, γιατί προσδίδει έναν επιπλέον βαθμό ευελιξίας. Οι εντολές του είναι αρκετά χαμηλού επιπέδου και δίνουν τη δυνατότητα να οριστούν με ακρίβεια παράμετροι όπως η θέση μνήμης στην οποία θα αναζητηθεί το kernel image, τα ορίσματα που θα περαστούν σε αυτό καθώς επίσης και διαφορετικές διευθύνσεις μνήμης όπου θα αναζητηθούν άλλα components όπως το device tree. Υπάρχει ακόμη και η δυνατότητα αυτο-αναβάθμισής του δίνοντας του απλώς την κατάλληλη εντολή ώστε να ψάξει για την αναβαθμισμένη έκδοση σε συγκεκριμένη θέση μνήμης!

Ο U-Boot έχει ακόμα τη δυνατότητα να εκκινήσει έναν Linux kernel χωρίς προηγουμένως να έχει οριστεί ένα σύστημα αρχείων. Ουσιαστικά αντιγράφει τα περιεχόμενα του kernel image στη μνήμη χωρίς να τα "καταλαβαίνει", και με κατάλληλη παράμετρο περνάει τον έλεγχο στον επεξεργαστή.

### 4.1.4 Σύστημα Αρχείων

Το σύστημα αρχείων είναι ένα σημαντικό τμήμα του λειτουργικού συστήματος το οποίο πρακτικά ορίζει τον τρόπο με τον οποίο γίνεται η εγγραφή και η προσπέλαση των δεδομένων, τι πληροφορίες αυτα περιλαμβάνουν και πως τοποθετούνται σε μια μη-πτητική μνήμη. Στο χώρο του Linux υπάρχει πληθώρα συστημάτων αρχείων όπως τα ext2/3/4, ReiserFS και ZFS. Στην περίπτωσή μας, επειδή η ταχύτητα είναι ένας σημαντικός παράγοντας, η δημιουργία ενός συστήματος αρχείων και η αποθήκευσή του σε μια μη-πτητική μνήμη είναι κάτι που θελήσαμε να αποφύγουμε. Επίσης λόγω της χρήσης του συστήματός μας δεν θεωρήθηκε απαραίτητη η χρήση μιας σταθερής εγκατάστασης λειτουργικού συστήματος-συστήματος αρχείων οπότε καταφύγαμε στη λύση του **Ramdisk**. Πρόκειται για έναν εναλλακτικό τρόπο να παρέχουμε το περιβάλλον ενός συστήματος αρχείων, χρησιμοποιώντας τμήμα της μνήμης RAM. Το μέγεθος του Ramdisk καθορίζεται ανάλογα με τις ανάγκες μας αλλά συνήθως 8 ή 16 MByte είναι αρκετά.

Στη συνέχεια φορτώνεται ένα συγκεκριμένο αρχείο που παράχθηκε κατά το build του Petalinux και περιλαμβάνει το filetree του λειτουργικού συστήματος μαζί με ορισμένες εφαρμογές και πιθανώς άλλα αρχεία που θέλουμε να συμπεριλάβουμε. Αυτό το αρχείο αποτελεί το RootFS και πρακτικά πρόκειται για ένα image του λειτουργικού συστήματος μαζί με το απαραίτητο σύστημα αρχείων, το οποίο αποσυμπιέζεται και τοποθετείται στο Ramdisk.

Περισσότερες πληροφορίες για τη διαδικασία και τα εργαλεία που χρησιμοποιήθηκαν θα συζητηθούν στο Κεφάλαιο 6.

## 4.2 Το Πρόγραμμα Οδήγησης (Driver)

Το λειτουργικό σύστημα μπορεί να επικοινωνήσει με ένα περιφερειακό με δύο τρόπους. Ο πρώτος είναι η απευθείας προσπέλαση της θέσης μνήμης στην οποία έχει χαρτογραφηθεί το περιφερειακό. Κάτι τέτοιο είναι εξαιρετικά απλό στην εφαρμογή του, αφού με μια εντολή εγγραφής/ανάγνωσης σε μια θέση μνήμης μπορούμε

να επικοινωνήσουμε με το εκάστοτε περιφερειακό. Η μέθοδος αυτή έχει ορισμένα πλεονεκτήματα κυρίως σε ό,τι αφορά στον έλεγχο και τη δοκιμή του περιφερειακού με συγκεκριμένα δεδομένα. Γενικά όμως πρόκειται για ένα δύσχρηστο τρόπο επικοινωνίας που δημιουργεί περισσότερα προβλήματα από όσα λύνει.

Ο δεύτερος τρόπος είναι με τη χρήση ενός **προγράμματος οδήγησης** ή **driver**. Ο driver αναλαμβάνει να παρέχει ένα υψηλότερο abstraction layer στην επικοινωνία με το περιφερειακό, μεταφράζοντας εντολές υψηλού επιπέδου σε αλληλουχία εντολών προσπέλασης της περιοχής μνήμης του περιφερειακού. Με τον τρόπο αυτό είμαστε σε θέση να γράψουμε userspace applications οι οποίες θα χρησιμοποιούν τις ρουτίνες του driver για να επικοινωνήσουν με το περιφερειακό και να αξιοποιήσουν τις δυνατότητές του.

Στην περίπτωση του συστήματός μας το περιφερειακό βρίσκεται "πίσω" από τη μηχανή AXI DMA. Επίσης, δεν λαμβάνει κάποια πολύπλοκα σήματα, παρα μόνο ένα σήμα ενεργοποίησης (ENABLE) και μια σταθερή ροή δεδομένων. Τα σήματα ρολογιού (CLOCK) και επαναφοράς (RESET) είναι καθολικά ενώ τα σήματα επικοινωνίας με το AXI καθορίζονται από την εσωτερική αρχιτεκτονική του επιταχυντή και της μηχανής DMA. Επομένως, στην πράξη δε χρειαζόμαστε κάποιο driver για το ίδιο το περιφερειακό, αλλά για τη μηχανή AXI DMA ωστε να ελέγχουμε τη ροή των δεδομένων, τα κατάλληλα interrupts και το descriptor chain προκειμένου να γίνεται με αξιόπιστο τρόπο η μεταφορά των δεδομένων από και προς τον επιταχυντή.

Για να το πετύχουμε αυτό χρησιμοποιήσαμε τον AXI DMA driver της Xilinx. Ο συγκεκριμένος driver αν και ιδιαίτερα προβληματικός, είχε το πλεονέκτημα ότι ήταν ήδη παραμετροποιημένος και μπορούσε να συμπεριληφθεί στο build του kernel. Τα χαρακτηριστικά αυτού του driver περιλαμβάνουν:

- Συμβατότητα με AXI4 και AXI4-Stream
- Υποστήριξη Scatter/Gather (SG) DMA. Όταν το Scatter/gather mode είναι ανενεργό, το IP λειτουργεί σε Simple DMA mode
- Υποστήριξη δεδομένων AXI4 Memory Map και AXI4-Stream μεγέθους 32, 64, 128, 256, 512, και 1024 bits
- Προαιρετικά, μηχανή Data Re-Alignment
- Προαιρετικά σήματα AXI Control και Status

- Multi-channel mode
- Υποστήριξη μέχρι και 64-bit Addressing
- Προαιρετική υποστήριξη Micro DMA mode

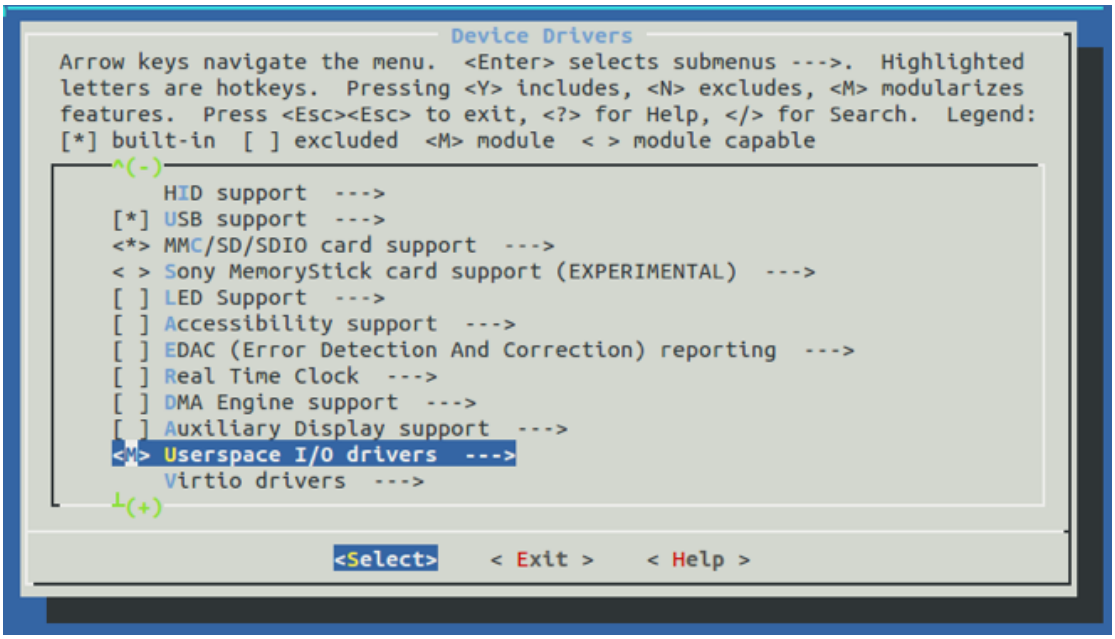
Ο εν λόγω driver περιλαμβάνεται στο kernel tree της Xilinx οπότε έχουμε τη δυνατότητα να τον επιλέξουμε κατά το configuration του kernel μέσα από τη ρύθμιση **Xilinx AXI DMA Driver**. Σε αυτό το σημείο οφείλουμε να σημειώσουμε μια ακόμη δυνατότητα που υπάρχει και αφορά στην εκτέλεση του driver από το user-space αντί για το kernel-space. Αυτή η μέθοδος ονομάζεται **UIO (User-space Input/Output)** και μπορεί να φανεί εξαιρετικά χρήσιμη σε ορισμένες περιπτώσεις. Μερικά από τα χαρακτηριστικά της είναι:

- Ένα, μοναδικό kernel module γράφεται μία φορά και χρησιμεύει ως ''γέφυρα'' μεταξύ του driver και του kernel. Είναι εύκολη η συγγραφή και η συντήρηση του
- Το βασικό τμήμα του driver αναπτύσσεται στο user-space έχοντας στη διάθεσή μας όλα τα εργαλεία και τις βιβλιοθήκες που ήδη γνωρίζουμε
- Πιθανά bugs στον driver δε θα προκαλέσουν ανεπανόρθωτα σφάλματα του πυρήνα (Kernel panics)
- Οποιαδήποτε αναβάθμιση του driver δεν απαιτεί recompile του kernel*

Για την ενεργοποίηση του UIO θα πρέπει να γίνει κατάλληλο configuration/compile του kernel όπως φαίνεται στην εικόνα 4.2

Στον αντίποδα, η χρήση της ''παραδοσιακής'' μεθόδου, αυτής δηλαδή του kernel-space driver παρέχει άλλα σημαντικά πλεονεκτήματα, όπως η προστασία από ταυτόχρονες προσβάσεις, απρόσκοπτη χρησιμοποίηση των interrupts χωρίς αναστολή της εκτέλεσης του user-space application, ενώ η βασική διαφορά είναι ότι μπορεί να λειτουργήσει ακόμα και με εικονικές διευθύνσεις, όταν η φυσική διεύθυνση του περιφερειακού είναι άγνωστη. Λόγω μεγαλύτερης ευκολίας (αφού ήδη συμπεριλαμβανόταν στον kernel) και αξιοπιστίας προτιμήσαμε τη δεύτερη μέθοδο, παρόλο που

*Το recompile του kernel είναι απαραίτητο μόνο αν ο driver είναι ενσωματωμένος σε αυτόν. Αν υφίσταται ως module τότε απαιτείται recompile μόνο του driver module.




**Σχήμα 4.2**: Ενεργοποίηση του UIO στο menuconfig του kernel

η πρώτη δοκιμάστηκε με μικρότερα περιφερειακά και θα μπορούσε να αποδειχθεί εξαιρετικά γρήγορη. Εντούτοις, επειδή οι περισσότεροι user-space AXI DMA drivers είναι γραμμένοι από τρίτους, δεν υποστηρίζουν ικανοποιητικά (ή και καθόλου) το μηχανισμό scatter-gather, οπότε κρίθηκε ακατάλληλη η χρήση τους.

Ο AXI DMA driver αποτελείται από περίπου 2.500 γραμμές κώδικα C και είναι διαθέσιμος στο kernel tree της Xilinx.

## 4.3 User-space Application

Προκειμένου να ελεγχθεί η αποτελεσματικότητα του ενσωματωμένου συστήματος και να έχουμε μια εκτίμηση των δυνατοτήτων του, δημιουργήσαμε μια εφαρμογή με την οποία στέλνουμε προκατασκευασμένα δεδομένα, τα αποτελέσματα της επεξεργασίας των οποίων γνωρίζουμε εξ' αρχής. Τα δεδομένα αυτά αναπαριστούν τα bytes από candidate και reference blocks και δεν απεικονίζουν κάποια πραγματική εικόνα. Η αρχική εφαρμογή γράφτηκε σε C χρησιμοποιώντας τις δυνατότητες της συνάρτησης mmap, με την οποία δεσμεύουμε αυθαίρετα περιοχές μνήμης. Ανάλογα με τις περιοχές μνήμης που δεσμεύουμε (και στις οποίες έχουν χαρτογραφηθεί οι registers

του περιφερειακού μας, είτε της μηχανής DMA είτε του επιταχυντή), εκτελείται και διαφορετική λειτουργία σύμφωνα με την αλληλουχία εντολών που καθορίζεται από το πρότυπο AXI Stream. Παρακάτω εμφανίζονται δύο χαρακτηριστικά τμήματα του κώδικα της εφαρμογής.

