Διεθνές Πανεπιστήμιο της Ελλάδος
Τμήμα Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών

Διπλωματική Εργασία **Πρόγραμμα Μεταπτυχιακών Σπουδών στην Ρομποτική**

Σχεδίαση συστήματος εξαγωγής ακμών εικόνας σε πραγματικό χρόνο, με κώδικα σε γλώσσα περιγραφής υλικού και υλοποίηση στο σύστημα PYNQ-Z1

Μπετούνης Ιωάννης

Διεθνές Πανεπιστήμιο της Ελλάδος

Σέρρες, Απρίλιος 2021

Η εργασία αυτήν υποβλήθηκε στο
Πρόγραμμα Μεταπτυχιακών Σπουδών στη Ρομποτική,
του Διεθνούς Πανεπιστημίου της Ελλάδος,
για τη μερική εκπλήρωση υποχρεώσεων για το Δίπλωμα Ειδίκευσης στη Ρομποτική

Επιβλέπων Καθηγητής : Ιωάννης Καλόμοιρος

Περίληψη

Η παρούσα διπλωματική εργασία έχει σκοπό την υλοποίηση εφαρμογής ανίχνευσης ακμών σε πραγματικό χρόνο, μέσω του φίλτρου Sobel, με χρήση της πλακέτας PYNQ-Z1 της εταιρίας Xilinx. Για την υλοποίηση, η είσοδος υψηλής ανάλυσης HDMI της πλακέτας δέχεται video από μία συσκευή με έξοδο HDMI. Στην συνέχεια, η έγχρωμη εικόνα μετατρέπεται σε εικόνα αποχρώσεων του γκρι, έπειτα παράγονται οι ακμές μέσω του φίλτρου Sobel και τέλος, τα παραγόμενα πλαίσια της ακολουθίας video εξάγονται στην έξοδο υψηλής ανάλυσης HDMI της πλακέτας. Από την έξοδο HDMI της πλακέτας PYNQ-Z1, το video προβάλλεται σε οθόνη, με ρυθμό εξήντα πλαισίων το δευτερόλεπτο.

Το σύστημα ανίχνευσης ακμών, με το φίλτρο Sobel, σχεδιάζεται σε γλώσσα περιγραφής υλικού VHDL. Για την υλοποίηση του κυκλώματος, γίνεται χρήση του προγραμματιστικού περιβάλλοντος Vivado της εταιρείας Xilinx. Τα κυκλώματα εισόδου και εξόδου του ψηφιακού βίντεο, υλοποιούνται με βαθμίδες IP (intellectual property) ώστε το κύκλωμα στην συνέχεια να δοκιμασθεί στη διάταξη Zynq 7000 της πλακέτας PYNQ-Z1. Η πλακέτα PYNQ-Z1 αναδεικνύεται σε ένα εύχρηστο εργαλείο σχεδίασης εφαρμογών ψηφιακού βίντεο, με σχετικά χαμηλό κόστος.

Abstract

The present diploma thesis aims to implement a real-time edge detection application, based on the Sobel filter, using the Xilinx PYNQ-Z1 board. For implementation, the HDMI high-resolution input of the board receives video from a device with HDMI output. The color image is then converted to a grayscale image, then the edges of the image are generated via the Sobel filter, and finally the generated video sequence frames are output to the HDMI display connector of the board. From the HDMI output of the PYNQ-Z1 board, the video is projected on a screen at a rate of sixty frames per second.

The edge detection system, with the Sobel filter, is designed using the VHDL hardware description language. To implement the circuit, the Xilinx Vivado programming environment was used, with the appropriate input and output IP blocks, so that the circuit can then be tested with the Zynq 7000 device of the PYNQ-Z1 board.

Περιεχόμενα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή της παρούσας διπλωματικής εργασίας κ. Καλόμοιρο Ιωάννη, για την αμέριστη βοήθεια που μου προσέφερε. Η συμβολή του ήταν καταλυτική για την ολοκλήρωση της παρούσας διπλωματικής, μέσω των γνώσεων και του χρόνου που διέθεσε.

Επίσης, θα ήθελα να ευχαριστήσω και τους υπόλοιπους εισηγητές του μεταπτυχιακού προγράμματος της Ρομποτικής, οι οποίοι προσέφεραν τις κατάλληλες γνώσεις και εφόδια, τόσο για την παρούσα υλοποίηση όσο και γενικότερα, για την κατανόηση της επιστήμης της ρομποτικής.

1. Κεφάλαιο Πρώτο. Επιτάχυνση αλγορίθμων – Συστήματα επεξεργασίας πραγματικού χρόνου

1.1 Εισαγωγή

Το θέμα της παρούσας διπλωματικής εργασίας είναι η επιτάχυνση της επεξεργασίας video από μια πηγή εικονοσειράς (video) υψηλής ανάλυσης, ώστε η επεξεργασία να γίνεται με υψηλό ρυθμό. Έτσι, η επεξεργασία μπορεί να βρει εφαρμογή σε ιδιαίτερα απαιτητικά προβλήματα. Η κεντρική βαθμίδα επεξεργασίας είναι μια βαθμίδα ανίχνευσης ακμών, που στηρίζεται στη μέθοδο του Sobel. Οι ακμές περιγράφουν το σχήμα των αντικειμένων του περιβάλλοντος. Η τεχνολογία των ακμών έχει εφαρμογές όπως

- Εξαγωγή χαρακτηριστικών για κινούμενα αυτόνομα οχήματα
- Αναγνώριση αντικειμένων
- Ιατρική απεικόνιση

Η επιτάχυνση της επεξεργασίας της εικόνας απαιτείται κυρίως σε εφαρμογές όπου υπάρχει σημαντικός ρυθμός ροής πληροφορίας. Για παράδειγμα, στα αυτόνομα οχήματα είναι πολύ σημαντικό η επεξεργασία της εικόνας να γίνεται με μεγάλο ρυθμό ανανέωσης πλαισίων, λόγω του ότι το περιβάλλον μέσα στο οποίο κινείται το όχημα είναι δυναμικό και μπορεί να εναλλάσσεται ταχύτατα. Το ίδιο συμβαίνει και σε άλλες εφαρμογές της ρομποτικής όρασης, όπου το ρομπότ καλείται να αποφασίσει σε πραγματικό χρόνο για τις ενέργειές που θα ακολουθήσουν, με βάση τις εικόνες που λαμβάνει από το περιβάλλον όπου κινείται. Ένα σύστημα όπου η επεξεργασία και η λήψη αποφάσεων γίνεται μέσα στα όρια του κρίσιμου χρόνου που καθορίζει η φύση του προβλήματος, ονομάζεται σύστημα επεξεργασίας πραγματικού χρόνου.

Η ρομποτική όραση είναι μια επιστήμη η οποία αναπτύχθηκε παράλληλα με τις τεχνολογικές εξελίξεις στην επιστήμη των μικροεπεξεργαστών και των κυκλωμάτων. Η ρομποτική όραση επιτυγχάνεται μέσω της ανάπτυξης κατάλληλων μαθηματικών αλγορίθμων, για την επεξεργασία της εικόνας εισόδου από πηγές εισόδου εικόνας, όπως μια κάμερα, ώστε να ληφθεί μια απόφαση από το σύστημα σχετικά με το περιβάλλον. Η ρομποτική όραση

μιμείται τον τρόπο με τον οποίο ο άνθρωπος μέσω της όρασης αντιλαμβάνεται το περιβάλλον.

Λόγω των εφαρμογών αυτών της ρομποτικής όρασης είναι πολύ σημαντικό να βελτιωθεί ο χρόνος επεξεργασίας της εικόνας εισόδου. Ο χρόνος που χρειάζεται μια εικόνα για να μετατραπεί σε εικόνα ακμών, μπορεί να επιταχυνθεί σημαντικά μέσω ενός FPGA (Field Programmable Gate Array), το οποίο έχει την δυνατότητα να εκτελέσει παράλληλη επεξεργασία δεδομένων. Με την χρήση ενός FPGA, η επεξεργασία της εικόνας με σκοπό την μετατροπή της σε εικόνα ακμών, μπορεί να γίνει με ρυθμό πολλών πλαισίων το δευτερόλεπτο, λόγω της επιτάχυνσης που επιτυγχάνουμε στους αλγορίθμους που εκτελούνται.

Οι υπολογισμοί που εκτελούνται στους αλγόριθμους, για την επεξεργασία της εικόνας σε πραγματικό χρόνο, ποικίλουν ως προς την πολυπλοκότητα και τον όγκο τους, ανάλογα με την εφαρμογή και το απαιτούμενο αποτέλεσμα. Έτσι, θα μπορούσαμε να έχουμε υπολογισμό πολλαπλών συνελίξεων, για παράδειγμα, εφαρμογή πολλών φίλτρων, σε παράθυρα μεγέθους 15 x 15 ή 25 x 25, ταυτόχρονα. Ένα άλλο παράδειγμα ανάγκης για επιτάχυνση των υπολογισμών, είναι μια εφαρμογή πυραμίδας gauss, με την εκτέλεση υπολογισμών για πέντε συνελίξεις. Άλλες εφαρμογές όπου απαιτείται επιτάχυνση αλγορίθμων, είναι οι εφαρμογές της μηχανικής μάθησης (Deep Learning) ή των εφαρμογών Cloud, όπου απαιτείται καλή υπολογιστική ισχύς στα data center. Η απόδοση των συστημάτων και των εφαρμογών αυτών, σχετίζεται με τον χρόνο που απαιτείται να εκτελεσθούν οι αλγόριθμοι. Όσο ελαχιστοποιούνται οι χρόνοι υπολογισμών, τόσο πιο αποδοτικά γίνονται τα συστήματα αυτά και ταυτόχρονα διευρύνεται η βάση εφαρμογής τους.

Στις ημέρες μας, η απαίτηση για υπολογιστική ισχύ συνεχώς αυξάνεται σε όλες τις συσκευές που χρησιμοποιούμε καθημερινά. Ένα απλό παράδειγμα είναι η χρήση των φορητών συσκευών, κινητής τηλεφωνίας. Στις συσκευές αυτές έχουν συγκεντρωθεί πολλές λειτουργίες και δυνατότητες για τον χρήστη τους. Για παράδειγμα η συσκευές αυτές προσφέρουν δυνατότητες όπως, λήψη φωτογραφιών και βίντεο υψηλής ανάλυσης (π.χ. 4K και 8K), σύνδεση εφαρμογών συσκευής με το Cloud, εφαρμογές τεχνητής νοημοσύνης (π.χ. Amazon Alexa), εφαρμογές αναγνώρισης αντικειμένων μέσω Cloud (π.χ. Google Lens) κ.τ.λ. . Οι εφαρμογές αυτές καθημερινώς εμπλουτίζονται και επίσης, νέες αντίστοιχες εφαρμογές

δημιουργούνται. Οι εφαρμογές αυτές επιζητούν συνεχώς καλύτερες υπολογιστικές δυνατότητες και περισσότερους πόρους, από τις συσκευές αυτές.