**Τμήμα κώδικα 4.2**: Ορισμός σταθερών (διευθυνσεις μνήμης περιφερειακού, αναγνωσης δεδομένων κλπ)

```
#define AXI_DMA_REGISTER_LOCATION  0x40400000 //AXI DMA Register Address Map
#define FSME_DUMMY_IP_BASE_ADDR  0x43C00000 //FSME DUMMY peripheral
#define DESCRIPTOR_REGISTERS_SIZE 0xFFF
#define SG_DMA_DESCRIPTORS_WIDTH  0x1FFF
#define FSME_DUMMY_MEM_HIGH_ADDR 0xFFFF
#define MEMBLOCK_WIDTH 0xFF //size of mem used by s2mm and mm2s
#define BUFFER_BLOCK_WIDTH 0xFF //size of memory block per descriptor in bytes
#define NUM_OF_DESCRIPTORS 0x1 //number of descriptors for each direction
#define HPo_DMA_BUFFER_MEM_ADDRESS 0x40000000
#define HPo_MM2S_DMA_BASE_MEM_ADDRESS  (HPo_DMA_BUFFER_MEM_ADDRESS)
#define HPo_S2MM_DMA_BASE_MEM_ADDRESS  (HPo_DMA_BUFFER_MEM_ADDRESS)
#define HPo_MM2S_DMA_DESCRIPTORS_ADDRESS  (HPo_MM2S_DMA_BASE_MEM_ADDRESS)
#define HPo_S2MM_DMA_DESCRIPTORS_ADDRESS  (HPo_S2MM_DMA_BASE_MEM_ADDRESS + DESCRIPTOR_REGISTERS_SIZE + 1)
#define HPo_MM2S_SOURCE_MEM_ADDRESS (0x00A00000)
#define HPo_S2MM_TARGET_MEM_ADDRESS (0x00B00000)
```

**Τμήμα κώδικα 4.3**: Εκχώρηση μνήμης στα περιφερειακά του συστήματος

```
axi_dma_register_mmap = mmap(NULL, DESCRIPTOR_REGISTERS_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dh,
    AXI_DMA_REGISTER_LOCATION);
mm2s_descriptor_register_mmap = mmap(NULL, DESCRIPTOR_REGISTERS_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dh,
    HPo_MM2S_DMA_DESCRIPTORS_ADDRESS);
s2mm_descriptor_register_mmap = mmap(NULL, DESCRIPTOR_REGISTERS_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, dh,
    HPo_S2MM_DMA_DESCRIPTORS_ADDRESS);
source_mem_map = mmap(NULL, BUFFER_BLOCK_WIDTH * NUM_OF_DESCRIPTORS, PROT_READ | PROT_WRITE, MAP_SHARED, dh, (off_t
    )(HPo_MM2S_SOURCE_MEM_ADDRESS));
dest_mem_map = mmap(NULL, BUFFER_BLOCK_WIDTH * NUM_OF_DESCRIPTORS, PROT_READ | PROT_WRITE, MAP_SHARED, dh, (off_t)(
    HPo_S2MM_TARGET_MEM_ADDRESS));
fsme_dummy_mmap = mmap(NULL, FSME_DUMMY_MEM_HIGH_ADDR, PROT_READ | PROT_WRITE, MAP_SHARED, dh,
    FSME_DUMMY_IP_BASE_ADDR);
```

Φυσικά, το πρώτο στάδιο πριν την εφαρμογή C ήταν ο έλεγχος τόσο του περιφερειακού όσο και των διεπαφών AXI ώστε να διαπιστωθεί η ικανοποιητική λειτουργία του συστήματος και να αποφευχθούν "εκπλήξεις" και απροσδόκητα λάθη κατά τη διάρκεια των δοκιμών. Έτσι σε πρώτο στάδιο δημιουργήσαμε πρώτα ένα TCL script με το οποίο αρχικοποιούσαμε χειροκίνητα τον επεξεργαστή και τις αντίστοιχες θέσεις μνήμης για κάθε register της μηχανής DMA αλλά και του FSME, ενώ στη συνέχεια επίσης χειροκίνητα διαβάζαμε τα αποτελέσματα από τις θέσεις μνήμης που αυτά αποθηκεύονταν. Ενδεικτικά παρουσιάζουμε ένα τμήμα του κώδικα.

**Τμήμα κώδικα 4.4**: Τμήμα του δοκιμαστικού script

```
####################################
#
# Initialize DRAM
#
####################################
# initialize memory which is read by mm2s
# WARNING ! If the transferSize is a big number,
# this code takes a long while to be executed in XMD.

for {set i 0} {$i < $transferWords} {incr i 1} {
        mwr [expr 0x00a00000+$i*4] [expr 0xFFFFFFF5+$i]
        #mwr 0x00a00004 0xa0000001
        #mwr 0x00a00008 0xa0000002
        #mwr 0x00a0000c 0xa0000003
        #mwr 0x00a00010 0xa0000004
        # .       .                   .
        # .       .                   .
        # .       .                   .
        # .       .                   .
}

# initializing memory which is being written to by s2mm
for {set i 0} {$i < $transferWords} {incr i 1} {
        mwr [expr 0x00b00000+$i*4] 0xbbbbbbbb
}

####################################
#
# start S2MM
#
####################################
# for s2mm write the current descriptor pointer
mwr 0x4040003c 0x00000000
mwr 0x40400038 0x40001000

# start s2mm engine
mwr 0x40400030 0x0101dfe3

# for s2mm write tail descriptor pointer
mwr 0x40400044 0x00000000
mwr 0x40400040 0x40001000
```

# 5
# Συμπεράσματα

Μετά την ολοκλήρωση του σχεδιασμού και των δοκιμών προχωρήσαμε σε αξιολόγηση των δεδομένων της σύνθεσης και προσομοίωσης του κυκλώματος ωστε να εξάγουμε ορισμένα συμπεράσματα σχετικά με την ταχύτητα λειτουργίας, την κατανάλωση ενέργειας, στατιστικά σχετικά με την πλατφόρμα υλοποίησης καθώς και την πιθανή δυνατότητα μελλοντικών επεκτάσεων ή/και αναβαθμίσεων. Καταρχάς, σε ότι αφορά τη μέγιστη συχνότητα λειτουργίας του κυκλώματος του επιταχυντή, αυτή αποδείχθηκε πως είναι τα τα 111,865 MHz (πρακτικά 112 MHz). Για μεγαλύτερες συχνότητες λειτουργίας ο logic synthesizer μας ενημέρωνε ότι υπήρχαν setup time violations για ορισμένα από τα στοιχεία μνήμης (flip-flop, registers) με αποτέλεσμα να μην είναι εγγυημένη η ομαλή και αξιόπιστη λειτουργία του συστήματος. Τα constraints που τέθηκαν ικανοποιούνταν λαμβάνοντας υπόψην και το chip στο οποίο καλούμασταν να υλοποιήσουμε το σύστημα, το ZYNQ Z-7020 της Xilinx. Πιθανότατα σε άλλα μεγαλύτερα chip οι δυνατότητες να είναι ακόμη μεγαλύτερες. Εκτός των παραπάνω, ο επιταχυντής κατέλαβε το 26% του συνολικού μεγέθους της επαναπρογραμματιζόμενης λογικής, ένα σχετικά μικρό ποσοστό, που μπορεί να μειωθεί ακόμα περισσότερο με περαιτέρω optimization του σχεδιασμού του κυκλώματος. Θεωρητικά θα μπορούσαν να βρίσκονται μέχρι και τέσσερις επιταχυντές παρόμοιου μεγέθους στο ίδιο chip, εκτελώντας διαφορετικές λειτουργίες, στη διάθεση του επεξεργαστή όποτε προκύπτουν συγκεκριμένες ανάγκες επεξεργασίας.

Στους παρακάτω πίνακες παρατίθενται λεπτομέρειες και στατιστικά βάσει των

οποίων έγινε μια πρώτη αξιολόγηση του σχεδιασμού. Οι μετρικές αναφέρονται στην έκταση της επαναπρογραμματιζόμενης λογικής που καταλήφθηκε (Πίνακας III), την κατανάλωση ενέργειας του ενσωματωμένου συστήματος (Πίνακες I & II) και τα αποτελέσματα του χρονισμού. Η σύνθεση και η προσομοίωση στόχευαν το chip xc7z020clg484-1.

**Πίνακας 5.1**: Xilinx Vivado Power Report

| Total On-Chip Power (W) | 1.735 |
|---|---|
| Dynamic (W) | 1.576 |
| Device Static (W) | 0.159 |
| Effective TJA (C/W) | 11.5 |
| Max Ambient (C) | 65.0 |
| Junction Temperature (C) | 45.0 |
| Thermal Margin (C) | 39.7 |

Στον Πίνακα II βρίσκονται οι πληροφορίες σχετικά με την κατανάλωση ενέργειας των επι μέρους υποσυστημάτων. Όπως ήταν αναμενόμενο, η περισσότερη ισχύς καταναλώνεται από τον ίδιο τον επεξεργαστή.

**Πίνακας 5.2**: Power Metrics Per Component

| **On-Chip** | **Power (W)** |
|---|---|
| Clocks | 0.03 |
| Signals | 0.005 |
| Slice Logic | 0.004 |
| PS7 | 1.532 |
| Static Power | 0.159 |
| Total | 1.735 |

**Πίνακας 5.3**: FPGA Area Utilization Report

| **Site Type** | **Used** | **Available** | **Utilization %** |
|---|---|---|---|
| Slice LUTs | 13912 | 53200 | 26.15 |
| LUT as Logic | 12041 | 53200 | 22.63 |
| LUT as Memory | 1871 | 17400 | 10.75 |
| LUT FF Pairs | 17141 | 53200 | 32.22 |
| Slice Registers | 11525 | 106400 | 10.83 |
| Block RAM Tiles | 3 | 140 | 2.14 |
| F7 Muxes | 232 | 26600 | 0.87 |
| F8 Muxes | 88 | 13300 | 0.66 |

Η υλοποίησή μας σύμφωνα με τα παραπάνω στοιχεία μοιάζει συγκρίσιμη με αυτή

που προτάθηκε στο [7] με μόνη διαφορά τα 900 παραπάνω LUTs που χρησιμοποιήθηκαν στη δική μας υλοποίηση. Φυσικά ένα τέτοιο σχετικά μικρό νούμερο μπορεί να οφείλεται στην υλοποίηση πολλών επιπλέον συστημάτων όπως οι DMA και AXI controllers.

Ο σχεδιασμός και η υλοποίηση αυτού του ενσωματωμένου συστήματος έδειξε τις δυνατότητες που υπάρχουν στη δημιουργία νέων συστημάτων με παράλληλο σχεδιασμό του υλικού και του λογισμικού. Επίσης ήταν μια επίδειξη για το πώς μπορούμε να χρησιμοποιήσουμε τα νέα SoC προκειμένου να κατασκευάσουμε ακόμα καλύτερα ψηφιακά συστήματα. Είναι φυσικά προφανές ότι οι δυνατότητες που υπάρχουν είναι τεράστιες, καθώς οι πλατφόρμες αυτές μπορούν να χρησιμοποιηθούν για τη δημιουργία επιταχυντών για πολλές άλλες εφαρμογές, που παραδοσιακά υλοποιούνται σε επίπεδο λογισμικού, με μεγάλα οφέλη στην ταχύτητα.

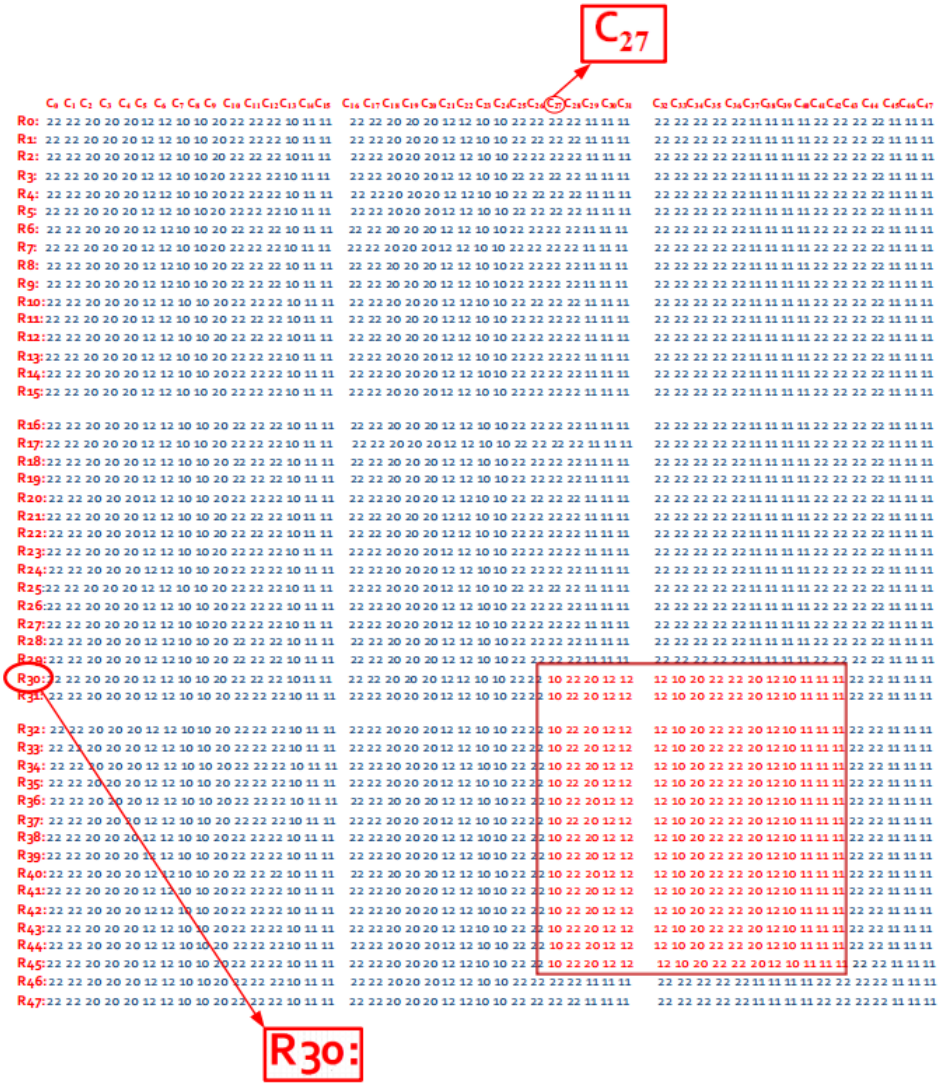
## 5.1 Επικυρωση Ορθης Λειτουργιας

Αν και σε επόμενη ενότητα θα αναφερθούμε εκτεταμένα στις μεθόδους που χρησιμοποιήσαμε για να ελέγξουμε την ορθή λειτουργία του κυκλώματος, στο σημείο αυτό αξίζει να αναφερθούμε στο test και τα δεδομένα που χρησιμοποιήσαμε προκειμένου να ελεγξουμε ότι πράγματι ο επιταχυντής λειτουργεί. Πρακτικά, η μέθοδος αποσκοπούσε στο να δημιουργήσουμε μια τυχαία περιοχή αναζήτησης 32 x 32 pixel και μέσα σε αυτήν να περιλαμβάνεται το τμήμα εκείνο το οποίο θα αποτελούσε το current (candidate) block, ωστε κατά τη διαδικασία της εξαντλητικής αναζήτησης να προκύψει μηδενικό SAD. Στο σχήμα 5.1 μπορούμε να δούμε τα δεδομένα με τα οποία ελέγξαμε τη λειτουργία του επιταχυντή, τη search area και το τμήμα της που ταυτίζεται με το CB.