Επίσης, έχουν αναπτυχθεί βιβλιοθήκες Υπολογιστικής Όρασης, όπως για παράδειγμα η OpenCV, που επιτρέπουν την επεξεργασία εικόνας και βίντεο με χρήση λογισμικού. Με την χρήση των βιβλιοθηκών αυτών, επιτυγχάνουμε την επεξεργασία της εικόνας ή του βίντεο, με υπολογισμούς που εκτελούνται από το λογισμικό. Αν και το τελικό αποτέλεσμα της επεξεργασίας της εικόνας μέσω τέτοιων συναρτήσεων είναι το ίδιο, ο ρυθμός της επεξεργασίας είναι μικρότερος, σε σχέση με την παραλληλοποίηση του αλγόριθμου σε υλικό, ειδικά σε περιπτώσεις μεγάλης ανάλυσης εικόνας ή σε περιπτώσεις της λεγόμενης "βαριάς επεξεργασίας". Αντίθετα, μια υλοποίηση τον υπολογισμών αυτών σε διάταξη FPGA, μπορεί να πετύχει πολύ υψηλούς ρυθμούς ανανέωσης πλαισίων. Για τον λόγο αυτό, οι διατάξεις FPGA έχουν κερδίσει σημαντικό έδαφος και η χρήση τους διευρύνεται συνεχώς, ειδικά στο πεδίο της επιτάχυνσης της επεξεργασίας video, αλλά και σε άλλα πεδία, όπου η επιτάχυνση είναι κρίσιμη.

1.2 Συστήματα επεξεργασίας πραγματικού χρόνου

Ο όρος συστήματα πραγματικού χρόνου παρουσιάστηκε πρώτα στον αμερικανικό στρατό. Συγκεκριμένα το έτος 1949, κατασκευάστηκε ένα σύστημα για το αμερικανικό ναυτικό, με την ονομασία Whirlwind project. Ένα έτος αργότερα, συγκεκριμένα το 1950, ένα ακόμα αμυντικό σύστημα πραγματικού χρόνου, κατασκευάστηκε για την αεροπορία της Αμερικής, με την ονομασία Semiautomatic Ground Environment – SAGE. [1] Το Whirlwind project, αναπτύχθηκε από το πανεπιστήμιο MIT. Το σύστημα αυτό, ήταν ο πρώτος υπολογιστής ο οποίος υπολόγιζε παράλληλα και όχι σε σειρά, όπως συνέβαινε έως τότε, χρησιμοποιώντας μνήμες με δυνατότητα μαγνητικής εγγραφής. [2]

Έτσι λοιπόν σύστημα πραγματικού χρόνου ονομάζεται πλέον, το σύστημα το οποίο εκτελεί περισσότερες από μια (multi-tasking) δραστηριότητες και εξάγει αποτελέσματα μέσα σε συγκεκριμένα χρονικά πλαίσια. Αυτό που χαρακτηρίζει κυρίως ένα σύστημα πραγματικού χρόνου, είναι ότι η σύνδεση και η επικοινωνία των επιμέρους συστημάτων του, γίνεται σε

προκαθορισμένο χρονικό πλαίσιο. Η σύνδεση και η επικοινωνία αυτή, εξαρτάται κυρίως από την ταχύτητα των επεξεργαστών ή των ελεγκτών, που υπάρχουν στο σύστημα. [1]

Ωστόσο η έννοια "πραγματικός χρόνος" εξακολουθεί να είναι σχετική και άρρηκτα συνδεδεμένη με την εφαρμογή και το αποτέλεσμα που θέλουμε να πετύχουμε. Για παράδειγμα εάν έχουμε μια κάμερα με δυνατότητα λήψης 60 fps, σε ένα αυτόνομο όχημα που κινείται γρήγορα, ο πραγματικός χρόνος για την επεξεργασία της εισερχόμενης εικόνας είναι τα 60 fps. Σε περίπτωση που επεξεργασία γινότανε με ρυθμό ανανέωσης μικρότερο, τότε θα υπήρχε υστέρηση στους τελικούς υπολογισμούς και πιθανότατα το τελικό αποτέλεσμα δεν θα ικανοποιούσε τον σκοπό του συστήματος. Σε αυτήν την περίπτωση παίζει μεγάλο ρόλο η ταχύτητα με την οποία κινείται το όχημα.

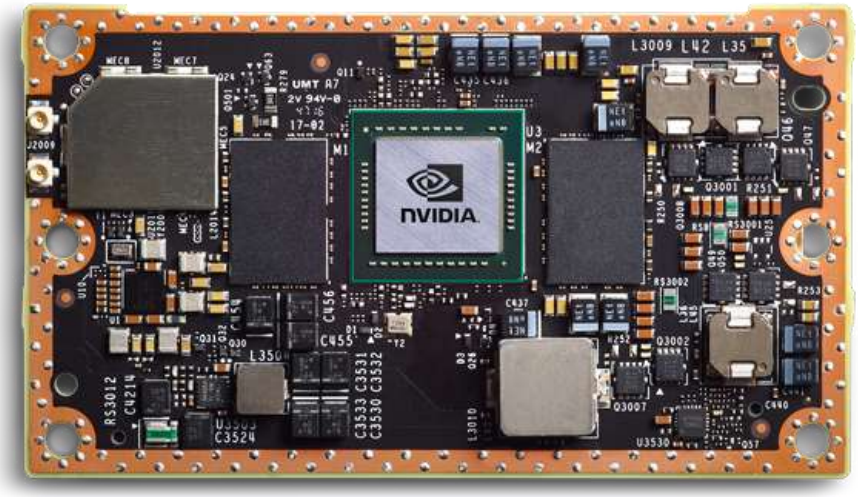
Ένας σημαντικός παράγοντας που παίζει ρόλο στον χρόνο επεξεργασίας της εισερχόμενης εικόνας, από τις κάμερες του αυτόνομου οχήματος, είναι η ανάλυση. Ανεξαρτήτως της ταχύτητας, λοιπόν, είναι πολύ σημαντικό να ορίσουμε από την αρχή την επιθυμητή ανάλυση της εικόνας που λαμβάνεται από τις κάμερες. Εάν επιλεγεί μικρή ανάλυση, για παράδειγμα 640 x 480, τότε ο απαιτούμενος χρόνος για την επεξεργασία της εικόνας θα είναι μικρός. Με τις σημερινές όμως αναλύσεις που υπάρχουν στις κάμερες, συνήθως η ελάχιστη ανάλυση είναι περίπου στα 1280 x 726. Αυτού του μεγέθους οι αναλύσεις απαιτούν μεγαλύτερη υπολογιστική ισχύ και επιτάχυνση των αλγορίθμων υπολογισμού, ώστε να εξαχθούν τα αποτελέσματα σε πραγματικό χρόνο, για τις ανάγκες της εκάστοτε εφαρμογής.

1.2.1 Συστήματα επεξεργασίας GPUs

Οι επεξεργαστές GPU (Graphics Processing Unit), είναι ειδικού σκοπού ηλεκτρονικά κυκλώματα. Σχεδιάστηκαν με σκοπό την επιτάχυνση της δημιουργίας εικόνας αλλά και των απαιτούμενων frame buffers, για την έξοδο της εικόνας στην οθόνη. Ο επεξεργαστής GPU, χρησιμοποιείται στα ενσωματωμένα συστήματα, στα κινητά τηλέφωνα, στους προσωπικούς υπολογιστές, σε σταθμούς εργασίας και σε παιχνιδομηχανές. Επίσης, χρησιμοποιούνται στη μηχανική όραση, με σκοπό την επεξεργασία της εισερχόμενης εικόνας. Οι επεξεργαστές GPU, που χρησιμοποιούν την πιο πρόσφατη τεχνολογία και είναι πολύ αποδοτικοί. Αυτό που

κάνει τους επεξεργαστές GPU αποδοτικότερους από τους CPU, είναι ότι ο σχεδιασμός τους αποτελείται από μια δομή υψηλού παραλληλισμού στοιχείων υπολογισμού. Ο σχεδιασμός αυτός, τους καθιστά ιδανικούς για αλγοριθμικές πράξεις παράλληλες για μεγάλο όγκο δεδομένων. [16]

Η ικανότητα των GPU να επεξεργάζονται ταχύτατα αλγόριθμους εικόνας, μπορεί να συγκριθεί με ένα τυπικό FPGA. Παρόλα αυτά όμως, αυτές οι υψηλές ταχύτητες, δεν είναι πλήρως διαθέσιμες κατά την επεξεργασία εικόνων. Αυτό συμβαίνει επειδή η εκτέλεση των αλγορίθμων ανατίθεται στον επεξεργαστή GPU από εξωτερική πηγή (π.χ. λογισμικό), με αποτέλεσμα αυτό να προκαλεί και καθυστέρηση στη ροή των δεδομένων, από την λήψη της εικόνας μέχρι την επεξεργασία της. [17]



Εικόνα 1.2.1.1 Η πλακέτα Jetson TX2 με δύο επεξεργαστές, έναν GPU και έναν ARM

Πηγή : https://developer.nvidia.com/embedded/jetson-tx2

Στην εικόνα 1.2.1.1, φαίνεται η πλακέτα Jetson TX2, της NVIDIA, η οποία χρησιμοποιείται σε εφαρμογές ρομποτικής όρασης. Η πλακέτα αυτήν έχει δύο επεξεργαστές, έναν GPU και έναν ARM. Ο επεξεργαστής ARM, συντονίζει την λειτουργία και στον επεξεργαστή GPU, εκτελούνται όλοι οι αλγοριθμικοί υπολογισμοί.