Κατά την εκτέλεση του ελέγχου, το αποτέλεσμα που έπρεπε να πάρουμε ισοδυναμούσε με τον κύκλο ρολογιού (position) κατά τον οποίο εντοπίστηκε το ελάχιστο SAD. Στο σχήμα 5.2 φαίνεται το αποτέλεσμα της προσομοίωσης.

### 5.1.1 Επιταχυνση

Η αξιολόγηση των επιδόσεων του επιταχυντή έγινε κατα βάση συγκρίνοντας την ταχύτητα εκτέλεσης μιας υλοποίησης του ίδιου αλγορίθμου σε software σε σχέση με την ταχύτητα εκτέλεσης του αλγορίθμου στο hardware με συχνότητα λειτουργίας τα 111,865 MHz, που αναφέραμε παραπάνω. Τα δεδομένα εισόδου ήταν και




**Σχήμα 5.1**: Η περιοχή αναζήτησης με σημειωμένο το κομμάτι στο οποίο υπάρχει απόλυτη ταύτιση με το CB

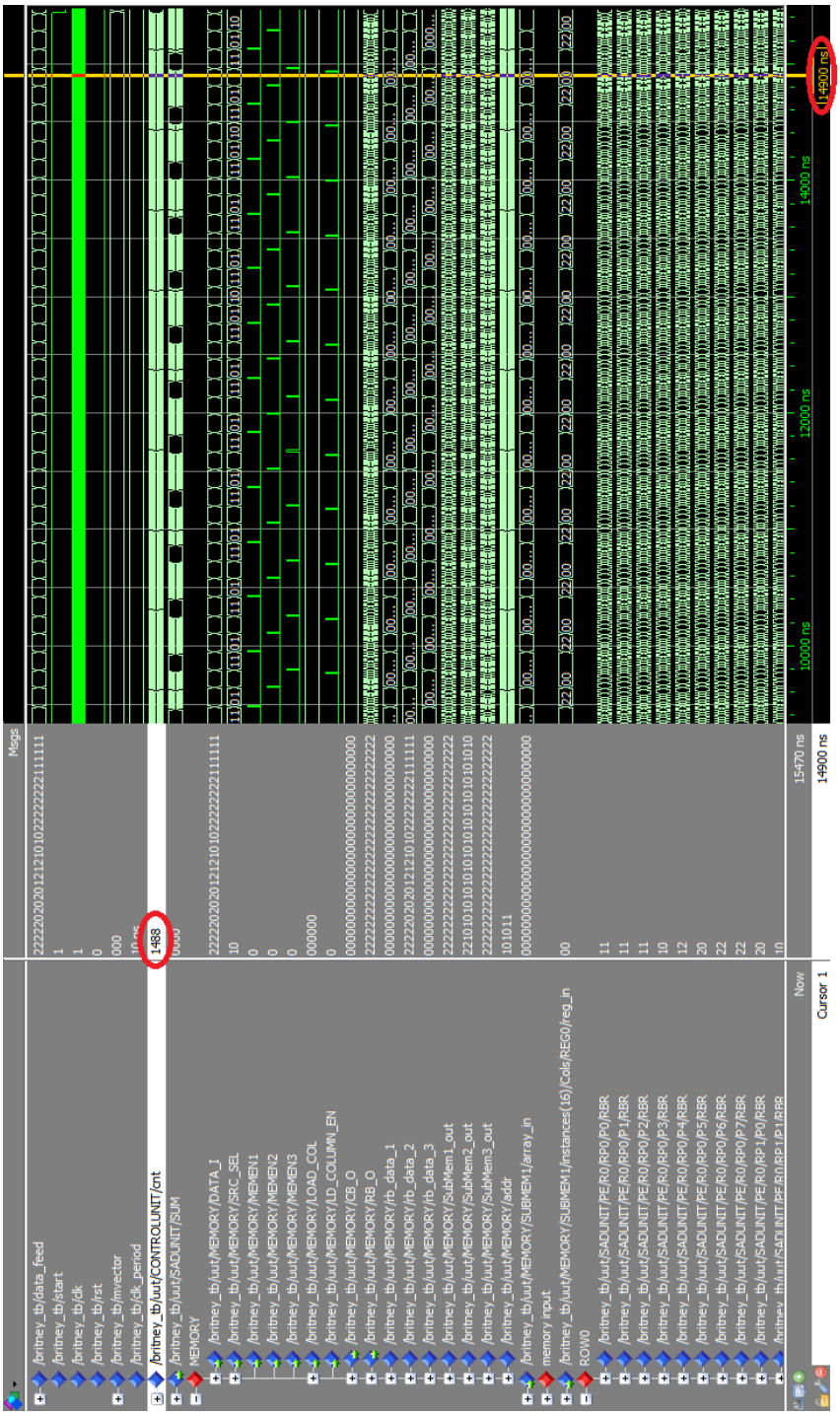
στις δύο περιπτώσεις ακριβώς τα ίδια. Το πρόγραμμα που υλοποιεί τον αλγόριθμο FSME γράφτηκε σε C. Ο λόγος είναι ότι η C μας δίνει τη δυνατότητα να είμαστε όσο πιο "κοντά" γίνεται στο επίπεδο του υλικού, χρησιμοποιώντας ωστόσο ρουτίνες "υψηλού επιπέδου" που κάνουν πιο εύκολη τη συγγραφή κώδικα. Επιπλέον, ο μεταγλωττιστής gcc (GNU C Compiler) μας δίνει τη δυνατότητα να επιλέξουμε το επίπεδο βελτιστοποίησης που θέλουμε, οπότε μπορεί να παράγει αυτόματα βελτι-

στοποιημένα εκτελέσιμα. Στο σχήμα 5.3 φαίνεται ο χρόνος εκτέλεσης του αλγορίθμου για διαφορετικά επίπεδα βελτιστοποίησης, σε σύγκριση με το χρόνο εκτέλεσης στο hardware.

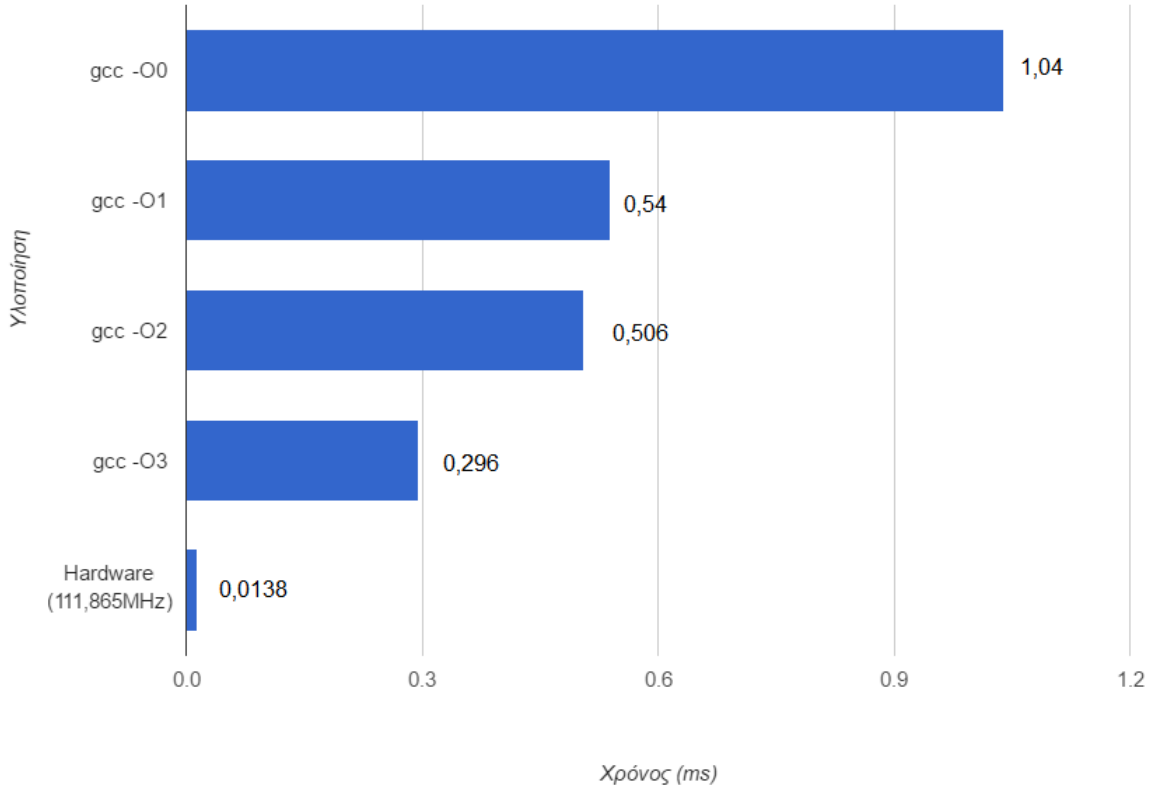
Είναι εμφανές ότι η βέλτιστη υλοποίηση σε software απέχει πολύ από την υλοποίηση σε hardware, με την τελευταία να είναι σχεδόν 21 φορές ταχύτερη! Αξίζει βέβαια να σημειώσουμε πως στο σημείο αυτό αξιολογούμε την ταχύτητα εκτέλεσης του αλγορίθμου και μόνο αυτή. Αυτό σημαίνει ότι τυχόν καθυστερήσεις και bottlenecks που προκύπτουν από άλλους παράγοντες όπως οι προσπελάσεις μνήμης, η ταχύτητα της διαμεταγωγής δεδομένων κ.λπ. δεν λαμβάνονται υπόψιν. Ωστόσο, είναι εύκολο να συμπεράνουμε ότι οποιαδήποτε επιπλέον καθυστέρηση υφίσταται, εύκολα αντισταθμίζεται από την πολύ μεγάλη ταχύτητα του περιφερειακού.

## 5.2 Μελλοντικες Επεκτασεις

Ο επιταχυντής που περιγράφηκε στην παρούσα διπλωματική εργασία είναι μια πρώτη βάση πάνω στην οποία μπορούν να στηριχθούν πολλά νέα και ενδιαφέροντα project. Βασική μας επιδίωξη είναι η δημιουργία ενός μεγαλύτερου συστήματος που θα είναι σε θέση να επεξεργαστεί σε πραγματικό χρόνο ροή βίντεο σε ποιότητα 4K με ρυθμό 30 καρέ/δευτερόλεπτο. Αυτό κρίνεται προς το παρόν εξαιρετικά δύσκολο, όμως υπάρχουν μέθοδοι που μπορούμε να εκμεταλλευτούμε για να ερευνήσουμε τις δυνατότητες που υπάρχουν. Αυτές οι μέθοδοι περιλαμβάνουν την υλοποίηση ενός πιο έξυπνου συστήματος επαναχρησιμοποίησης δεδομένων, σε δύο νέα επίπεδα C και D, καθώς επίσης και την υλοποίηση κάποιου υποτυπώδους pipelining ωστε να επιτευχθεί κάποιος βαθμός παραλληλίας.




**Σχήμα 5.2**: Οι κυματομορφές της προσομοίωσης δείχνουν τον κύκλο εκτέλεσης όπου εντοπίστηκε η ελάχιστη τιμή SAD




**Σχήμα 5.3**: Χρόνος εκτέλεσης (Runtime) διαφορετικών υλοποιήσεων

# Παραρτήματα

# A

# Εργαλεία και Μεθοδολογία Σχεδιασμού

Για την εκπόνηση της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκε πληθώρα εργαλείων και μεθοδολογιών τόσο κατά τη φάση του σχεδιασμού όσο και κατά τη φάση της υλοποίησης του συστήματος. Τα εργαλεία που χρησιμοποιήθηκαν περιλάμβαναν τόσο λογισμικό (γλώσσες προγραμματισμού, εφαρμογές προσομοίωσης και σύνθεσης κ.λπ.) όσο και υλικό (hardware) όπως demonstration boards (πλακέτες επίδειξης και αξιολόγησης), communication interfaces και λογικούς αναλυτές (logic analyzers). Στην ενότητα αυτή παρουσιάζονται λεπτομερώς τα εργαλεία και οι μεθοδολογία που χρησιμοποιήθηκαν για να επιτευχθεί το τελικό αποτέλεσμα.

## A.1 Μεθοδολογια Σχεδιασμου

Για την ανάπτυξη του ενσωματωμένου συστήματος ακολουθήσαμε μια συγκεκριμένη μεθοδολογία η οποία μας επέτρεψε να δουλεψουμε πάνω σε διακριτά τμήματα όπως το υλικό και το λογισμικό, ενώ ταυτόχρονα μείωνε στο ελάχιστο δυνατό τα πιθανά σφάλματα. Αρχικά, έπειτα από έρευνα αποφασίσαμε την αρχιτεκτονική του επιταχυντή που θα υλοποιούσαμε, ενώ δώσαμε ιδιαίτερη έμφαση στο να σχεδιάσουμε κάθε σύστημα με τέτοιο τρόπο ώστε να διασφαλίσουμε από την αρχή τη σωστή λειτουργία, στα πλαίσια των δυνατοτήτων μας. Σχεδιάσαμε κάθε κομμάτι του επιταχυντή ξεχωριστά, σε VHDL ενώ ταυτόχρονα ελέγχαμε τη σωστή λειτουργία προσομοιώνοντας το στο ModelSim. Περισσότερα για τη διαδικασία της προσομοί-

ωσης και τις εφαρμογές θα αναφέρουμε παρακάτω. Όπως ήταν φυσικό, η απευθείας προσομοίωση ήταν εφικτή για μικρά τμήματα όπως καταχωρητές, μετρητές, κ.λπ. Για τα μεγαλύτερα κυκλώματα του επιταχυντή ακολουθήσαμε τη λογική των testbenches. Αφού ολοκληρώθηκε η σύνθεση και ο έλεγχος των επιμέρους τμημάτων, ακολούθησε η μεταξύ τους σύνδεση και ο έλεγχος σε επίπεδο συστήματος, πάλι με ένα συνολικό tesbench.

Το επόμενο στάδιο περιλάμβανε την ενσωμάτωση του επιταχυντή στο οικοσύστημα του SoC Zynq-7000. Όπως αναφέραμε και σε προηγούμενη ενότητα, κάθε περιφερειακό που πρόκειται να ενσωματωθεί στο συγκεκριμένο SoC οφείλει να επικοινωνεί πάνω από το πρωτόκολλο AXI. Ταυτόχρονα με αυτό το γεγονός, λήφθηκε ορισμένες ακόμα αποφάσεις που αφορούσαν την αρχιτεκτονική του όλου συστήματος όπως, η μέθοδος διαμεταγωγής δεδομένων, το εύρος των διαύλων διασύνδεσης και η έκδοση του AXI που θα χρησιμοποιούνταν. Έτσι καταλήξαμε στην υλοποίηση του AXI4-Stream interface για τη διαμεταγωγή δεδομένων ενώ στη χρήση ενός μικρότερου, AXI4-Lite interface, για τη μετάδοση των σημάτων ελέγχου. Παρόλες τις διευκολύνσεις που μας παρείχε η πλατφόρμα Vivado της Xilinx, τελικά αποδείχθηκε απλούστερο (και ταυτόχρονα ουσιαστικότερο από πλευράς γνώσης) να υλοποιήσουμε από το μηδέν τα δικά μας AXI4 interfaces αξιοποιώντας το υλικό και τα datasheet τόσο της ARM όσο και της Xilinx. Με τον τρόπο αυτό, είχαμε πολύ καλύτερη εποπτεία του σχεδιασμού και μπορούσαμε να εντοπίσουμε ευκολότερα τα περισσότερα σφάλματα.