1.2.2 Συστήματα επεξεργασίας ASICs

Τα συστήματα ASICs σχεδιάζονται αποκλειστικά για μια εφαρμογή κάθε φορά και δεν μπορούν να επαναχρησιμοποιηθούν για άλλη εφαρμογή από αυτήν που σχεδιάστηκαν. Συστήματα ASICs, περιέχονται στις φορητές συσκευές κινητής τηλεφωνίας. Χαρακτηριστικό τους είναι η χαμηλή κατανάλωση ενέργειας και οι υψηλές επιδόσεις. Στα μειονεκτήματα αυτών των κυκλωμάτων συγκαταλέγονται το υψηλό κόστος παραγωγής τους και ο μεγάλος χρόνος που απαιτείται για την ανάπτυξη τους. [6] Τα τσιπ ASICs, μπορούν να χρησιμοποιηθούν, για παράδειγμα, για την κατασκευή ενός ψηφιακού εγγραφέα φωνής ή για την κατασκευή μιας συσκευής υψηλής επίδοσης, για την εξόρυξη bitcoin (bitcoin miner). Τα σημερινά κυκλώματα ASICs, συχνά περιλαμβάνουν μικροεπεξεργαστές, μνήμες ROM, RAM, EEPROM και τέλος μνήμες flash. Τα κυκλώματα ASICs πολλές φορές χαρακτηρίζονται ως system-on-chip (SoC) και οι σχεδιαστές τους συχνά χρησιμοποιούν γλώσσα περιγραφής υλικού (HDL), όπως τη γλώσσα Verilog ή την γλώσσα VHDL, ώστε να περιγράψουν τη λειτουργικότητα που επιτελεί ένα κύκλωμα ASIC. [7]

Ανάλογα την εφαρμογή για την οποία προορίζεται ένα τσιπ ASIC, μπορεί να παρουσιάσει πλεονεκτήματα έναντι των FPGA. Ένα πλεονέκτημα για παράδειγμα, έναντι των FPGA, είναι ότι μπορεί να έχει μεγαλύτερες ταχύτητες χρονισμού. Τα τσιπ ASIC, βοηθάνε πολύ τις φορητές συσκευές κινητής τηλεφωνίας, στους υπολογισμούς που καλούνται να κάνουν. Για παράδειγμα η λήψη φωτογραφιών υψηλής ανάλυσης, αλλά και η επεξεργασία τους μετά την λήψη γίνεται γρήγορα, χάρη στα τσιπ αυτά.

Στην ρομποτική όραση οι εφαρμογές που χρησιμοποιούν τα τσιπ ASIC, μπορεί να είναι, για παράδειγμα, εφαρμογές για την μετατροπή της εισερχόμενης εικόνας σε εικόνα ακμών. Λόγω της μεγάλης ταχύτητας χρονισμού που μπορούν να πετύχουν τα τσιπ αυτά, καθίστανται ιδανικά για αυτές τις εφαρμογές.

1.2.3 Συστήματα επεξεργασίας FPGAs

Η τεχνολογία των διατάξεων πυλών που προγραμματίζονται στο πεδίο (Field-Programmable Gate Arrays-FPGAs) προέκυψε από την προγραμματιζόμενη μνήμη μόνο για ανάγνωση (PROM) και τις προγραμματιζόμενες λογικές συσκευές (PLDs). Η μνήμη PROM και οι συσκευές PLDs, είχαν την δυνατότητα να προγραμματιστούν είτε κατά την κατασκευή τους στη γραμμή παραγωγής (programmed in batches in a factory) είτε με προγραμματισμό στο πεδίο (field-programmable).

Ο αρχιτεκτονική κατασκευής ενός FPGA διαφέρει από κατασκευαστή σε κατασκευαστή. Μια τυπική δομή ενός FPGA περιέχει προγραμματιζόμενες λογικές πύλες, προγραμματιζόμενα I/O μπλοκ και προγραμματιζόμενες συνδέσεις μεταξύ τους. [35]

Εικόνα 1.2.3.1 Το FPGA τσιπ του PYNQ-Z1

Η τεχνολογία των διατάξεων FPGAs σήμερα, συνεχίζει να αναπτύσσεται συνεχώς. Για παράδειγμα, η τεχνολογία FPGA χρησιμοποιείται στους λεγόμενους μαλακούς επεξεργαστές (soft-processors), οι οποίοι αντικαθιστούν τους σκληρούς επεξεργαστές. Οι μαλακοί επεξεργαστές έχουν την δυνατότητα να επαναπρογραμματίζονται σε πραγματικό χρόνο λειτουργίας τους, έτσι ώστε να προσαρμόζονται στις υπολογιστικές και λειτουργικές απαιτήσεις που εκτελούν ανά πάσα στιγμή. Η τελευταία εξέλιξη στις διατάξεις FPGA είναι η ανάπτυξη αρχιτεκτονικής η οποία περιέχει μια υβριδική μορφή. Σε αυτήν την υβριδική μορφή συνυπάρχουν επεξεργαστές με μία σειρά από πυρήνες, καθώς και λογικά στοιχεία τεχνολογίας FPGA, στην ίδια διάταξη. Ένα τέτοιο παραδείγματα υβριδικής τεχνολογίας είναι η διάταξη Zynq-7000. [34]

Τα FPGA χρησιμοποιούνται πολύ στην επεξεργασία εικόνας, καθώς προσφέρουν ευελιξία και βελτιωμένους χρόνους επιτάχυνσης αλγοριθμικών υπολογισμών. Οι υπολογισμοί σε ένα FPGA, μπορεί να εκτελούνται παράλληλα, εφόσον γίνει ο σχεδιασμός του κυκλώματος σε αυτό.

1.2.4 Συστήματα επεξεργασίας με επεξεργαστή ARM και GPU

Ένα συστήμα επεξεργασίας με επεξεργαστή ARM αλλά και GPU, είναι η πλακέτα Raspberry Pi 4. Η αναπτυξιακή πλακέτα αυτή, λόγω των δυνατοτήτων που παρέχει μπορεί να χρησιμοποιηθεί, στην ρομποτική όραση. Η πλακέτα αυτήν διαθέτει λειτουργικό σύστημα Linux. Αυτό επιτρέπει να φορτωθεί η γλώσσα Python σε αυτό και λόγω αυτού μπορεί να εκτελεί στην συνέχεια εντολές της OpenCV.

Το Raspberry Pi 4, διαθέτει θύρες USB και HDMI, δίνοντας την δυνατότητα να συνδέσουμε σε αυτό κάμερες. Με την βοήθεια της Python και της GPU που έχει, είναι δυνατόν να εκτελεσθούν αλγόριθμοι για την επεξεργασία εικόνας. Ο επεξεργαστής ARM που διαθέτει, έχει το ρόλο του συντονιστή, για την εκτέλεση των αλγορίθμων της προς επεξεργασίας εικόνας, διατηρώντας χαμηλούς τους χρόνους επεξεργασίας.

1.2.4 Ετερογενές σύστημα επεξεργασίας

Τα ετερογενή συστήματα επεξεργασίας, είναι αυτά τα οποία συνδυάζουν τις δυνατότητες ενός επεξεργαστή ARM και τις δυνατότητες που παρέχει ένα FPGA. Ένα τέτοιο σύστημα για παράδειγμα είναι το Zynq 7000. Τα ετερογενή συστήματα λόγω ότι διαθέτουν επεξεργαστή ARM, εκτελούν ένα λειτουργικό σύστημα Linux και επίσης μπορούν να εκτελέσουν την γλώσσα Python και κατ’ επέκταση εντολές της OpenCV, για την επεξεργασία εικόνας. Η Python επίσης διαθέτει βιβλιοθήκες, οι οποίες μπορούν να αξιοποιήσουν της δυνατότητες του FPGA και έτσι να εκτελεσθούν γρήγορα αλγόριθμοι για την επεξεργασία της εικόνας.

Τα ετερογενή συστήματα είναι αποδοτικά, λόγω των δυνατοτήτων που τους δίνει το FPGA και χρησιμοποιούνται για την επεξεργασία εικόνας σε πραγματικό χρόνο. Πραγματικός χρόνος, εδώ, σημαίνει ότι η επεξεργασία γίνεται στα όρια του κρίσιμου χρόνου, που ορίζει η ροή της κίνησης, που αποτυπώνει το video.

1.3 Επίλυση του προβλήματος

Για την επίλυση του προβλήματος της επιτάχυνσης της ανίχνευσης ακμών, δημιουργήθηκε κατάλληλο ψηφιακό σύστημα σε γλώσσα περιγραφής υλικού VHDL, με εφαρμογή του φίλτρου Sobel. Επίσης, σχεδιάστηκε ένα ολοκληρωμένο σύστημα επεξεργασίας, για τα περιφερειακά της πλακέτας PYNQ-Z1, με χρήση του λογισμικού Vivado. Στη συνέχεια, έγινε διαμόρφωση της διάταξης Zynq-7000, στην οποία στηρίζεται η υπολογιστική επεξεργασία της πλακέτας PYNQ.

Η χρήση FPGA για την επίλυση του προβλήματος, προσφέρει βελτίωση του χρόνου επεξεργασίας της εικόνας εισόδου. Άλλη συνεισφορά της λύσης που επιλέχθηκε είναι εκπαιδευτική, καθώς επιτυγχάνεται η καλύτερη κατανόηση της χρήσης και λειτουργίας των διατάξεων FPGA. Οι γνώσεις επάνω στην τεχνολογία του FPGA διευρύνονται. Επίσης, εμβαθύνουμε στη χρήση της γλώσσας περιγραφής υλικού VHDL για την δημιουργία κυκλωμάτων και τέλος αποκτάται μια χρήσιμη εμπειρία – εξοικείωση στην χρήση των περιφερειακών συσκευών της διάταξης. Τέλος, εξετάζουμε τις δυνατότητες της κάρτας PYNQ-Z1 στη σχεδίαση εφαρμογών όρασης μηχανής, καθώς πρόκειται για ευέλικτο σύστημα ανάπτυξης εφαρμογών, που έχει σχετικά χαμηλό κόστος.

Κεφάλαιο Δεύτερο. Μεθοδολογία και εργαλεία ανάπτυξης

2.1 Η διάταξη Zynq-7000 SoC

Πριν την ανάπτυξη του Zynq, τα συστήματα που συνδύαζαν επεξεργαστή με FPGA, παρουσίαζαν αρκετή πολυπλοκότητα στην επικοινωνία μεταξύ της προγραμματιζόμενης λογικής (Programmable Logic-PL) και του συστήματος επεξεργασίας (Processing System-PS). Η αρχιτεκτονική του Zynq βασίζεται στο AXI (Advanced eXtensible Interface), με αποτέλεσμα να παρέχει συνδέσεις μεταξύ του επεξεργαστή και του FPGA, οι οποίες είναι γρήγορες και υψηλού εύρους σε bits. Επιπλέον, η χρήση ενός διπύρηνου επεξεργαστή ARM Cortex - A9 στο Zynq, παρέχει περαιτέρω σημαντικές βελτιώσεις στην απόδοση. [15]

Το ZYNQ-7000, επιτρέπει στους μηχανικούς να υλοποιούν συστήματα χαμηλού κόστους, με υψηλές επιδόσεις. Η ύπαρξη του επεξεργαστή ARM Cortex - A9 στο Zynq, επιτρέπει την χρήση λογισμικού όπως το Linux, δίδοντας επιπλέον δυνατότητες, όπως για παράδειγμα τη χρήση των βιβλιοθηκών της Python για την επεξεργασία εικόνας ή βίντεο. Το ZYNQ-7000, έχει διαφορετική τροφοδοσία ενέργειας για το PL και το PS. Αυτό δίνει την δυνατότητα στον μηχανικό να απενεργοποιήσει το PL σύστημα, χωρίς να απενεργοποιηθεί το PS σύστημα. [42]

Υπάρχουν κάποια κοινά σημεία μεταξύ ενός κοινού FPGA και του Zynq στην ροή της σχεδίασης. Στο πρώτο στάδιο της σχεδίασης μιας αρχιτεκτονικής, ορίζονται οι προδιαγραφές και οι απαιτήσεις του συστήματος. Στο δεύτερο στάδιο, γίνεται ο σχεδιασμός του συστήματος και καθορίζονται οι συνδέσεις μεταξύ των υποσυστημάτων. Το δεύτερο στάδιο είναι σημαντικό, διότι η απόδοση του όλου συστήματος στηρίζεται σε αυτό. Στο τρίτο στάδιο, γίνεται η ανάπτυξη του υλικού και του λογισμικού μέρους του συστήματος και τέλος ο έλεγχος του συστήματος που σχεδιάστηκε. Σε επίπεδο προγραμματιζόμενης λογικής (PL), θα πρέπει να προσδιορισθούν επακριβώς τα απαιτούμενα μπλοκ και να δομηθούν ως βαθμίδες, με τις κατάλληλες συνδέσεις μεταξύ τους. Σε επίπεδο συστήματος επεξεργασίας (PS), γίνεται η ανάπτυξη κατάλληλου κώδικα για την εκτέλεση του. [41]

Σε ένα ολοκληρωμένο περιβάλλον σχεδίασης ψηφιακού συστήματος, τα προγραμματιζόμενα λογικά κυκλώματα, παρέχονται συχνά με την μορφή βιβλιοθηκών, που

περιλαμβάνουν προσχεδιασμένες βαθμίδες. Έτσι, ο σχεδιαστής μπορεί να έχει πρόσβαση με ευκολία σε αυτά, μέσω του λογισμικού σχεδίασης, όπως είναι για παράδειγμα το Vivado. Με αυτόν τον τρόπο, ο σχεδιαστής μπορεί να διαλέξει αυτό που ταιριάζει στην εφαρμογή που σχεδιάζει, με μια απλή αναζήτηση στις βιβλιοθήκες. Σε περίπτωση που κανένα κύκλωμα που βρίσκεται στην βιβλιοθήκη δεν ταιριάζει για την εφαρμογή, ο σχεδιαστής μπορεί να κατασκευάσει ένα νέο κύκλωμα και στη συνέχεια, αφού το αποθηκεύσει, να το χρησιμοποιήσει ξανά και ξανά σε μελλοντικές εφαρμογές. [15]

Σε μια τυπική εφαρμογή κυκλώματος σε ένα FPGA, ο μηχανικός θα πρέπει να οργανώσει σε αυτό την ακολουθία των κυκλωμάτων και να διασφαλίσει την φόρτωση των bitstream. Με το Zynq SoC, ο ενσωματωμένος επεξεργαστής του, μπορεί να εκτελέσει τις εργασίες ενός συμβατικού μικροελεγκτή, περιλαμβάνοντας και την λειτουργία ελέγχου των προγραμματιζόμενων λογικών κυκλωμάτων και των άλλων ενσωματωμένων περιφερειακών. [15]

2.2 Η αρχιτεκτονική του Zynq

Το Zynq διαθέτει ένα επεξεργαστή ARM cortex-A9, με δύο πυρήνες. Το σύστημα περιέχει μνήμη on-chip, διεπαφές εξωτερικής μνήμης και ένα μεγάλο αριθμό από περιφερειακά I/O. Ο σχεδιασμός του Zynq έγινε για υψηλές επιδόσεις με χαμηλή κατανάλωση ρεύματος, με τεχνολογία των 28 nm. Τα κυκλώματά του είναι κατασκευασμένα από υλικά υψηλής αγωγιμότητας. [43]

Όπως συμβαίνει σε κάθε FPGA, η προγραμματιζόμενη λογική του Zynq, περιέχει λογικές βαθμίδες (CLBs) τα οποία περιέχουν δύο μέρη. Κάθε μέρος περιέχει τέσσερεις πίνακες look-up (LUTs), οκτώ Flip-Flops (FFs) και έναν πίνακα διασυνδέσεων (accompanying switch matrix). Επίσης, η διάταξη περιέχει μπλοκ μνήμης RAM και DSP. [41]

Ο επεξεργαστής μπορεί να διαμορφώσει τα περιφερειακά κυκλώματα I/O, με βάση το GUI του συστήματος PS. Βάσει του σχεδιασμένου συστήματος μπορούν να διαμορφωθούν απλές κοινές λειτουργίες για τα I/O, όπως σειριακές θύρες, SPI, I2C κλπ. Πολυπλοκότερες

λειτουργίες είναι δυνατόν να πραγματοποιηθούν κατόπιν παρέμβασης στο μητρώο του συστήματος. Οι πυρήνες του Zynq συνδέουν τα σήματα διασύνδεσης με τα υπόλοιπα ενσωματωμένα συστήματα στην προγραμματιστική λογική. Αυτές οι διασυνδέσεις μεταξύ του συστήματος επεξεργασίας και της προγραμματιζόμενης λογικής αποτελούνται από τρεις ομάδες. Η πρώτη ομάδα είναι της εκτεταμένης πολυπλεξιμότητας I/O (EMIO), η δεύτερη ομάδα είναι η προγραμματιζόμενη λογική I/O και η τρίτη είναι η ομάδα AXI I/O. Ο πυρήνας του συστήματος έχει συμβατότητα με την διασύνδεση AXI4. Οι διασυνδέσεις AXI μπορούν να χρησιμοποιηθούν για μεταφορά δεδομένων από ή προς μια βαθμίδα AXI4-συμβατή. Οι βαθμίδες AXI4 χαρακτηρίζονται ως host ή client και συνδέονται με τον πυρήνα του επεξεργαστή ARM.

Η παραμετροποίηση του πυρήνα του Zynq, μπορεί να είναι διαφορετική για κάθε εφαρμογή. Οι παράμετροι ενεργοποιούν ή απενεργοποιούν τις διεπαφές και τις λειτουργίες της συσκευής. Αυτές οι παράμετροι ενημερώνονται κατά τη σχεδίαση της εφαρμογής. Οι θύρες που θα χρησιμοποιηθούν για την εφαρμογή και σχετίζονται με συγκεκριμένα περιφερειακά της πλακέτας ενεργοποιούνται ή απενεργοποιούνται. Κατά την μεταγλώττιση της σχεδίασης χρησιμοποιούνται οι πληροφορίες για τις θύρες που θα ενεργοποιηθούν, για να γίνει η περιγραφή του συστήματος επεξεργασίας (PS) στο αρχείο .tcl που αποθηκεύει όλα τα στοιχεία του υλικού.

Όσον αφορά τα ρολόγια του συστήματος Zynq, αυτά μπορούν να οριστούν σε ένα εύρος συχνοτήτων. Ο επεξεργαστής μπορεί να λάβει χρονισμό από 50 MHz έως 667 MHz (λαμβάνει τον συγχρονισμό από το ARM PLL), η μνήμη από 200 MHz έως 534 MHz (λαμβάνει τον συγχρονισμό από το DDR PLL) και οι θύρες I/O από 10 MHz έως 534 MHz (λειτουργεί με τον συγχρονισμό από το I/O PLL). [43]

2.3 Το PYNQ

Το PYNQ είναι ένα project ανοιχτού κώδικα, από την εταιρεία Xilinx, που έχει υλικό και λογισμικό μέρος. Το PYNQ είναι ένας συνδυασμός από τσιπ και πλακέτες, με σκοπό την υποστήριξη της Python, για την επιτάχυνση των υπολογισμών που θα εκτελεσθούν. Το PYNQ διευκολύνει επίσης την χρήση των πλατφορμών της Xilinx. Οι μηχανικοί και οι απλοί

χρήστες, μπορούν να κατασκευάσουν τα ηλεκτρονικά κυκλώματα πιο αποδοτικά με αυτό. Το PYNQ επιτρέπει την χρήση της Python και των βιβλιοθηκών της, ώστε οι σχεδιαστές να μπορούν να κάνουν χρήση των οφελών της προγραμματιστικής λογικής και των μικροεπεξεργαστών. [14] Το λογισμικό μέρος του PYNQ, διαθέτει πολλές βιβλιοθήκες της Python, με τις οποίες ο χρήστης μπορεί να διαχειρίζεται το υλικό, μέσω του λογισμικού. Πιο συγκεκριμένα, οι βιβλιοθήκες αυτές έχουν την δυνατότητα να χρησιμοποιούν την προγραμματιζόμενη ψηφίδα (FPGA) του PYNQ, για την εκτέλεση των υπολογισμών, επιταχύνοντας με αυτόν τον τρόπο την διαδικασία.