Όπως και στα υπόλοιπα τμήματα του κυκλώματος, έτσι και στα AXI4 interfaces προχωρήσαμε σε εκτεταμένες προσομοιώσεις προκειμένου να βεβαιωθούμε ότι λειτουργούν σύμφωνα με τους κανόνες που διέπουν το πρωτόκολλο. Κατόπιν τούτου, δημιουργήσαμε ένα "περιτύλιγμα" (wrapper) για το ίδιο το κύκλωμα του επιταχυντή και τα AXI4 interfaces που τελικά αποτέλεσε και την τελική μορφή του περιφερειακού (IP) που χρησιμοποιήθηκε στην υπόλοιπη διαδικασία του σχεδιασμού. Στο σημείο αυτό ξεκινήσαμε τη δόμηση του ενσωματωμένου συστήματος block-προς-block στο περιβάλλον του Vivado. Κατά τη φάση αυτή σχεδιάστηκε το ανώτερο στάδιο της αρχιτεκτονικής του συστήματος που αποτελούνταν από μεγάλα διακριτά τμήματα όπως επεξεργαστή, μνήμη, μηχανή DMA, μονάδες διασύνδεσης περιφερειακών, Block-RAMs κ.λπ. Εν πολλοίς η μορφή που πήρε το σύστημα σε αυτό το στάδιο είχε προαποφασιστεί από αρχιτεκτονικές επιλογές που κάναμε στα προηγούμενα.

Με το πέρας του σχεδιασμού σε αυτό το επίπεδο, έπρεπε πλέον να ελέγξουμε

τη σωστή λειτουργία του συστήματος ως σύνολο. Για το λόγο αυτό αποφασιστηκε πρώτα η χρησιμοποίηση ενός απλού περιφερειακού στη θέση του επιταχυντή, με γνωστά και προαποφασισμένα αποτελέσματα το οποίο ωστόσο θα διέθετε ακριβώς τις εισόδους και εξόδους του πραγματικού επιταχυντή. Με τον τρόπο αυτό ήταν πολύ εύκολο να διαπιστώσουμε αν το περιφερειακό ανταποκρίνεται κατα τον προσδοκώμενο τρόπο στα σήματα ελέγχου και συγχρονισμού, ενώ ταυτόχρονα θα ήμασταν σίγουροι για την αξιοπιστία των αποτελεσμάτων.

Για τον έλεγχο του συστήματος επιστρατεύτηκαν δύο εργαλεία: το XMD (Xilinx Microprocessor Debugger) και η γλώσσα TCL. Με τη χρήση ενός TCL script το οποίο έδινε οδηγίες μία-προς-μία για την αρχικοποίηση του συστήματος και τα σήματα που έπρεπε να ενεργοποιηθούν μπορούσαμε να ελέγξουμε πότε και σε ποια χρονική στιγμή γινόταν η ενεργοποίηση του επεξεργαστή, της μηχανής DMA και του περιφερειακού μας, ενώ με τον XMD βλέπαμε σε πραγματικό χρόνο τα αποτελέσματα που παράγονταν. Στο σημείο αυτο αξίζει να αναφερθούμε σε ένα ακόμα τμήμα αυτού του ελέγχου που ήταν η χρηση των ILA cores και του λογικού αναλυτή. Όπως ήταν αναμενόμενο, η λειτουργία του συστήματος δεν ήταν εξ'αρχής ικανοποιητική, ούτε φυσικά ανευ σφαλμάτων και απροσδόκητων συμπεριφορών. Προκειμένου να έχουμε τη μέγιστη εποπτεία, πέρα από των έλεγχο του software (πρακτικά των οδηγιών που παρείχαμε με το TCL script) έπρεπε να είμαστε σε θέση να βλέπουμε τα σήματα που οδηγούνταν σε επίπεδο hardware σε πραγματικό χρόνο, ωστε να διαπιστώσουμε αν υπάρχει οποιοδήποτε ζήτημα σε επίπεδο υλοποίησης του κυκλώματος. Τόσο τα ILA cores όσο και ο λογικός αναλυτής φάνηκαν εξαιρετικά χρήσιμα εργαλεία για τη διόρθωση αρκετών σφαλμάτων και την πρόληψη πολλών περισσότερων. Περαιτέρω πληροφορίες θα δωθούν σε επόμενη ενότητα, όπου θα αναλυθεί η χρησιμοτητα και η λειτουργία τους.

Με το τέλος αυτής της φάσης των δοκιμών, συνδέσαμε τελικά το περιφερειακό του επιταχυντή και επαναλάβαμε τις δοκιμές για να σιγουρευτούμε ότι όλα λειτουργούν όπως αναμενόταν. Μετά από ρυθμίσεις και παραμετροποιήσεις κυρίως στο κύκλωμα του controller του επιταχυντή και της FSM του, καταλήξαμε σε επιβεβαίωση της σωστής λειτουργίας του συστήματος χρησιμοποιώντας απευθείας εντολές "τύπου" assembly (πρακτικά δεν επρόκειτο για assembly, αλλά για εντολές του XMD που ενεργούσαν απευθείας πάνω στον επεξεργαστή και τη μνήμη του συστήματος). Σειρά είχε πλέον η εγκατάσταση μιας διανομής Linux με πλήρη αναγνώριση και χαρτογράφηση του συστήματος, καθώς και η δημιουργία μιας εφαρμογής χρήστη

(user-space application) η οποία θα δεχόταν δεδομένα και θα παρήγαγε τα απαιτούμενα αποτελέσματα.

## Α.2 Λογισμικο που Χρησιμοποιηθηκε

Σε όλη τη διάρκεια του σχεδιασμού και της υλοποίησης του συστήματος χρησιμοποιήσαμε πληθώρα λογισμικού. Στο σημείο αυτό θα κάνουμε μια επιγραμματική αναφορά πάνω στα εργαλεία και τις ιδιαίτερες δυνατότητες τους που μας οδήγησαν στο να τα επιλέξουμε και να δουλέψουμε με αυτά.

### Α.2.1 VHDL

Η VHDL είναι μια γλώσσα περιγραφής υλικού (Hardware Description Language). Περιγράφει τη συμπεριφορά ενός ηλεκτρονικού κυκλώματος ή συστήματος, με βάση την οποία μπορεί στη συνέχεια να υλοποιηθεί το κύκλωμα ή το σύστημα. Το ακρωνύμιο VHDL προκύπτει από τα αρχικά γράμματα της φράσης VHSIC Hardware Description Language. Ο όρος VHSIC είναι με τη σειρά του αρκτικόλεξο της φράσης Very High Speed Integrated Circuits, μια πρωτοβουλία που χρηματοδοτήθηκε από το Υπουργείο Άμυνας των Η.Π.Α. κατά τη δεκαετία του '80 και η οποία είχε ως αποτέλεσμα τη δημιουργία της VHDL. Η πρώτη έκδοση της γλώσσας ήταν η VHDL-87 ενώ η δεύτερη η VHDL-93. Πρόκειται για την πρώτη γλώσσα περιγραφής υλικού που προτυποποιήθηκε από την IEEE μέσω του στάνταρ IEEE-1076, ενώ επεκτάθηκε με το στάνταρ IEEE-1164 το οποίο εισήγαγε ένα λογικό σύστημα πολλαπλών τιμών.

Η VHDL προορίζεται τόσο για τη σύνθεση όσο και την προσομοίωση κυκλωμάτων. Μολονότι κάθε συντακτικά σωστός κώδικάς VHDL μπορεί να προσομοιωθεί, δεν είναι όλες οι δομές συνθέσιμες.Το βασικό ωστόσο κίνητρο για τη χρήση της VHDL (ή του αντίπαλου δέους - της Verilog) είναι η προτυποποίηση της γλώσσας πράγμα που σημαίνει ότι είναι ανεξάρτητη τεχνολογίας και κατασκευαστή του εκάστοτε κυκλώματος. Αυτό σημαίνει ότι μπορεί να μεταφερθεί και να επαναχρησιμοποιηθεί για οποιαδήποτε τεχνολογία και κατασκευαστή. Οι δύο άμεσες εφαρμογές της VHDL είναι στον τομέα των Προγραμματιζόμενων Λογικών Στοιχείων όπως τα CPLD και οι FPGA καθώς επίσης και στον τομέα των ASIC. Στην πρώτη περίπτωση μπορούμε να προργαμματίσουμε μόνοι μας το λογικό στοιχείο και να το χρησιμοποιήσουμε άμεσα, ενώ στη δεύτερη θα πρέπει να στείλουμε τον κώδικα σε κάποιο εργοστάσιο προκειμένου να παράξει το chip.

### A.2.2 Xilinx Vivado Suite

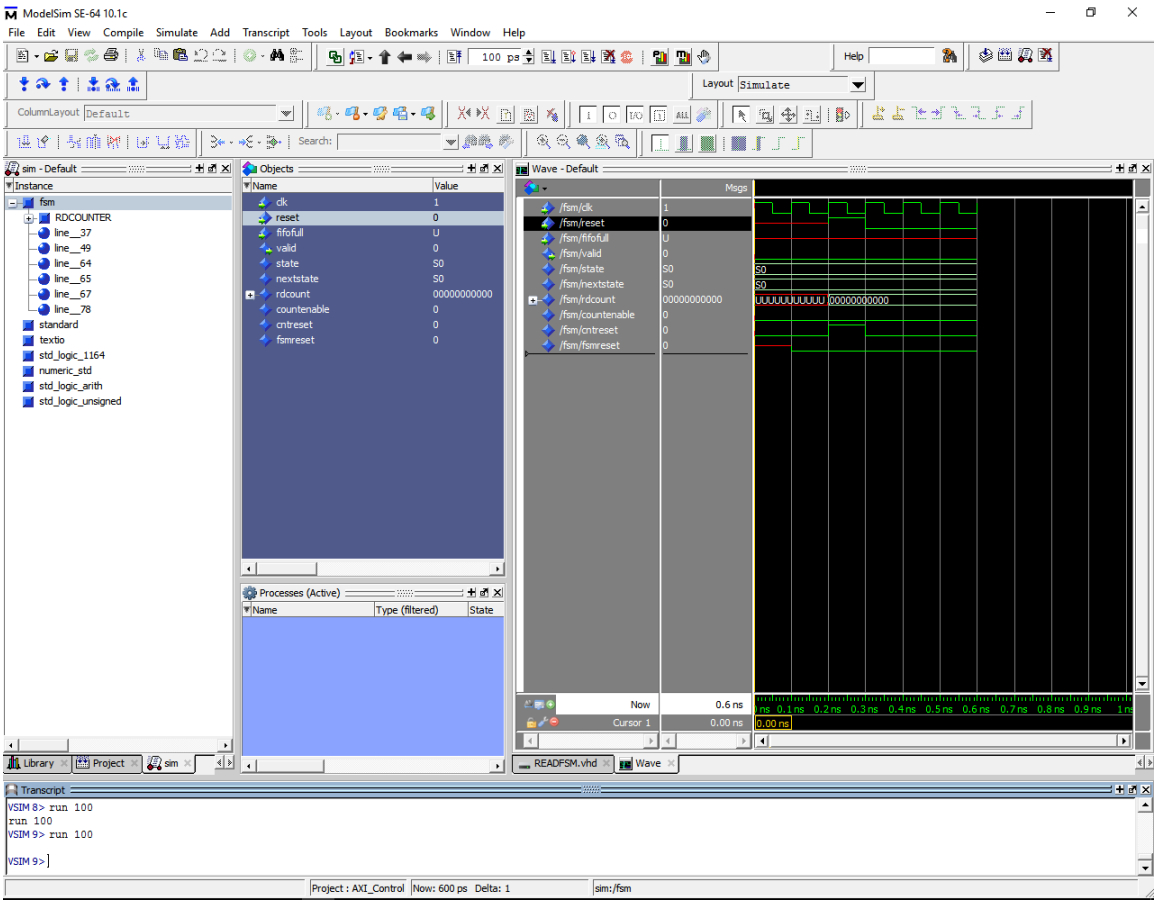
Η σουίτα Vivado της Xilinx αποτελεί τον αντικαταστάτη του παλαιότερου ISE και πρόκειται στην ουσία για μια ολοκληρωμένη πλατφόρμα σύνθεσης και ανάλυσης κυκλωμάτων σε HDL. Εκτός από "κλασικά" εργαλεία για τη σύνθεση, τον έλεγχο, το timing analysis, την παραγωγή διαγραμμάτων RTL κ.λπ., η σουϊτα του Vivado περιλαμβάνει τα εξής εργαλεία:

- Vivado HLS (High-Level Synthesis) - Το εργαλείο αυτό μας επιτρέπει να συνθέσουμε κυκλώματα προοριζόμενα για πλατφόρμες της Xilinx χρησιμοποιώντας μόνο κώδικα C, C++ ή SystemC. Με αυτό τον τρόπο επιταχύνεται μεγαλύτερη παραγωγικότητα, αφού δεν χρειάζεται να γραφεί "χειροκίνητα" RTL.
- Vivado Simulator - Πρόκειται για το νέο εργαλείο προσομοίωσης της Xilinx. Υποστηρίζει ετερογενείς υλοποιήσεις (γραμμένες σε πολλές διαφορετικές HDL), TCL scripts και κρυπτογραφημένα IP.
- Vivado IP Integrator - Επιτρέπει την ταχεία ενσωμάτωση και παραμετροποίηση έτοιμων IP από τη βιβλιοθήκη της Xilinx. Συνεργάζεται επίσης με το Simulink και το Matlab για την εισαγωγή υλοποιήσεων από αυτα.

Από το 2015 κι έπειτα η πλατφόρμα Xilinx Zynq-7000 υποστηρίζεται πλήρως από το Vivado, ενώ αναβαθμίζεται συνεχώς.

### A.2.3 ModelSim

Το ModelSim αποτελεί ενα πολύ ισχυρό εργαλείο για την προσομοίωση λογικών κυκλωμάτων, που δημιουργήθηκαν με κάποια γλώσσα περιγραφής υλικού. Δημιουργήθηκε από τη Mentor Graphics και αποτελεί ένα από τα standard προγράμματα που χρησιμοποιούνται τόσο στη βιομηχανία, όσο και σε ακαδημαϊκό επίπεδο. Το πρόγραμμα αποτελείται από τρία κυρίως κομμάτια, τον επεξεργαστή κειμένου, την κονσόλα και την προβολή των κυματομορφών (waveform viewer). Καθώς ήταν φυσικό, μετά τη σύνθεση του κώδικά προέκυπταν αρκετά προβλήματα, τα οποία καλούμασταν να λύσουμε και το ModelSim αποδείχθηκε πολύτιμος σύμμαχος. Επιπλέον, κατά τον έλεγχο ορθής λειτουργίας των επιμέρους modules του επιταχυντή, ήταν το βασικό εργαλείο με το οποίο προσομοιώναμε και αναπαριστούσαμε γραφικά τη λειτουργία τους. Η έκδοση του Modelsim που χρησιμοποιήσαμε ήταν η SE-64 10.1c.




**Σχήμα A.1**: Το περιβάλλον του Modelsim

### A.2.4 TCL

Η Tcl (αρχικά των αγγλικών λέξεων "Tool Command Language", "Γλώσσα Εντολών Εργαλείων", συνήθως εμφανίζεται ως "Tcl" ή "TCL") είναι μια γλώσσα προγραμματισμού σεναρίων που δημιουργήθηκε από τον John Ousterhout στο Πανεπιστήμιο Berkeley. Συχνά χρησιμοποιείται για γρήγορη ανάπτυξη εφαρμογών (rapid application development/rapid prototyping), εφαρμογές σεναρίων, γραφικές διεπαφές και δοκιμές λογισμικού. Η Tcl χρησιμοποιείται σε πλατφόρμες ενσωματωμένων συστημάτων, τόσο στην πλήρη της έκδοση, όσο και σε διάφορες άλλες εκδόσεις με μικρότερες απαιτήσεις. Ο συνδυασμός της Tcl με τη γραφική εργαλειοθήκη Tk ονομάζεται Tcl/Tk.

Η TCL είναι ευρέως διαδεδομένη σε πλατφόρμες ανάπτυξης λογικών κυκλωμάτων και ενσωματωμένων συστημάτων όπως το Quartus της Altera, οι σουϊτες ISE και Vivado της Xilinx, καθώς και προσομοιωτές όπως το Modelsim. Η βασική χρησιμότητα της TCL στις εφαρμογές αυτές είναι πρακτικά η παροχή ενός εναλλακτικού interface προκειμένου να αυξηθεί η παραγωγικότητα και η ταχύτητα σχεδιασμού μέσω της αυτοματοποίησης συγκεκριμένων διαδικασιών. Αυτή ακριβώς τη δυνατότητα χρησιμοποιήσαμε τόσο στο περιβάλλον του XMD όσο και στο περιβάλλον του Modelsim προκειμένου να επιταχύνουμε τις δοκιμές μας. Πιο συγκεκριμένα, δημιουργήσαμε ορισμένα TCL scripts τα οποία αναλάμβαναν συγκεκριμένες εργασίες παραμετροποίησης οι οποίες όταν γίνονταν χειροκίνητα ήταν ιδιαίτερα χρονοβόρες, ειδικά λόγω του πλήθους τους. Γίνεται εύκολα αντιληπτό λοιπόν, ότι σε ένα περιβάλλον που απαιτούνται συνεχείς δοκιμές μετά από κάθε νέα διόρθωση του κώδικα, η σπατάλη ακόμα και 30 δευτερολέπτων για επανάληψη των ίδιων ρυθμίσεων, τελικά οδηγεί σε σημαντική καθυστέρηση.

Γενικά δημιουργήθηκαν δύο TCL script, το ένα για την αυτοματοποίηση των ελέγχων στο περιβάλλον του XMD και το άλλο για την αυτοματοποίηση της παραμετροποίησης της προσομοίωσης στο περιβάλλον του Modelsim. Ειδικά στην δεύτερη περίπτωση, στα τελικά στάδια της προσομοίωσης, ήταν αδύνατον να ελέγξουμε, να ομαδοποιήσουμε και να αρχικοποιήσουμε τις τιμές μερικών εκατοντάδων σημάτων σε κάθε επανάληψη της προσομοίωσης, επομένως το συγκεκριμένο script ήταν ιδιαίτερα χρήσιμο.

**Τμήμα κώδικα A.1**: Τμήμα του TCL script, για την προσομοίωση της μονάδας υπολογισμού SAD (SADUNIT)

```
for {set j 0} {$j < 8} {incr j 1} {
                add wave -group ROW$i \
                sim:/britney_tb/uut/SADUNIT/PE/R$i/RP0/P$j/RBR;
                radix signal sim:/britney_tb/uut/SADUNIT/PE/R$i/RP0/
                    P$j/RBR -hex
        }
```

## Α.3 Υλικο που Χρησιμοποιηθηκε

### Α.3.1 Zynq-7000

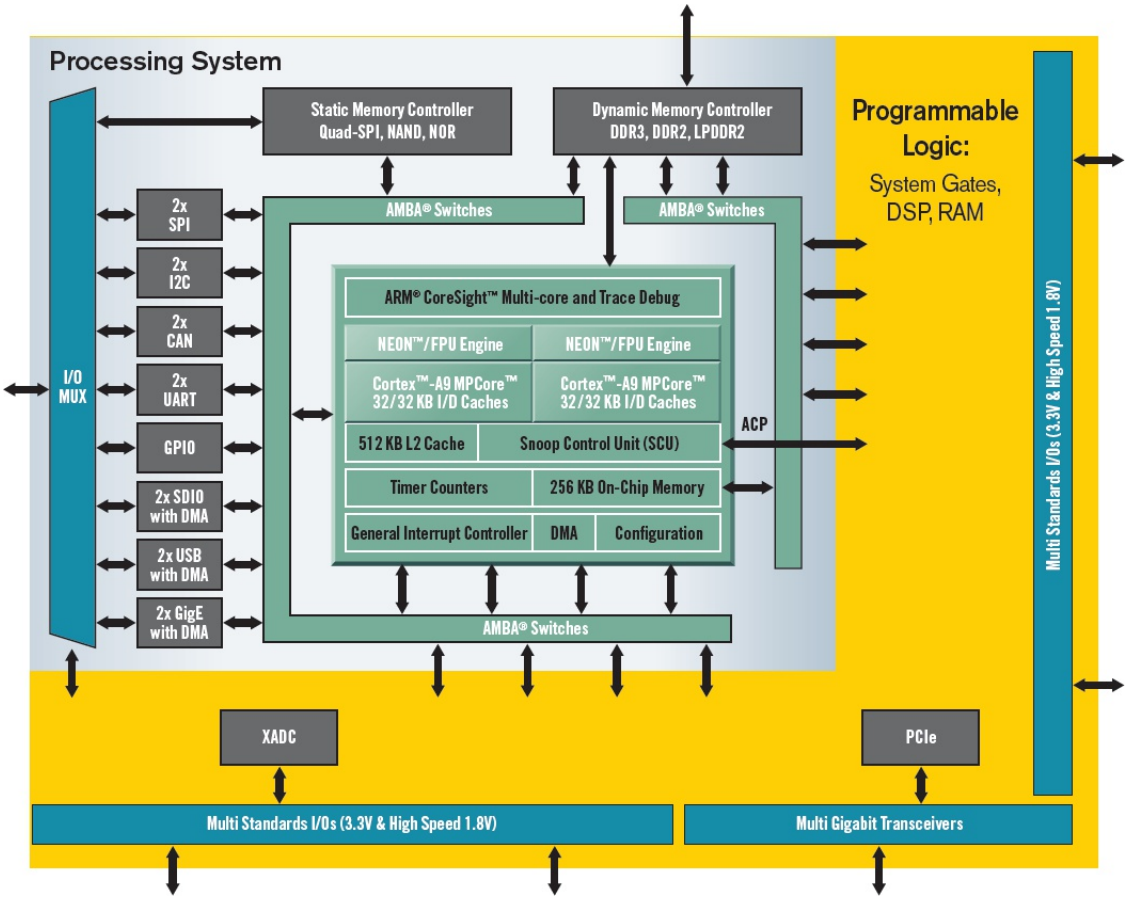
Η βασική πλατφόρμα υλοποίησης του συστήματος ήταν το System-on-Chip (SoC) Zynq-7000 της Xilinx. Πρόκειται για ένα System-on-Chip το οποίο περιλαμβάνει πολλά διαφορετικά components τα οποία μπορούν να συνεργαστούν, ώστε να δημιουργήσουν πιο πολύπλοκες εφαρμογές. Μεταξύ άλλων τα τσιπ της σειράς Zynq εμπεριέχουν έναν διπύρηνο επεξεργαστή ARM, επαναπρογραμματιζόμενη λογική FPGA, ελεγκτή μνήμης DDR3 και DSP slices για ψηφιακή επεξεργασία σημάτων. Τα πλεονεκτήματα μιας τέτοιας δομής είναι προφανή. Ένα σύστημα Zynq μπορεί τόσο να χρησιμοποιηθεί μεμονωμένα ως επεξεργαστής για εφαρμογές ARM όσο και σε συνδυασμό με την FPGA και τα υπόλοιπα στοιχεία, για την υλοποίηση συστημάτων προσαρμοσμένων σε συγκεκριμένες ανάγκες.

Ο κύριος λόγος για τον οποίο επιλέξαμε τη χρήση του συγκεκριμένου SoC και όχι κάποιας μεμονωμένης FPGA, είναι η ταχύτητα με την οποία θα ήμασταν σε θέση να υλοποιήσουμε ένα πλήρως λειτουργικό ενσωματωμένο σύστημα, χωρίς να ασχοληθούμε με επουσιώδη ζητήματα όπως ο τύπος και η δομή του επεξεργαστή, οι δίαυλοι διασύνδεσης κ.λπ. Με αυτόν τον τρόπο δώσαμε απόλυτη προσοχή στο σχεδιασμό και την υλοποίηση του επιταχυντή καθ' αυτού παρά σε ζητήματα τα οποία ήταν δευτερεύουσας σημασίας για την επιτυχία του project.

Η δομή του Zynq αποτελείται από δύο διακριτά στοιχεία: το Σύστημα Επεξεργασίας (Processing System - PS) και την Προγραμματιζόμενη Λογική (Programming Logic - PL). Το κομμάτι PL περιλαμβάνει επαναπρογραμματιζόμενη λογική με τη μορφή FPGA, στοιχεία μνήμης με τη μορφή Block RAM, στοιχεία ψηφιακής επεξεργασίας σημάτων (DSP Slices) καθώς και interface επικοινωνίας όπως το PCIe interface. Το κομμάτι του PS από την άλλη περιλαμβάνει τον επεξεργαστή ARM, τον δίαυλο διασύνδεσης AXI μαζί με τα κατάλληλα interface, static και dynamic memory controllers, IO interfaces όπως USB, UART, GPIO και Gigabit Ethernet.

### Α.3.2 ZEDBoard

Ένας πολύ βολικός τρόπος για να πειραματιστούμε και να εξερευνήσουμε τις δυνατότητες ενός τσιπ είναι τα evaluation boards. Πρόκειται για πλακέτες οι οποίες περιλαμβάνουν το εκάστοτε τσιπ (συνήθως κάποιο μικροεπεξεργαστή, FPGA ή SoC) μαζί με διάφορα περιφερειακά, όπως LED, διακόπτες, οθόνες LCD, Ethernet



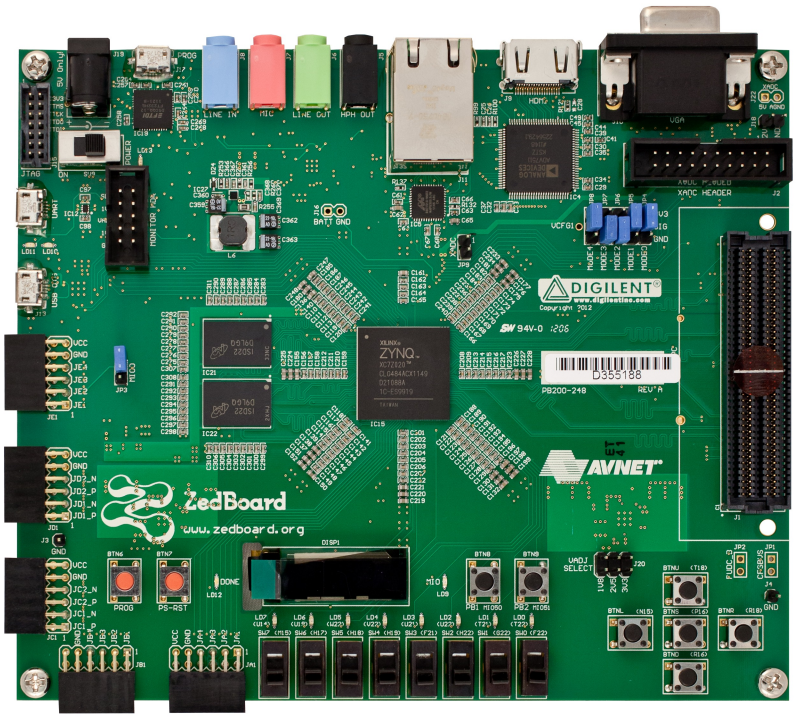

**Σχήμα Α.2**: Η αρχιτεκτονικη του Zynq-7000

modules, I/O για εικόνα και ήχο (πχ. VGA, HDMI, 3,5mm jacks) και μνήμες RAM και Flash. Η χρησιμότητα των evaluation boards βρίσκεται στο ότι μας επιτρέπουν να αναπτύξουμε ένα πρωτότυπο της εφαρμογής μας, να το δοκιμάσουμε και να πειραματιστούμε με αυτό. Στο συγκεκριμένο project χρησιμοποιήσουμε την Zedboard (Rev. D), μια πλακέτα η οποία βασίζεται στο τσιπ Zynq-7020 και εμπεριέχει όλα τα παραπάνω περιφερειακά μαζί με αρκετά ακόμα.

Η Zedboard είναι μια αναπτυξιακή πλακέτα χαμηλού κόστους. Είναι ένα σύστημα υλοποιημένο σε ολοκληρωμένο κύκλωμα (SoC) που ανήκει στην οικογένεια Zynq-7000 της Xilinx. Συνδυάζει την ύπαρξη Υπολογιστικού Συστήματος με δύο επεξεργαστές ARM με την ύπαρξη Επαναπρογραμματιζόμενης Λογικής. Υποστηρίζει την υλοποίηση Linux, Android, Windows, OS/RTOS εφαρμογών. Τα κύρια χαρα-

κτηριστικά της είναι:

- Μνήμη: δυναμική (DDR3) και στατική μνήμη (SPI Flash, SD Card Interface)
- USB: USB-to-UART σύνδεση, πρωτόκολλο JTAG
- Εικόνα και Ήχος: HDMI Transmitter, Analog Device Audio Codec, OLED Display
- Clock Sources: 33.33 MHz ρολόι για το PS ενώ το ίδιο παράγει έως 4 ρολόγια για το PL του ZYNQ. Στην πλακέτα υπάρχει ένας ταλαντωτής 100MHz και η όποιες αλλαγές στη συχνότητα γίνονται με PLL
- User I/O: 7 user GPIO push button, 8 user dip switches, 8 LEDs.
- 10/100/1000 Ethernet PHY: Ethernet θύρα για σύνδεση στο διαδίκτυο




**Σχήμα A.3**: ZEDBoard Rev. A

## Α.4 Προσομοιωση

Όπως αναφέραμε και προηγουμένως η διαδικασία της προσομοίωσης ήταν ιδιαίτερα σημαντική αλλά και επίπονη αφού ήταν η πρώτη ένδειξη για το αν το σύστημα λειτουργούσε όπως θα έπρεπε. Προκειμένου να το πετύχουμε αυτό χρησιμοποιήσαμε δύο εργαλεία που μας βοήθησαν σημαντικά στο να ελέγξουμε τη λειτουργία του επιταχυντή και διορθώσουμε τα όποια σφάλματα παρουσιάζονταν: τα ILA cores και τα Testbenches.