Το PYNQ μπορεί να χρησιμοποιηθεί σε αναπτυξιακά συστήματα που στηρίζονται σε διατάξεις όπως Zynq, Zynq UltraScale+, Zynq RFSoC της εταιρείας Xilinx, για την δημιουργία εφαρμογών υψηλής απόδοσης όπως: [14]

- Παράλληλη εκτέλεση με χρήση υλικού
- Υψηλής ευκρίνειας επεξεργασία βίντεο
- Επιτάχυνση αλγορίθμων με χρήση υλικού
- Επεξεργασία σήματος σε πραγματικό χρόνο
- Υψηλό εύρος ζώνης IO
- Έλεγχος καθυστέρησης σε χαμηλό επίπεδο

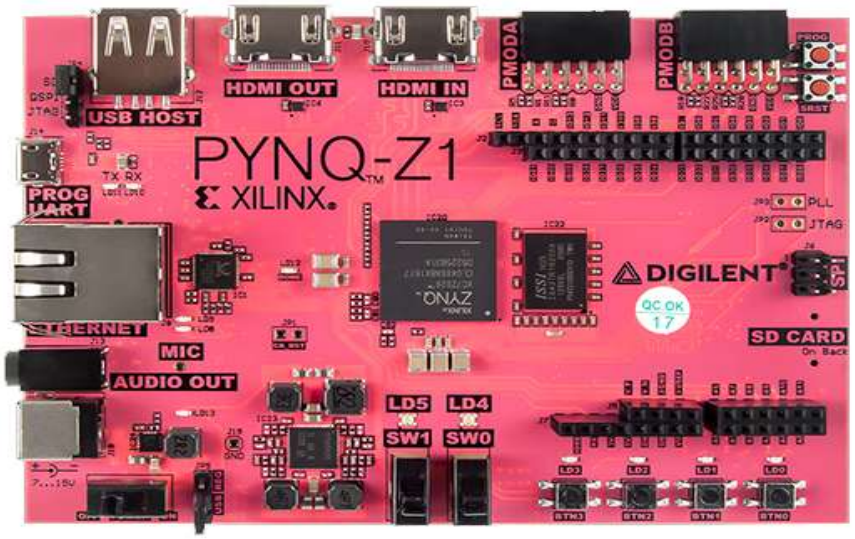
Η χρήση του PYNQ γενικότερα ενδείκνυνται για: [14]

- κατασκευή κυκλωμάτων, τα οποία μπορούν αργότερα να επαναχρησιμοποιηθούν ολόκληρα ή τμηματικά, από άλλους χρήστες
- ταχεία και εύκολη κατασκευή πρωτοτύπων και αναπτυξιακών μοντέλων
- για τον σχεδιασμό κυκλώματος υλικού χωρίς την χρήση σχεδιαστικών εργαλείων ASIC.

Το PYNQ προσφέρει μεγάλη ευχέρεια στον σχεδιασμό ενός κυκλώματος, χωρίς να απαιτείται η χρήση πολύπλοκων σχεδιαστικών προγραμμάτων, για τον σχεδιασμό των λογικών πυλών του κυκλώματος. Για παράδειγμα ακόμα και με την απλοποίηση των προγραμματιζόμενων λογικών κυκλωμάτων κατά την μεταγλώττισή τους, οι σχεδιαστές στο παρελθόν έπρεπε να διαχειριστούν ένα πολύπλοκο σύστημα διαδικασιών, για τη φόρτωση των απαιτούμενων αρχείων διαμόρφωσης (bitstream). Με το PYNQ αυτή η διαδικασία έχει απλοποιηθεί αποτελεσματικά, μέσω του περιβάλλοντος που προσφέρει. [15]

2.4 Η αναπτυξιακή πλακέτα PYNQ-Z1

Η αναπτυξιακή πλακέτα PYNQ-Z1 είναι ένα μοντέλο της Digilent, η οποία εκτός από τον επεξεργαστή ZYNQ XC7Z020-1CLG400C διαθέτει και ένα επεξεργαστή 650MHz dual-core Cortex-A9 processor. Αυτό δίδει την δυνατότητα στο μοντέλο αυτό να τρέχει λειτουργικό σύστημα Linux, το οποίο υποστηρίζει προγραμματισμό σε μια γλώσσα υψηλότερου επιπέδου, όπως είναι η Python. Αυτή η αναπτυξιακή πλακέτα παρέχει κάποια πλεονεκτήματα. Ένα πλεονέκτημα είναι ότι λόγω της χρήσης της Python και των βιβλιοθηκών της, είναι εύκολη η επεξεργασία βίντεο και εικόνων. Ένα άλλο πλεονέκτημα είναι ότι με την χρήση κατάλληλων βιβλιοθηκών μπορεί να γίνει ανάπτυξη στην πλακέτα εφαρμογής Deep Learning ή Machine Learning. Στην εργασία αυτή, ωστόσο, δεν χρησιμοποιήσαμε την Python ως γλώσσα προγραμματισμού. Οι δοκιμές που κάναμε περιλαμβάνουν σχεδίαση μόνον στους πόρους της διάταξης FPGA, με χρήση της γλώσσας περιγραφής υλικού VHDL.



Εικόνα 2.4.1 Η αναπτυξιακή πλακέτα PYNQ-Z1

2.5 Η γλώσσα περιγραφής υλικού VHDL

Για την διεκπεραίωση της παρούσας διπλωματικής εργασίας, χρησιμοποιήθηκε η γλώσσα περιγραφής υλικού VHDL. Η γλώσσα περιγραφής υλικού VHDL γενικότερα,

αναπτύχθηκε με σκοπό τον έλεγχο και την τεκμηρίωση ψηφιακών κυκλωμάτων στα κέντρα έρευνας και ανάπτυξης του υπουργείου άμυνας των ΗΠΑ. Η γλώσσα VHDL σήμερα, χρησιμοποιείται για την κατασκευή μοντέλων κυκλωμάτων σε μορφή κειμένου, περιγράφοντας με αυστηρό τρόπο ένα λογικό κύκλωμα. Η γλώσσα αυτήν είναι ικανή να χειρίζεται παραλληλισμούς που υπάρχουν στην σχεδίαση υλικού.

Τα πλεονεκτήματα της VHDL, είναι ότι χρησιμοποιείται τόσο για την περιγραφή των κυκλωμάτων, όσο και για την προσομοίωση του σχεδιαζόμενου κυκλώματος. Η προσομοίωση του συστήματος, γίνεται πριν την μεταγλώττιση του κώδικα και την φόρτωση της σύνθεσης σε ένα FPGA. Ένα ακόμα πλεονέκτημα είναι ότι η VHDL, είναι γλώσσα ροής δεδομένων, επιτρέποντας έτσι τον σχεδιασμό ταυτόχρονων συστημάτων. Τέλος με την γλώσσα VHDL, υπάρχει η δυνατότητα χρήσης ενός υπάρχοντος σχεδιασμένου κυκλώματος, σε μια νέα σχεδίαση ενός άλλου κυκλώματος. Επιτρέπει, δηλαδή την επαναχρησιμοποίηση κώδικα (reuse) και την ιεραρχική σχεδίαση.

Στη σχεδίαση των κυκλωμάτων με την γλώσσα VHDL, φροντίζουμε το κύκλωμα να αποτελείται πάντα από μια τουλάχιστον οντότητα, η οποία θα περιγράφει τη βαθμίδα και την αρχιτεκτονική της. Η αρχιτεκτονική, περιγράφει τον τρόπο με τον οποίο θα γίνει η υλοποίηση του κυκλώματος. Ειδικότερα, η σειρά σύνταξης ενός κυκλώματος κατά την συγγραφή του κώδικα στην VHDL είναι :

α) εισαγωγή βιβλιοθηκών

β) περιγραφή οντότητας

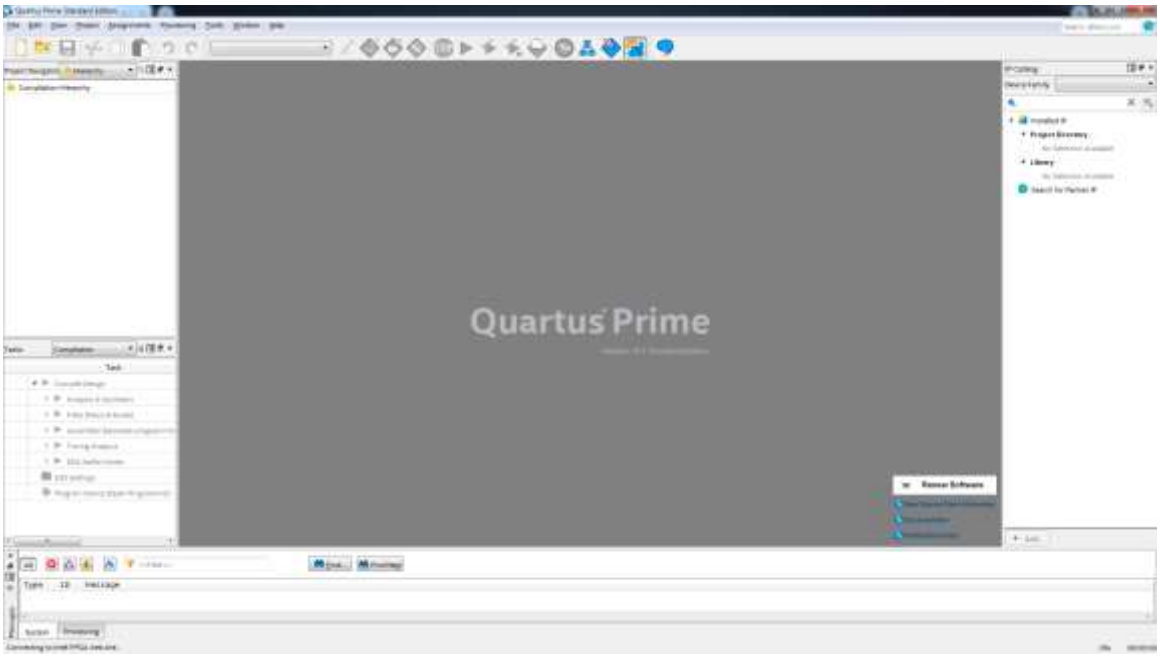
γ) αρχιτεκτονική της οντότητας

2.6 Το λογισμικό Quartus

Το περιβάλλον λογισμικού Quartus, χρησιμοποιήθηκε για την ανάπτυξη του κυκλώματος του φίλτρου Sobel, του κυκλώματος για την αποθήκευση των δεδομένων των γραμμών που βρίσκεται εκείνη την στιγμή η μάσκα του φίλτρου και τέλος του test bench, για τον έλεγχο λειτουργίας των κυκλωμάτων που σχεδιάστηκαν. Επιλέχθηκε η ανάπτυξη αυτών των κυκλωμάτων στο Quartus, λόγω του ότι το λογισμικό προσφέρει την δυνατότητα συγγραφής καθαρού κώδικα VHDL. Είναι δυνατό, το κύκλωμα που θα γραφεί στο Quartus σε VHDL, να μπορεί να χρησιμοποιηθεί αργότερα και από άλλα λογισμικά ανάπτυξης

κυκλωμάτων. Τα αρχεία που θα παραχθούν στο λογισμικό Quartus στην συνέχεια θα εξομοιωθούν με το λογισμικό ModelSim, με σκοπό τον έλεγχο της ορθότητας των αποτελεσμάτων, για τα οποία σχεδιάστηκε το κύκλωμα.