### Α.4.1 Testbenches

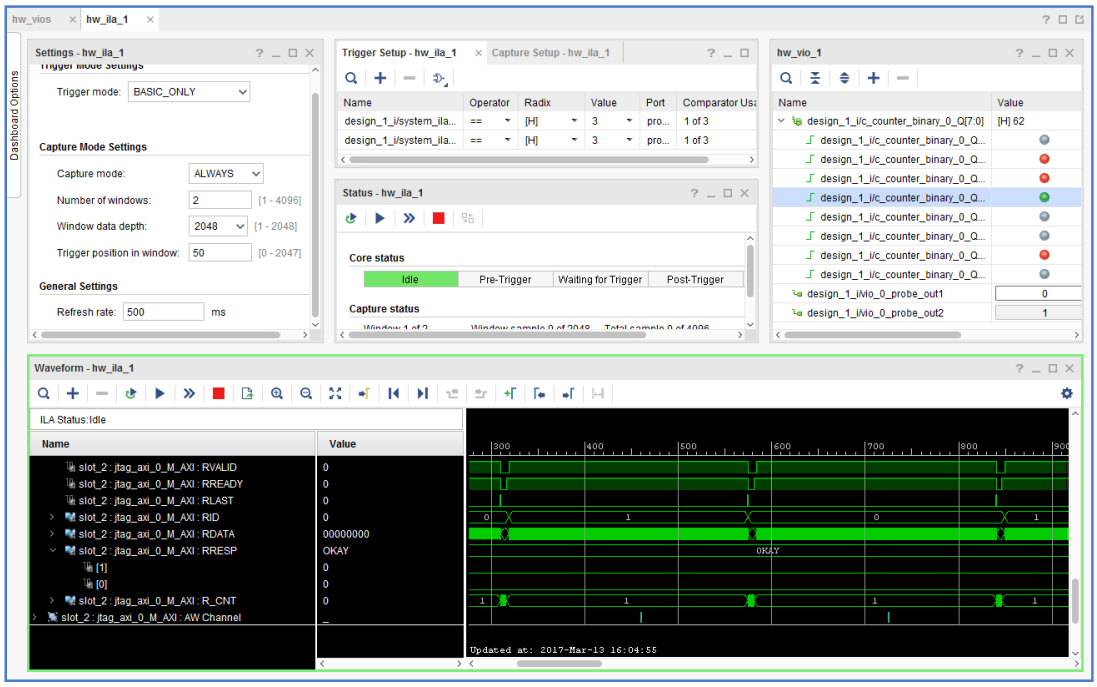
Ένα testbench είναι στην ουσία ένα εικονικό περιβάλλον (κώδικας VHDL πρακτικά) μέσα στο οποίο τοποθετείται ο πραγματικός κώδικάς VHDL που θέλουμε να ελέγξουμε/επαληθεύσουμε. Στην απλούστερη περίπτωση, εφαρμόζονται διανύσματα εισόδου στην υπό έλεγχο μονάδα και σε κάθε κύκλο ρολογιού εξετάζονται οι έξοδοι για να διαπιστωθεί αν συμμορφώνονται μ' ένα προκαθορισμένο σύνολο αναμενόμενων δεδομένων. Στην συγκεκριμένη εργασία χρησιμοποιήθηκαν έξι μεγάλα testbenches για τον έλεγχο και την επαλήθευση ξεχωριστών μονάδων του συστήματος. Οι μικρότερες μονάδες είτε ελέγχθηκαν απευθείας, εφαρμόζοντας χειροκίνητα διανύσματα δεδομένων στις εισόδους και ελέγχοντας την έξοδο, είτε με πολύ μικρά testbences τα οποία δε χρήζουν ιδιαίτερης αναφοράς.

### Α.4.2 ILA Cores

Ένα ILA core αποτελεί ένα παραμετροποιήσιμο IP περιφερειακό που περιλαμβάνεται στη βιβλιοθήκη της Xilinx και λειτουργεί ως λογικός αναλυτής (ILA - Integrated Logic Analyzer) μέσα στο σύστημα που θα ενσωματωθεί. Χρησιμεύει πρακτικά για τον ίδιο ακριβώς σκοπό που χρησιμεύει και ένας κανονικός λογικός αναλυτής, την παρακολούθηση και την απεικόνιση συγκεκριμένων εσωτερικών σημάτων του λογικού κυκλώματος. Ο ILA ενσωματώνει πολλές από τις προηγμένες λειτουργίες των αυτόνομων λογικών αναλυτών όπως σκανδαλισμό με εξισώσεις boolean και σκανδαλισμό κατά τη μετάβαση σε ακμή (edge-transition triggering). Τα κύρια χαρακτηριστικά του ILA Core IP είναι:

- Καθορισμός από τον χρήστη του αριθμού και του εύρους των σημάτων προς καταγραφή

- Καταγραφή πολλαπλών σημάτων με μία μόνο συνθήκη σκανδαλισμού
- AXI Interface για την αποσφαλμάτωση περιφερειακών AXI



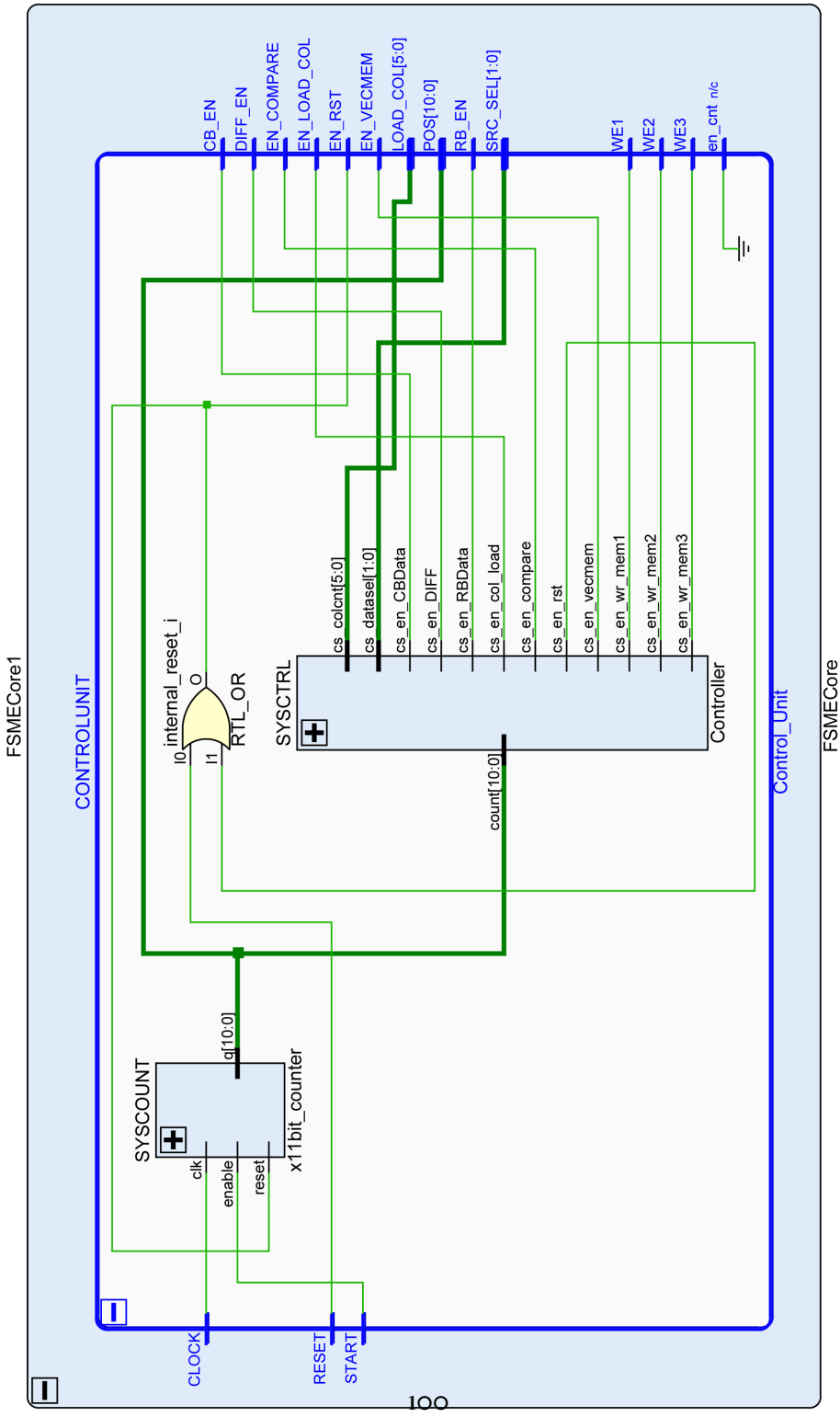

**Σχήμα Α.4**: Το περιβάλλον του ILA Debug μέσα από το Vivado

Οφείλουμε να σημειώσουμε ότι παρά την ευκολία που προσφέρει η χρήσ των ILA Cores οι δυνατότητες καταγραφής τους περιορίζονται από το ίδιο το περιβάλλον στο οποίο καλούνται να λειτουργήσουν. Αυτό σημαίνει ότι δεν μπορούμε να παρακολουθήσουμε οσοδήποτε μεγάλο αριθμό σημάτων, ούτε να έχουμε έναν πολύ μεγάλο ρυθμό δειγματοληψίας και αυτό διότι ο ILA core χρησιμοποιεί Block RAMs για να αποθηκεύει τα καταγεγραμμένα δείγματα. Δεδομένου ότι ο αριθμός των BRAM είναι πεπερασμένος (μεγάλο τμήμα των BRAM μπορεί επίσης να καταλαμβάνεται από το design που ελέγχουμε), εξίσου πεπερασμένος είναι και ο αριθμός των σημάτων (ή/και δειγμάτων) που μπορούμε να καταγράψουμε. Σε κάθε περίπτωση, χρησιμοποιώντας ένα ικανοποιητικό μέγεθος μνήμης για την αποθήκευση των δειγμάτων (8192 δείγματα είναι συνήθως ένα καλό tradeoff, ανάμεσα στη λεπτομέρεια της καταγραφής και το μέγεθος των BRAM που μπορούμε να χρησιμοποιήσουμε)

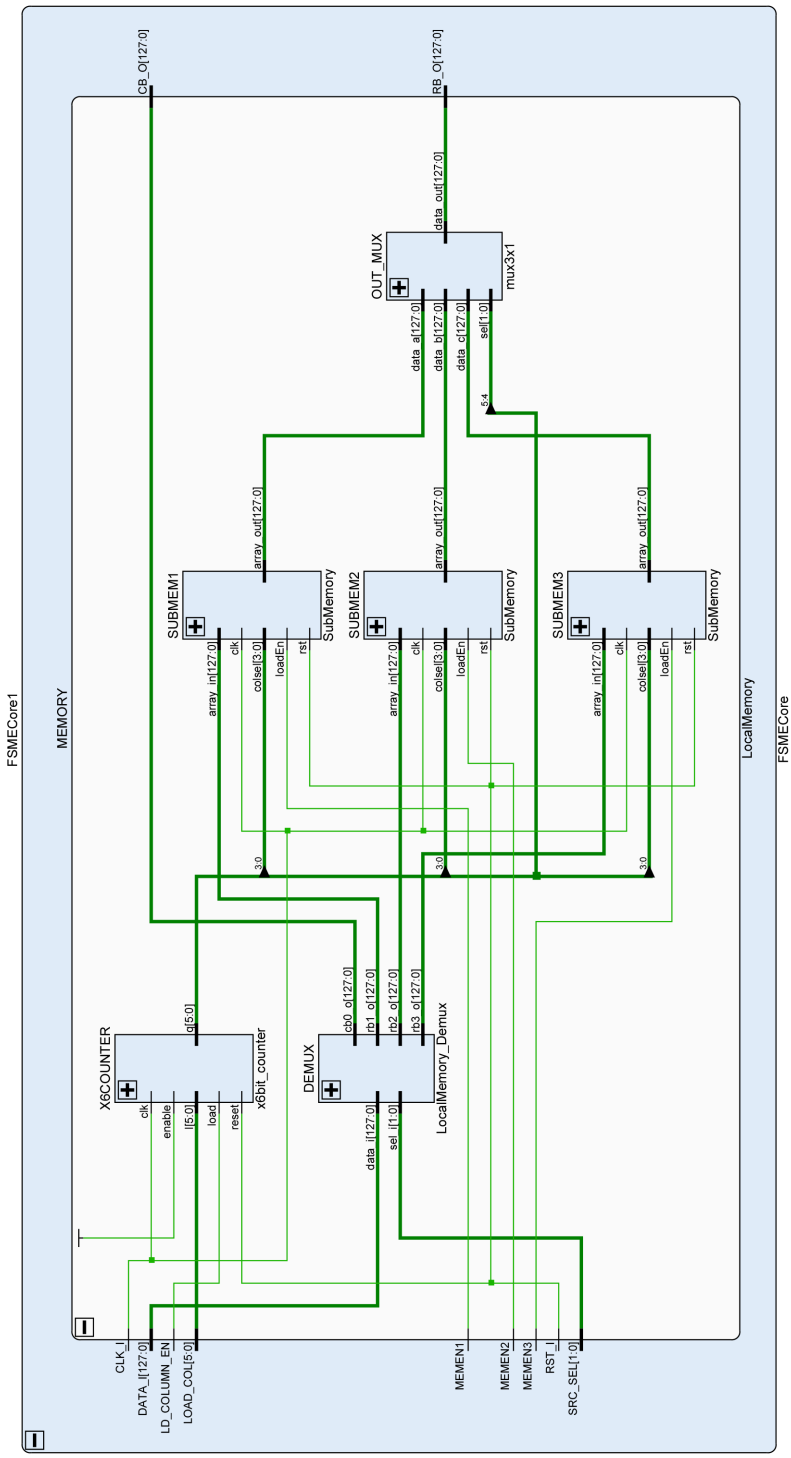
και ''έξυπνες'' συνθήκες καταγραφής μπορούμε να έχουμε εξαιρετικά αποτελέσματα που θα βοηθήσουν σημαντικά στην αντιμετώπιση τυχόν προβλημάτων.