Το λογισμικό Quartus, αναπτύχθηκε αρχικά από την εταιρεία Altera και στη συνέχεια από την εταιρεία Intel. Το λογισμικό αυτό υποστηρίζει τον προγραμματισμό σε γλώσσα περιγραφής υλικού. Οι μηχανικοί, μέσω αυτού του λογισμικού, μπορούν να αναπτύξουν και να εξομοιώσουν κυκλώματα. Οι υποστηριζόμενες γλώσσες περιγραφής υλικού για το Quartus είναι η VHDL και η Verilog. Το λογισμικό Quartus, παρέχει την δυνατότητα συγγραφής κώδικα κυκλώματος, χωρίς να προσθέτει αναγκαστικά σε αυτόν επιπλέον πληροφορίες, όπως την αναπτυξιακή πλακέτα για την οποία προορίζεται ο κώδικας. Δηλαδή δίδει την δυνατότητα συγγραφής κυκλώματος με καθαρό κώδικα.



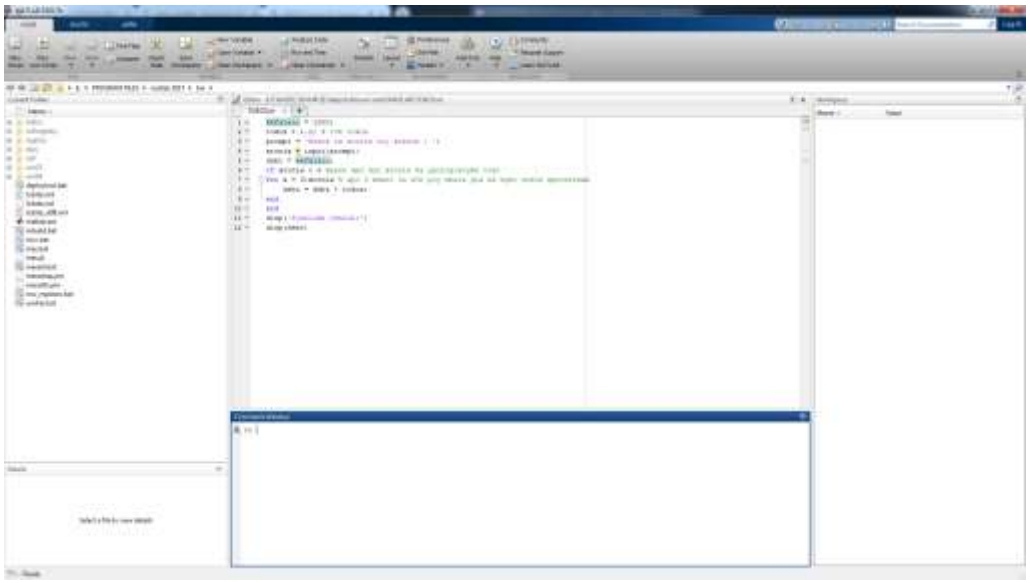
Εικόνα 2.6.1 Το παραθυρικό περιβάλλον εργασίας του Quartus

2.7 Το λογισμικό Matlab της εταιρείας Mathworks

Το λογισμικό Matlab της Mathworks, είναι χρήσιμο περιβάλλον προγραμματισμού για Μηχανικούς, με πολλές δυνατότητες, όπως επεξεργασία εικόνας, εκτέλεση πολύπλοκων μαθηματικών αλγορίθμων, επεξεργασία πινάκων και σχεδίαση κυκλωμάτων. Στο πακέτο του Matlab είναι ενσωματωμένο και το λογισμικό Simulink στο οποίο μπορούμε να εκτελέσουμε προσομοιώσεις. [46]

Το Matlab υποστηρίζει ανάλυση και τον σχεδιασμό συστημάτων. Για τον σχεδιασμό των συστημάτων, μπορεί να γίνει ανάπτυξη script με τη C-like γλώσσα του Matlab. Με το λογισμικό του Matlab μπορεί κάποιος να αναλύσει δεδομένα, να αναπτύξει αλγορίθμους και να δημιουργήσει μοντέλα και εφαρμογές. Το Matlab μπορεί να υλοποιήσει συναρτήσεις πραγματικές, μιγαδικές, πεπλεγμένες συναρτήσεις δύο μεταβλητών και άλλες συναρτήσεις. Το Matlab μπορεί τέλος να δώσει ιστογράμματα, τομεογράμματα, ραβδογράμματα, εμβαδογράμματα, και άλλα, τα οποία χρησιμοποιούνται στην στατιστική. [12]

Το λογισμικό Matlab, χρησιμοποιήθηκε για την μετατροπή της αρχικής εικόνας σε αρχείο δεδομένων dat, αποθηκεύοντας σε αυτό όλες τις τιμές των pixel της εικόνας, σε ένα μονοδιάστατο πίνακα, με τιμές από μηδέν (0), έως διακόσια πενήντα πέντε (255). Το αρχείο των δεδομένων, στην συνέχεια, θα χρησιμοποιηθεί για την εξομοίωση του κυκλώματος. Από την εξομοίωση θα προκύψει ένα νέο αρχείο δεδομένων dat. Το νέο αυτό αρχείο περιέχει τα δεδομένα της εικόνας ακμών, την οποία θα εμφανίσουμε μέσω ενός νέου κατάλληλου κώδικα, που θα γραφεί και αυτός στο λογισμικό Matlab.



Εικόνα 2.7.1 Το παραθυρικό περιβάλλον εργασίας του Matlab

2.8 Το λογισμικό ModelSim

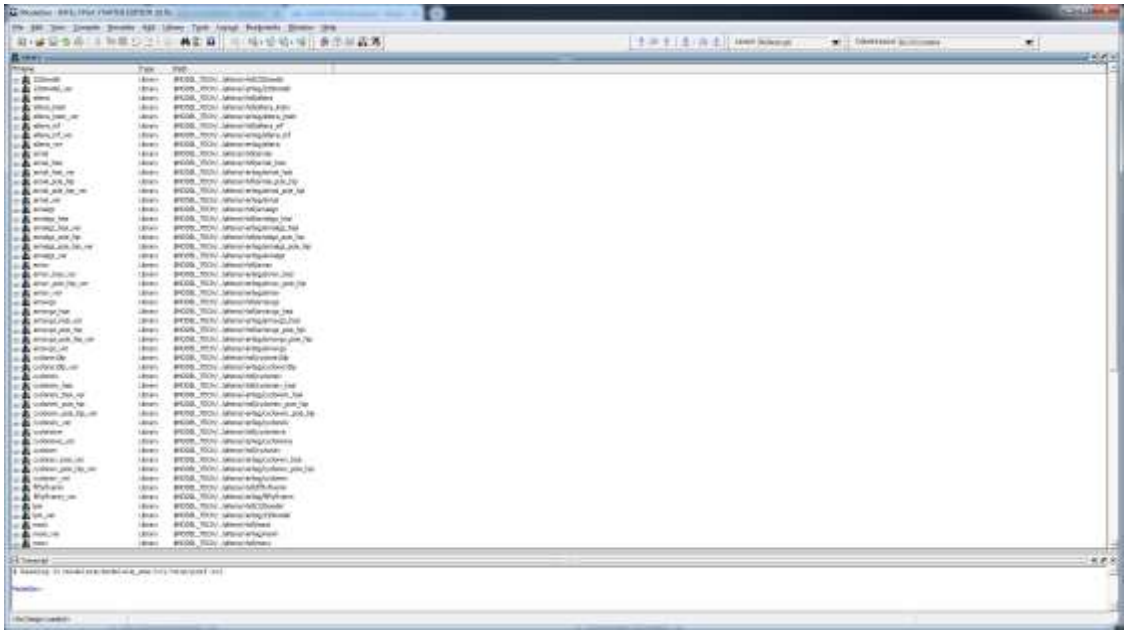
Το Modelsim είναι ένα λογισμικό της εταιρείας Intel. Μέσω του λογισμικού αυτού μπορούμε να εξομοιώσουμε ένα κύκλωμα (με σκοπό τον έλεγχο της λειτουργίας του), που

αναπτύξαμε, σε γλώσσα προγραμματισμού περιγραφής κυκλωμάτων, όπως την γλώσσα VHDL και την γλώσσα Verilog. Το λογισμικό αυτό, μπορεί να μας δώσει την συμπεριφορά των σχεδιαζόμενων κυκλωμάτων, για ένα FPGA, καθώς και το αποτέλεσμα που προκύπτει από την λειτουργία του σχεδιασμένου κυκλώματος, μέσω της προσομοίωσης αυτού. [13]

Μερικές από τις δυνατότητες του Modelsim είναι :

- Υποστήριξη σχεδιασμού σε όλες τις συσκευές που φέρουν FPGA
- Υποστήριξη σε γλώσσες προγραμματισμού περιγραφής υλικού
- Υποστήριξη για κυκλώματα περιγραφής υλικού μικρά αλλά και μεγάλα με όριο της 10000 γραμμές κώδικα περιγραφής υλικού (ειδικά για την έκδοση Starter Edition)
- Υποστήριξη σε κυκλώματα τα οποία έχουν σχεδιαστεί με συνδυασμό διπλής γλώσσας προγραμματισμού περιγραφής υλικού, VHDL και Verilog.

Το Modelsim είναι ένα λογισμικό, το οποίο προσφέρει πολλές δυνατότητες στους μηχανικούς σχεδιασμού κυκλωμάτων με τις λειτουργίες του. Επίσης προσφέρει πολλά στην ανάπτυξη ενός κυκλώματος, μέσω του χρόνου που εξοικονομείται από την προσομοίωση του. Μέσω τον αποτελεσμάτων της προσομοίωσης ο μηχανικός μπορεί να εξάγει ασφαλή συμπεράσματα, για την απόδοση του κυκλώματος και την ορθότητα των αποτελεσμάτων, για τα οποία σχεδιάστηκε το κύκλωμα.



Εικόνα 2.8.1 Το παραθυρικό περιβάλλον εργασίας του ModelSim

2.9 Το λογισμικό Vivado

Το λογισμικό Vivado ανήκει στην εταιρεία Xilinx και σ' αυτό μπορούμε να σχεδιάζουμε και να εξομοιώνουμε κυκλώματα. [9] Με το λογισμικό Vivado μπορούμε να γράψουμε απλή VHDL, αλλά κυρίως χρησιμοποιείται για να δημιουργήσουμε συστήματα σε τσιπ (SoC) τα οποία αποτελούνται από βαθμίδες πνευματικής περιουσίας ή αλλιώς IPs. Το λογισμικό Vivado, για την ανάπτυξη των συστημάτων, βασίζεται σε ανοιχτό περιβάλλον βιομηχανικών στάνταρ, όπως το AMBA ®, AXI4, IP-XACT. Πολλές βαθμίδες στις βιβλιοθήκες του Vivado στηρίζονται στα παραπάνω πρωτόκολλα και μπορούν να χρησιμοποιηθούν στις δικές μας σχεδιάσεις. [10]

Για την σχεδίαση ενός συστήματος στο Vivado, μπορούμε να χρησιμοποιήσουμε τρεις τρόπους. Ο πρώτος τρόπος είναι να γραφεί κώδικας σε γλώσσα VHDL ή Verilog και στην συνέχεια να γίνει η σύνθεση του συστήματος. Ο δεύτερος τρόπος είναι να μεταβούμε στην λειτουργία create block design (Δημιουργία διαγράμματος βαθμίδων) και να σχεδιάσουμε εκεί το σύστημα με έτοιμα block RTL, από την υπάρχουσα βιβλιοθήκη του προγράμματος ή από άλλες βιβλιοθήκες που θα βρούμε στο διαδίκτυο.

Ο τρίτος τρόπος είναι ο συνδυασμός των δύο προηγούμενων τρόπων. Δηλαδή η σχεδίαση ενός μέρος του συστήματος με την συγγραφή του σε κώδικα και το υπόλοιπο μέρος

του συστήματος, να γίνει στο create block design από τις έτοιμες βιβλιοθήκες που υπάρχουν εκεί. Στον τρίτο τρόπο, τα συστήματα που σχεδιάσθηκαν με τον κώδικα, θα εισαχθούν στο block design ως RTL blocks και στην συνέχεια θα δημιουργηθούν οι κατάλληλες συνδέσεις επικοινωνίας, για όλα τα μέρη του συστήματος.

Το πρόγραμμα Vivado, χρησιμοποιήθηκε στην παρούσα διπλωματική, για την δημιουργία του διαγράμματος βαθμίδων (block diagram) του κυκλώματος ανίχνευσης ακμών. Ο λόγος επιλογής του Vivado, είναι ότι η αναπτυξιακή πλακέτα PYNQ-Z1 που χρησιμοποιήθηκε για την υλοποίηση του κυκλώματος ακμών, μπορεί να συνεργασθεί μόνο με το πρόγραμμα Vivado.

Κεφάλαιο 3. Ανίχνευση ακμών

3.1 Τεχνολογίες ανίχνευσης ακμών

Η ανίχνευση των ακμών είναι μια τεχνική που χρησιμοποιείται στην επεξεργασία εικόνας με σκοπό να αναγνωρίζει τις απότομες αλλαγές χρώματος στην εικόνα. [29] Για τον σκοπό αυτό εφαρμόζουμε τους ανιχνευτές ακμών. Κάποιοι βασικοί ανιχνευτές ακμών είναι ο Canny edge detector, ο Prewitt edge detector και ο Sobel edge detector. [30] Και οι τρεις αυτοί τύποι ανιχνευτών ακμών βασίζονται σε μαθηματικούς τύπους, οι οποίοι εφαρμόζονται σε μια περιοχή παραθύρου, το οποίο ολισθαίνει επάνω στις οριζόντιες γραμμές των pixel της εικόνας, διαδοχικά. Ενδεικτικά, ένα συνηθισμένο μέγεθος παραθύρου στην ανίχνευση ακμών είναι 3x3.

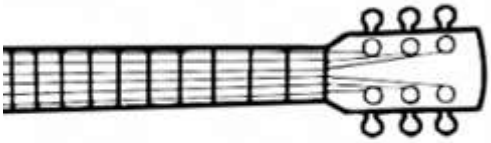
Ο τρόπος που θα επιλέξουμε για να εφαρμόσουμε την ανίχνευση ακμών εξαρτάται από τον χρόνο εξαγωγής του αποτελέσματος του συστήματος που θέλουμε να πετύχουμε. Ποιο συγκεκριμένα, εάν θέλουμε για παράδειγμα να σχεδιάσουμε έναν ανιχνευτή ακμών για ένα αυτοκινούμενο όχημα σε συνθήκες πραγματικές, τότε ο χρόνος εξαγωγής της εικόνας ακμών, είναι πολύ σημαντικό να είναι όσο το δυνατόν μικρότερος. Στην περίπτωση, όμως, που θέλουμε να εξάγουμε εικόνα ακμών σε μια φωτογραφία που τραβήχτηκε από ένα κινητό τηλέφωνο, ο χρόνος που απαιτείται για την εξαγωγή της εικόνας ακμών μπορεί να είναι και μεγαλύτερος σε σχέση με το προηγούμενο παράδειγμα.

Για την υλοποίηση της διπλωματικής εργασίας, επιλέχθηκε η χρήση του φίλτρου Sobel για την εξαγωγή της εικόνας ακμών, καθώς αυτό είναι πιο σύνθετο στην υλοποίηση του, σε σχέση με το φίλτρο Prewitt, αλλά πιο απλό στην υλοποίηση του σε σχέση με το φίλτρο Canny. Όσον αφορά την ποιότητα της παραγόμενης εικόνας ακμών, η σειρά κατάταξης των τριών αυτών φίλτρων από το καλύτερο προς το ικανοποιητικό, είναι πρώτο το φίλτρο Canny, δεύτερο το φίλτρο Sobel και τρίτο το φίλτρο Prewitt. Η σειρά κατάταξης αυτή συμφωνεί με την σειρά κατάταξης, ως προς την πολυπλοκότητα στους υπολογισμούς που απαιτούνται για την παραγόμενη εικόνα ακμών για κάθε φίλτρο.

3.2 Η ανίχνευση ακμών

Η ανίχνευση των ακμών γενικότερα, εφαρμόζεται στις εικόνες με σκοπό την εξαγωγή χρήσιμων πληροφοριών από αυτές. Η ανίχνευση των ακμών επικεντρώνεται στην αναγνώριση ξαφνικών αλλαγών, με την μορφή ασυνέχειας σε μια εικόνα. [5]

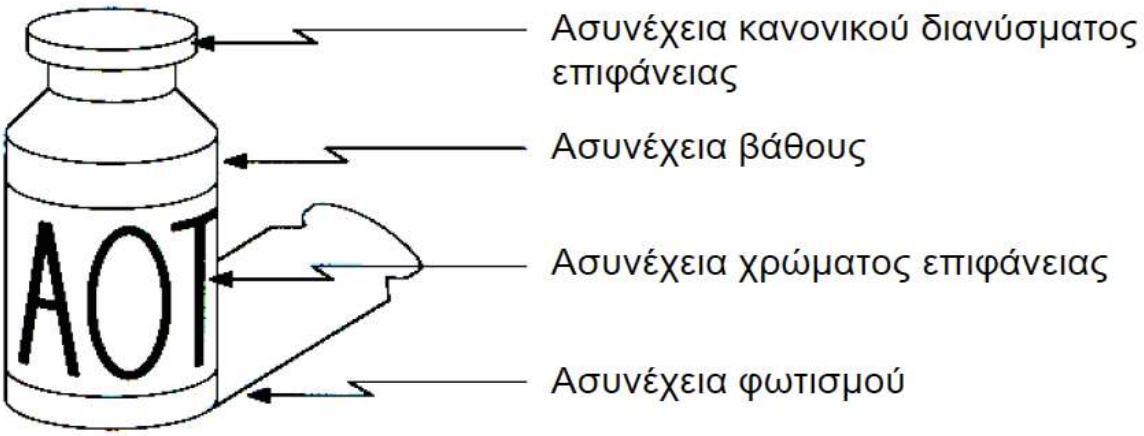
Είναι γνωστό ότι διαισθητικά ο άνθρωπος λαμβάνει τις περισσότερες πληροφορίες για ένα αντικείμενο ή ένα σχήμα, από τις ακμές του.



Εικόνα 3.2.1 Γραμμικό σχέδιο μέρους κιθάρας

Στην εικόνα 3.2.1 φαίνεται ένα απλό γραμμικό σχέδιο μέρους κιθάρας. Από το σχέδιο αυτό, γίνεται αντιληπτό το αντικείμενο που απεικονίζεται, το μπράτσο μαζί με τις χορδές μιας κιθάρας. Στην εικόνα βλέπουμε ότι έχουν σχεδιασθεί μόνον οι κύριες γραμμές χωρίς χρωματισμό, των μερών της κιθάρας. Παρόλα αυτά, μόνο από τη σχεδίαση των περιγραμμάτων, γίνεται κατανοητό το τι απεικονίζει το σχήμα αυτό. Η πληροφορία του χρώματος που απουσιάζει από αυτήν την εικόνα, δεν αποτελεί εμπόδιο για την κατανόηση του σχήματος.

Οι ακμές στα αντικείμενα που μας περιβάλλουν δημιουργούνται από διάφορους παράγοντες, όπως μια ασυνέχεια βάθους, μια ασυνέχεια χρώματος, μια ασυνέχεια φωτισμού, ή τέλος μια ασυνέχεια κανονικού διανύσματος (εικόνα 3.2.2).