# B

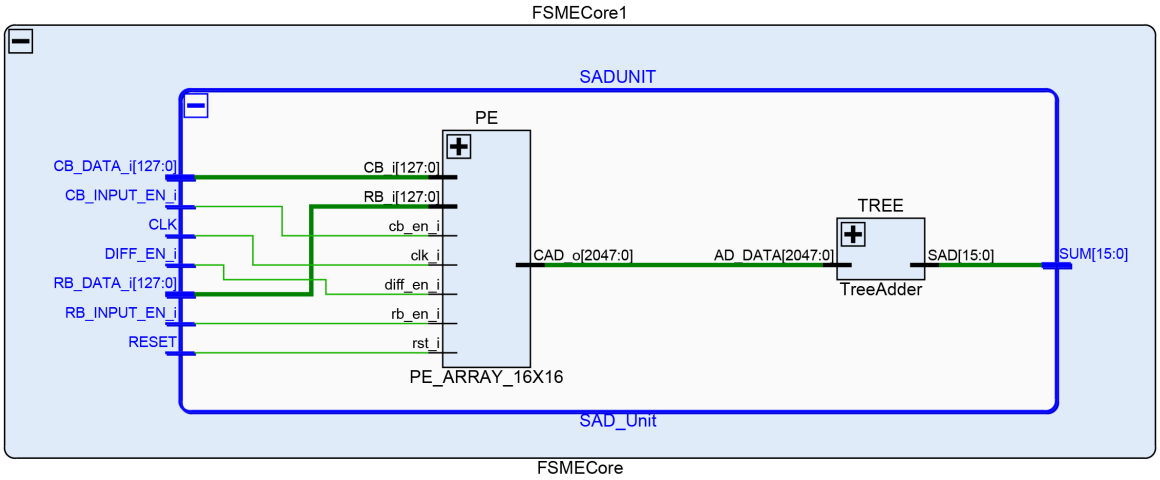
# Σχήματα και Διαγράμματα



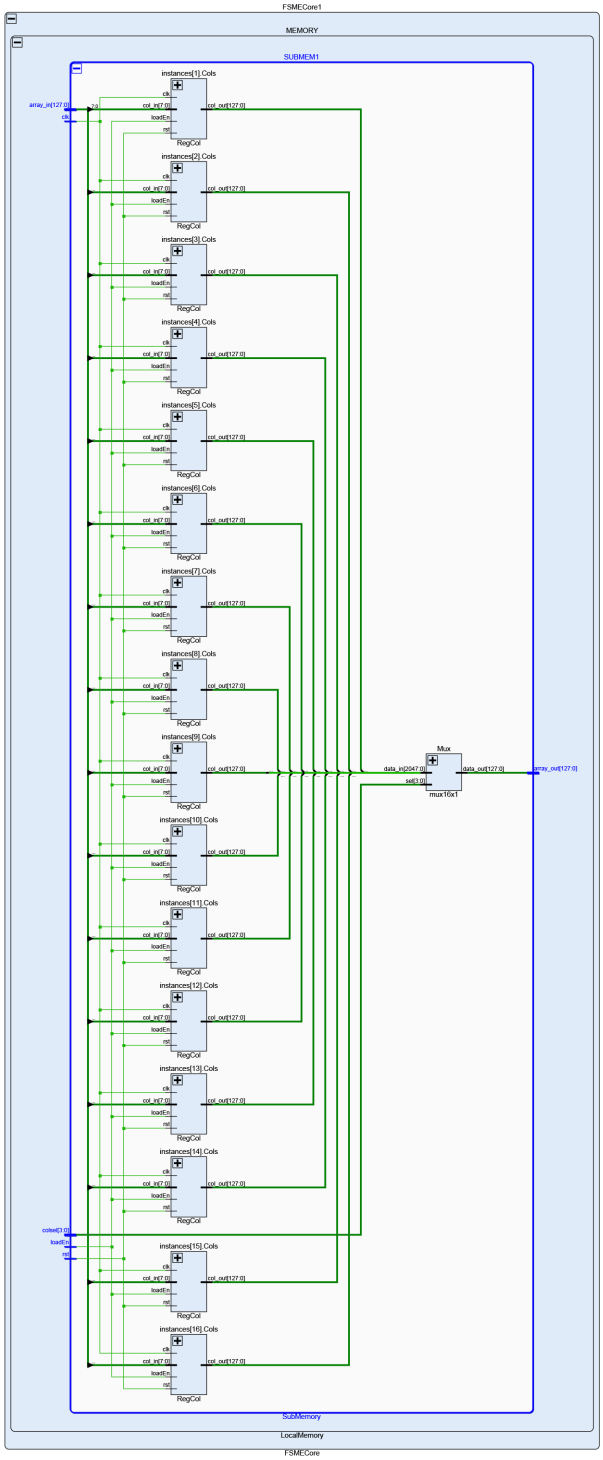

**Σχήμα Β.1**: RTL Διάγραμμα της μονάδας ελέγχου



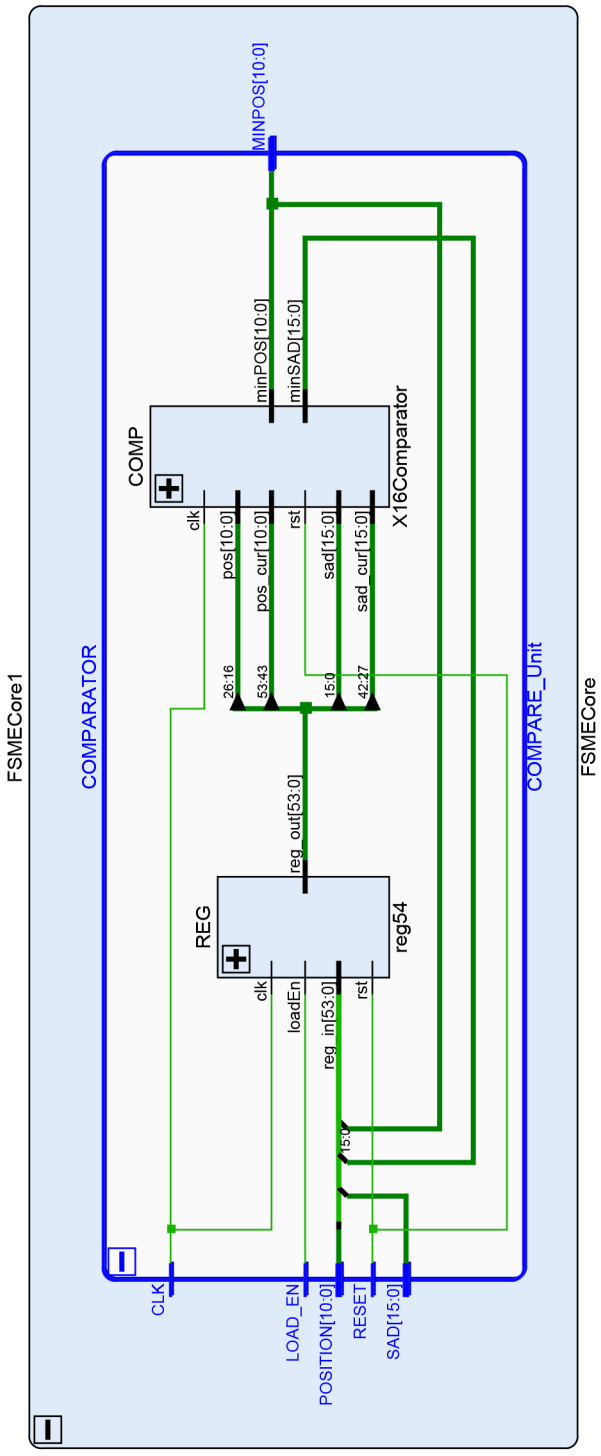

**Σχήμα B.2**: RTL Διάγραμμα της τοπικής μονάδας μνήμης



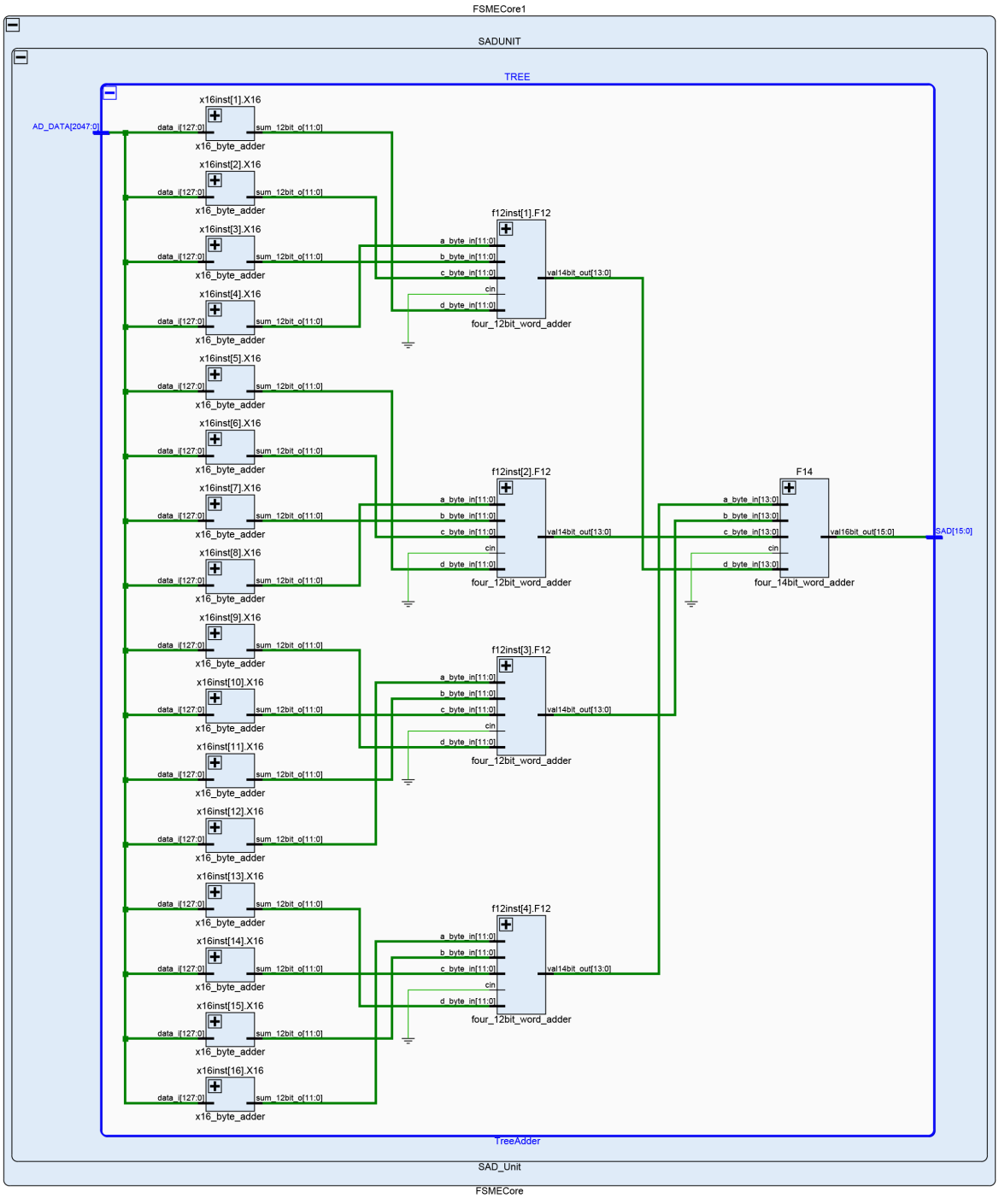

**Σχήμα B.3**: RTL Διάγραμμα της μονάδας υπολογισμού SAD



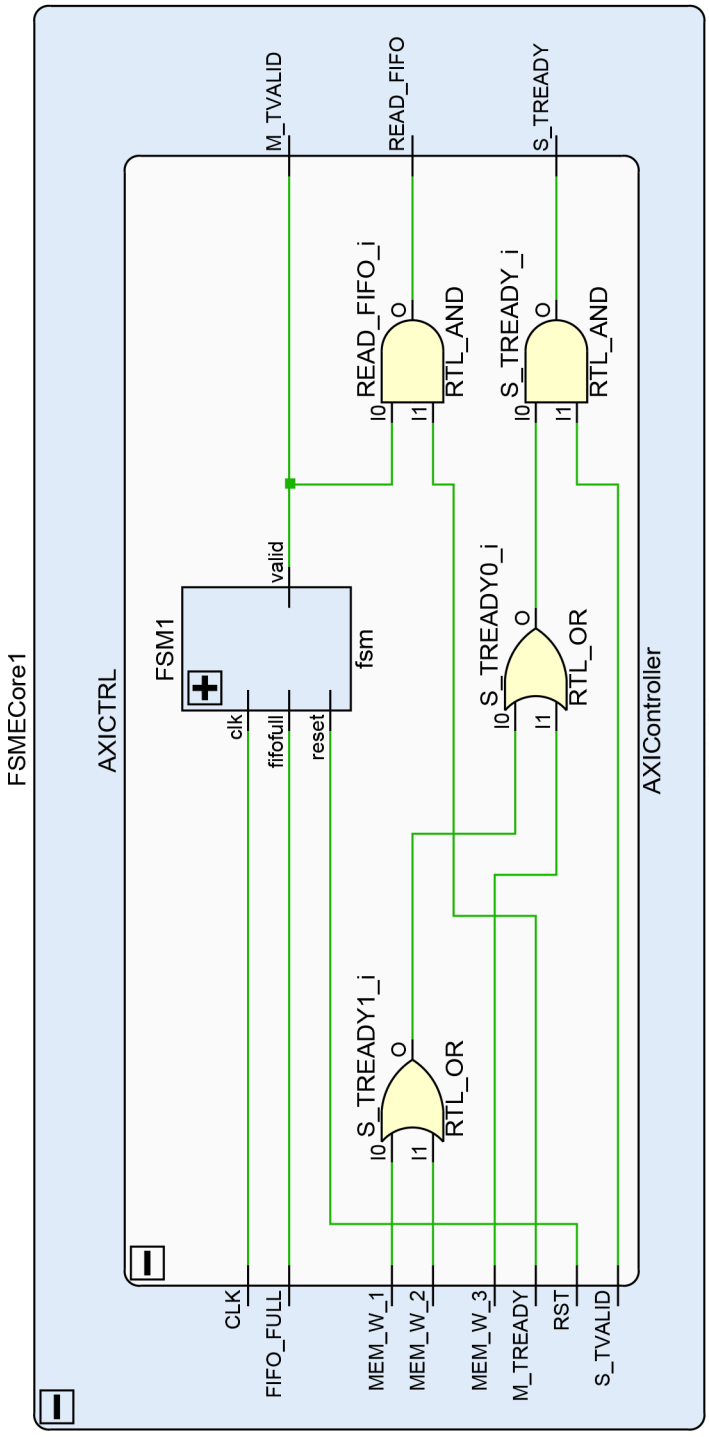

**Σχήμα B.4**: RTL Διάγραμμα τψν υπομονάδων μνήμης (submemories)



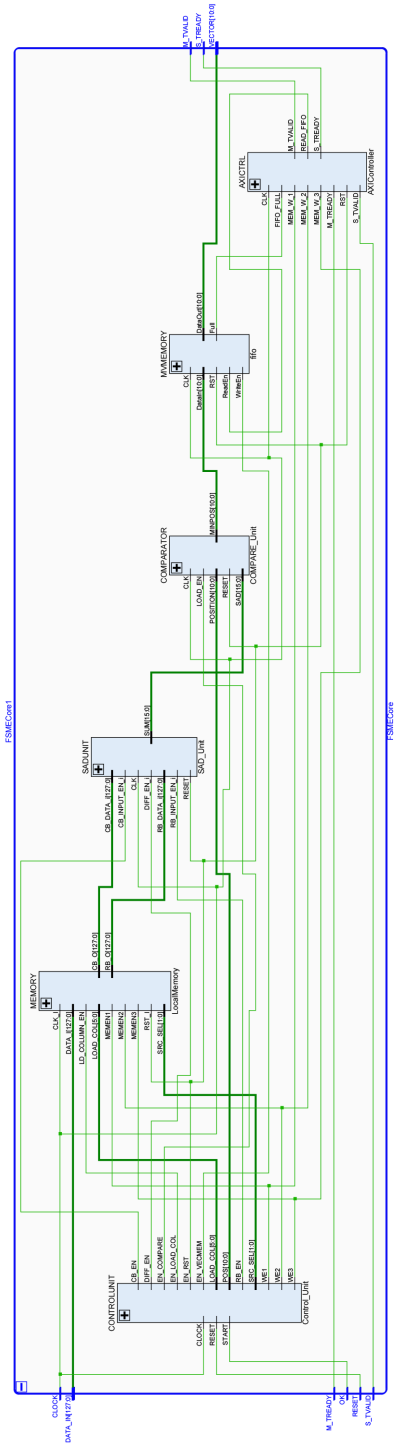

**Σχήμα B.5**: RTL Διάγραμμα του συγκριτή



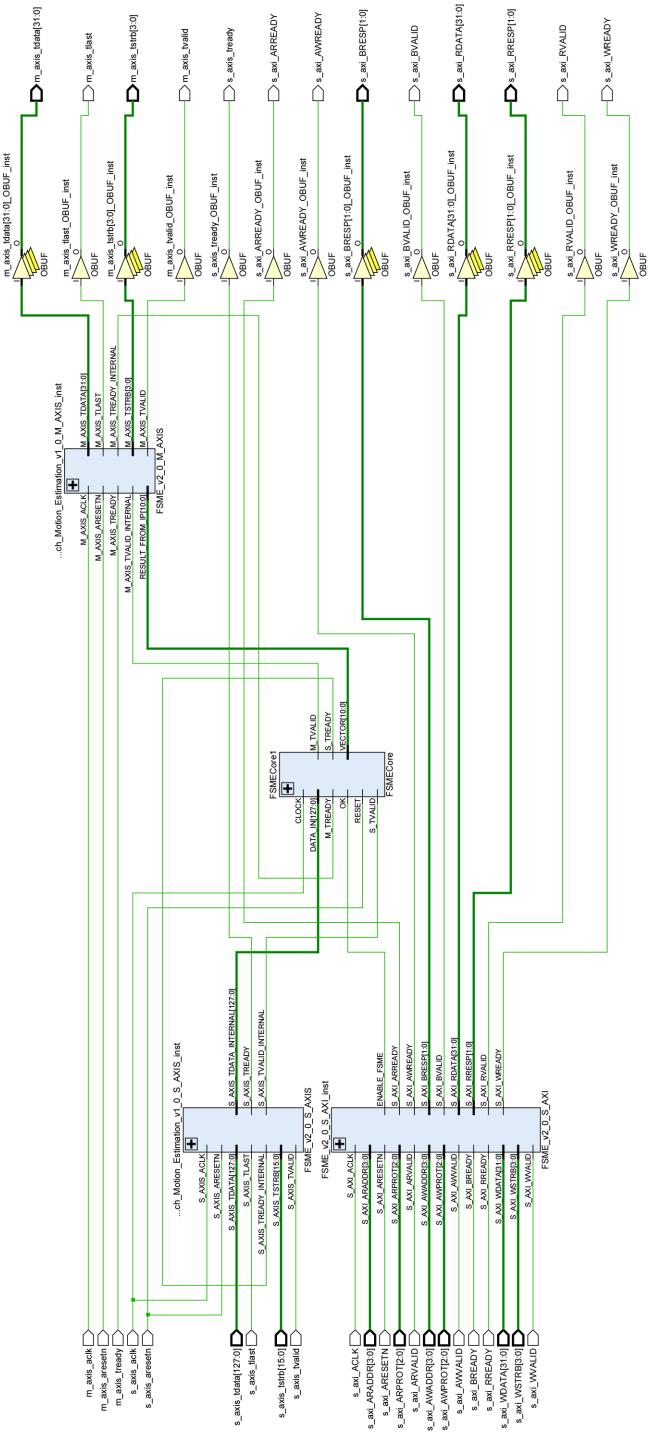

**Σχήμα Β.6**: RTL Διάγραμμα του αθροιστή δέντρου




**Σχήμα B.7**: RTL Διάγραμμα του ελεγκτη AXI Stream




**Σχήμα B.8**: RTL Διάγραμμα του επιταχυντή (Top-Level)



**Σχήμα B.9**: RTL Διάγραμμα ολόκληρου του Ενσωματωμένου Συστήματος

# Βιβλιογραφία


[1] Sven Andersson. New horizons zynq blog. Svenand.blogdrive.com. N, 2014, January 2016.

[2] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart. The zynq book tutorials. v, 1:2, September 2014.

[3] R. Dobai and L. Sekanina. Towards evolvable systems based on the xilinx zynq platform. In Evolvable Systems (ICES), pages 89–95, vol., no., , 16-19, April 2013. 2013 IEEE International Conference on.

[4] F. Eberli. Next generation fpgas and socs - how embedded systems can profit. In Computer Vision and Pattern Recognition Workshops (cvprw, editors, IEEE Conference on , vol, pages 610–613, 23-28, June 2013. no.

[5] C. Economakos, H. Sidiropoulos, and G. Economakos. Rapid prototyping of digital controllers using fpgas and esl/hls design methodologies. Automation and Computing (ICAC), 2013(19):1–6, 2013.

[6] Clément Foucher. Installing embedded linux on zedboard. 2015.

[7] Sumeer Goel, Yasser Ismail, and Magdy Bayoumi. High-speed motion estimation architecture for real-time video transmission. In Computer Journal, vol 55., Issue 1., , 25, pages 35–46. 2010.

[8] Y. Ismail, W. El-Medany, and H. et al. J Al-Junaid. Real-time image proc (2016) 11: 633. "high performance architecture for real-time hdtv broadcasting". Journal of Real-Time Image Processing, 27, May 2014.

[9] Yasser Ismail. A complete verification of a full search motion estimation engine. International Journal of Computing and Digital Systems, University of Bahrain, 1, October 2015.

[10] Yasser Ismail. A cost – effective programmable soc for h.265/hevc full search motion estimation using xilinx zynq-7 zc706 fpga. International Journal of Computing and Digital Systems, University of Bahrain, 1, January 2016.

[11] Jaswant Jain and Anil Jain. Displacement measurement and its application in interframe image coding. IEEE Transactions on communications, 29(12):1799–1808, 1981.

[12] Reoxiang Li, Bing Zeng, and Ming L Liou. A new three-step search algorithm for block motion estimation. IEEE transactions on circuits and systems for video technology, 4(4):438–442, 1994.

[13] H. Loukil, F. Ghozzi, and A. Samet. et al. "hardware implementation of block matching algorithm with fpga technology," 6th international conference on microelectronics, proceedings. pp, pages 542–546, 2004.

[14] T. Makryniotis and M. Dasygenis. Implementation of a motion estimation hardware accelerator on zynq soc. In 2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST), pages 1–4, May 2017.

[15] Thomas Makryniotis and Minas Dasygenis. Rapid implementation of embedded systems using xilinx zynq platform. In Proceedings of the SouthEast European Design Automation, Computer Engineering, Computer Networks and Social Media Conference, SEEDA-CECNSM '16, pages 6–10, New York, NY, USA, 2016. ACM.

[16] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar. Fpgas in industrial control applications. IEEE Transactions on Industrial Informatics, 7(2):224–243, 2011.

[17] David M. Harris Neil H. E. Weste. CMOS VLSI Design.

[18] J. Olivares, J. Hormigo, J. Villalba, I. Benavides, and E. L. Zapata. Sad computation based on online arithmetic for motion estimation. Microprocessors and Microsystems, Accepted with ref IJB/, 2005, 2005.

[19] Joaquın Olivares, Ignacio Benavides, Javier Hormigo, and Julio Villalba. Emilio Zapata "Fast Full-Search Block Matching Algorithm Motion Estimation Alternatives in FPGA".

[20] Volnei A. Pedroni. Σχεδιασμός κυκλωμάτων με τη VHDL.

[21] Harsha Prakash Redkar and Sonia Kuwelkar. Full search block matching algorithm motion estimation on fpga. in International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), 4:4, April 2015.

[22] Altera Press Release. Altera and ibm unveil fpga-accelerated power systems with coherent shared memory. SuperComputing, 2014, November 2014.

[23] S. Ren, Y. He, S. Elnikety, and K. S. McKinley. Exploiting processor heterogeneity in interactive services. Microsoft Research, ICAC, pages 45–58, 2013.

[24] Kumar Rethinagiri, S.; Palomar, O.; Arias Moreno, J.; Unsal, and O.; Cristal. A. In Heterogeneous Platform to Accelerate Compute Intensive Applications, pages 31–31, 2015 IEEE 23rd Annual International Symposium on , vol., no., , 2-6, May 2015. in Field-Programmable Custom Computing Machines (FCCM).

[25] D. Roggow, P. Uhing, P. Jones, and J. Zambreno. A project-based embedded systems design course using a reconfigurable soc platform. In Microelectronics Systems Education (MSE), pages 9–12, vol., no., , 20-21, May 2015. 2015 IEEE International Conference on.

[26] N. Roma, T. Dias, and L. Sousa. Customisable core-based architectures for real-time motion estimation on fpgas. LNCS, 2778:745–754, 2003.

[27] A. Ryszko and K. Wiatr. An assesment of fpga suitability for implementation of real-time motion estimation. EUROMICRO Symposium On Digital Systems Design, Proceedings, pp, pages 364–367, 2001.

[28] J. Silva, V. Sklyarov, and I. Skliarova. Comparison of on-chip communications in zynq-7000 all programmable systems-on-chip. in Embedded Systems Letters, IEEE, 7(1):31–34, March 2015.

[29] F. Slomka, M. Dorfel, R. Munzenberger, and R. Hofmann. Hardware/software codesign and rapid prototyping of embedded systems. in IEEE Design & Test of Computers, 17(2):28–38, 2000.

[30] Adam P. Taylor. How to use interrupts on the zynq soc. in Xilinx XCell Journal, 87:38–43, 2014.

[31] G. van der Wal, D. Zhang, I. Kandaswamy, J. Marakowitz, K. Kaighn, Joe Zhang, and S. Chai. Fpga acceleration for feature based processing applications. In Computer Vision and Pattern Recognition Workshops (cvprw, editors, IEEE Conference on , vol, pages 42–47, 7-12, June 2015. no.

[32] S. Wong, S. Vassiliadis, and S. Cotofana. A sum of absolute differences implementation in fpga hardware. 28:183–188, 2002.

[33] Xilinx. Ug585 - zynq-7000 all programmable soc technical reference manual, 2015. v1.10.

[34] Xilinx. Pg021 - axi interconnect v2.1 product guide, 2016.

[35] Xilinx. Pg059 - axi interconnect v2.1 product guide, 2017.

[36] H. Yun, Y. P. Lee, Y. S. Moon, and Y. Bae. Implementation of motor controller using zynq epp. In Soft Computing and Intelligent Systems (scis, editors, Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium, pages 1224–1228, Kitakyushu, 2014. on.

[37] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block matching motion estimation. In Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on, volume 1, pages 292–296. IEEE, 1997.

# Αρκτικόλεξα

**AMBA** Advanced Microcontroller Bus Architecture.

**ASIC** Application-Specific Integrated Circuit.

**AXI** Advanced eXtensible Interface.

**BRAM** Block-Random Access Memory.

**CB** Current Block.

**CPLD** Complex Programmable Logic Device.

**CPU** Central Processing Unit.

**DCT** Discrete Cosine Tranform.

**DDR** Double Data Rate.

**DMA** Direct Access Memory.

**DMAC** DMA Controller.

**DPCM** Differential Pulse-Code Modulation.

**DSP** Digital Signal Processor.

**DTB** Device Tree Blob.

**DTS** Device Tree Source file.

**DWT** Discrete Wavelet Transform.

**FHD** Full High Definition.

**FPGA** Field Programmable Gate Array.

**FPS** Frames Per Second.

**FSBL** First Stage Boot Loader.

**FSBMA** Full Search Block Matching Algorithm.

**FSM** Finite State Machine.

**FSME** Full Search Motion Estimation.

**GPIO** General Purpose Input Output.

**HD** High Definition.

**HEVC** High Efficiency Video Coding.

**HLS** High Level Synthesis.

**IEEE** Institute of Electrical and Electronics Engineers.

**ILA** Integrated Logic Analyzer.

**IP** Intellectual Property.

**JPEG** Joint Photographic Experts Group.

**LSB** Least Significant Bit.

**MAD** Mean of Absolute Difference.

**MJPEG** Motion JPEG.

**MM2S** Memory-Mapped to Stream.

**MPC** Maximum Pixel Count.

**MSB** Most Significant Bit.

**MSE** Mean Square Error.

**NTSC** National Television System Committee.

**PE** Processing Element.

**PL** Programmable Logic.

**PNG** Portable Network Graphics.

**PS** Prossesing System.

**RAM** Random Access Memory.

**RB** Reference Block.

**S2MM** Stream to Memory-Mapped.

**SAD** Sum of Absolute Difference.

**SG** Scatter-Gather.

**SoC** System-on-Chip.

**UART** Universal Asynchronous Receiver-Transmitter.

**UHD** Ultra High Definition.

**USB** Universal Serial Bus.

**VHDL** VHSIC Hardware Description Language.