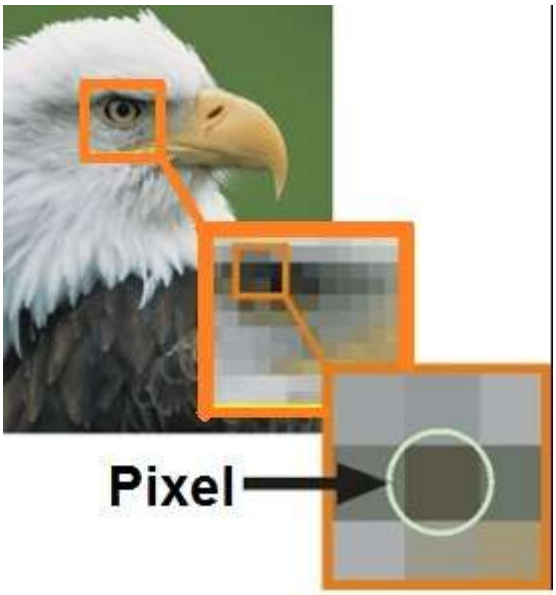
Εικόνα 3.2.2 Γραμμικό σχέδιο αντικειμένου, με επεξήγηση των εξαγόμενων ακμών από αυτό

Η ανίχνευση και στην συνέχεια η εξαγωγή των ακμών σε μια εικόνα είναι σημαντική, διότι βοηθάει έτσι στην ταχύτερη αναγνώριση των αντικειμένων από τον ηλεκτρονικό υπολογιστή. Η ανίχνευση των ακμών προσφέρει μια παραγόμενη εικόνα, η οποία περιέχει μικρότερο αριθμό πληροφοριών - δεδομένων, με αποτέλεσμα την ταχύτερη επεξεργασία τους. Οι πληροφορίες που περιέχει η εξαγόμενη εικόνα ακμών, όπως προαναφέρθηκε, είναι αυτές οι οποίες βοηθούν στην αναγνώριση ενός αντικειμένου, μέσω της σύγκρισης τους με άλλες εικόνες από την βάση δεδομένων του συστήματος, με σκοπό συνήθως την λήψη αποφάσεων από το σύστημα, όπως για το τι αντικείμενο απεικονίζεται. Μια χαρακτηριστική εφαρμογή λήψης αποφάσεων είναι η κίνηση ενός αυτόνομου οχήματος σε δυναμικό περιβάλλον, με τη χρήση καμερών. Στην περίπτωση αυτή του παραδείγματος, η επεξεργασία της εικόνας από τις κάμερες επιτρέπει στο σύστημα την αναγνώριση των αντικειμένων, όσο το όχημα κινείται στο δυναμικό περιβάλλον.

3.3 Χαρακτηριστικά της ψηφιακής εικόνας

Η εικόνα που λαμβάνεται από έναν ηλεκτρονικό υπολογιστή, μπορεί να είναι σε αναλογική μορφή (με την μορφή σήματος) ή σε ψηφιακή μορφή (με την μορφή δεδομένων). Η ψηφιακή εικόνα, μέσα στον υπολογιστή, μετατρέπεται σε ένα πίνακα με μήκος x και ύψος

y. Το κάθε στοιχείο του πίνακα έχει πληροφορίες αποθηκευμένες για τα τρία βασικά χρώματα, κόκκινο (R), πράσινο (G) και μπλε (B). Οι πληροφορίες για το κάθε χρώμα δίδουν και τον χρωματισμό του κάθε στοιχείου του πίνακα ή αλλιώς του κάθε pixel της εικόνας. Οι αποθηκευμένες πληροφορίες σε κάθε στοιχείο του πίνακα γενικότερα, χαρακτηρίζουν και ένα χρώμα από την χρωματική παλέτα του κόκκινου (R), του πράσινου (G) και τέλος του μπλε (B). Έτσι, ανάλογα με την απόχρωση, δημιουργούνται οι πληροφορίες για τα χρώματα της εικόνας από τα οποία αποτελείται αυτή. Η εισερχόμενη ψηφιακή εικόνα έχει συγκεκριμένες διαστάσεις, οι οποίες μετρώνται σε εικονοστοιχεία (pixels) και χαρακτηρίζουν τις τελικές διαστάσεις του πίνακα που σχηματίζεται. Έτσι, για παράδειγμα, εάν έχουμε μια έγχρωμη εικόνα διαστάσεων 800 x 600 pixel, τότε ο πίνακας με μήκος x και ύψος y που θα δημιουργηθεί για αυτήν, θα έχει διαστάσεις μήκος για τον άξονα x 800 pixel και ύψος για τον άξονα y 600 pixel.



Εικόνα 3.3.1 Λεπτομέρεια εικονοστοιχείου (pixel)

Στην εικόνα 3.3.1, διακρίνουμε την χρωματική πληροφορία που περιέχει ένα εικονοστοιχείο (pixel) της εικόνας. Το εικονοστοιχείο (pixel) αυτό έχει βάθος χρώματος 24 bit, λόγω του ότι μέσα σε αυτό περιέχεται χρωματική πληροφορία RGB. Κάθε απόχρωση περιέχει πληροφορία 8 bit και έτσι και οι τρεις αποχρώσεις μαζί δίνουν πληροφορία μεγέθους 24 bit στο κάθε εικονοστοιχείο (pixel).

Τα χαρακτηριστικά μιας ψηφιακής εικόνας είναι η ανάλυσή της (resolution), οι πληροφορίες για τα χρωματισμό του κάθε εικονοστοιχείου (pixel) και τέλος το εύρος σε bit των εικονοστοιχείων (pixels) που περιέχει. Αυτά τα χαρακτηριστικά της εικόνας λαμβάνονται υπόψη για την επεξεργασία της σε πραγματικό χρόνο. Για παράδειγμα, όταν θέλουμε να επεξεργασθούμε μια εικόνα για την εξαγωγή από αυτή των ακμών της, θα πρέπει να ελέγξουμε τα δεδομένα εισόδου της εικόνας, εάν είναι σε έγχρωμη μορφή. Στην περίπτωση αυτήν θα πρέπει πρώτα να μετατρέψουμε την εικόνα σε εικόνα κλίμακας του γκρι, με πληροφορία ανά εικονοστοιχείο μεγέθους 8 bit και έπειτα να γίνει η επεξεργασία της.

3.4 Το φίλτρο ανίχνευσης ακμών Prewitt

Ο ανιχνευτής ακμών Prewitt, χρησιμοποιείται στην επεξεργασία εικόνας, ως μέρος αλγορίθμων ανίχνευσης ακμών εικόνας. Το φίλτρο χρησιμοποιεί προσέγγιση για την διαφοροποίηση, στην χρωματική ένταση της εικόνας. Το φίλτρο αυτό, κατά την εφαρμογή του σε μια εικόνα, δαπανά λιγότερους πόρους σε υπολογισμούς σε σύγκριση με τα φίλτρα Sobel και Kayyali.

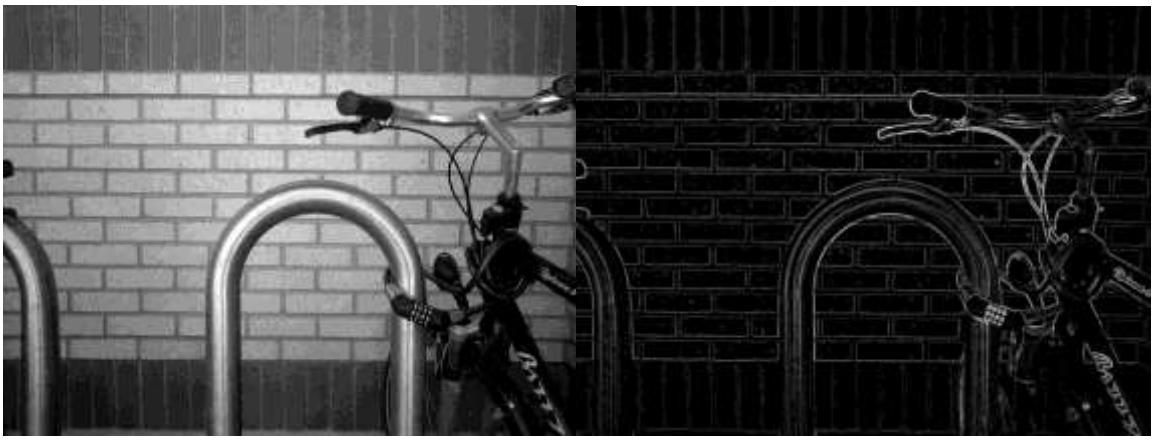
Η εφαρμογή του φίλτρου στην εικόνα γίνεται σε μια περιοχή 3 X 3, η οποία ολισθαίνει επάνω στην εικόνα και εκτελεί τους υπολογισμούς. Οι πράξεις που εκτελούνται στην περιοχή αυτήν είναι οι πράξεις από την εφαρμογή των ακόλουθων (εικόνα 3.4.1) πινάκων του φίλτρου.

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{A}$$

Εικόνα 3.4.1 Οι απαιτούμενοι υπολογισμοί για την υλοποίηση του φίλτρου Prewitt

Πηγή : https://en.wikipedia.org/wiki/Prewitt_operator

Τα αποτελέσματα των υπολογισμών είναι μια εικόνα με μαύρο φόντο, που επάνω σε αυτήν εμφανίζονται με λευκές γραμμές τα περιγράμματα των αντικειμένων που απεικονίζονται σε αυτήν. [28]



Εικόνα 3.4.2 Παράδειγμα εφαρμογής του φίλτρου Prewitt

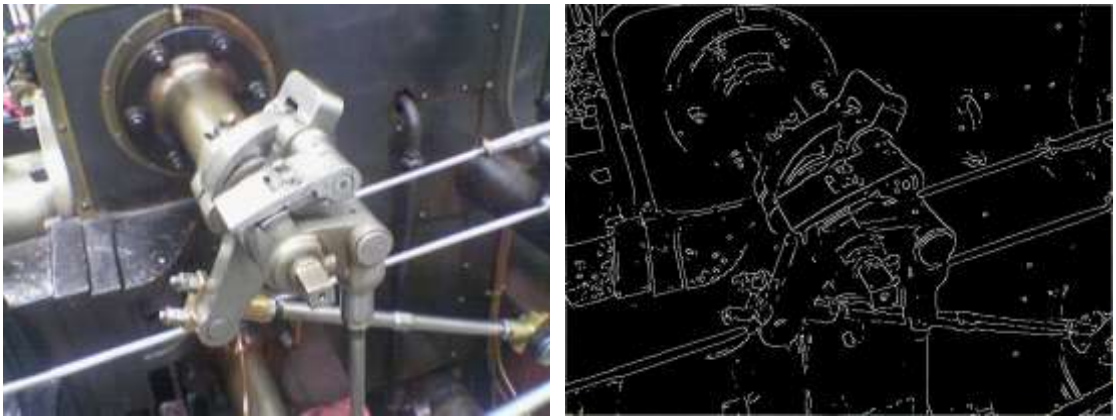
Πηγή : https://en.wikipedia.org/wiki/Prewitt_operator

3.5 Το φίλτρο ανίχνευσης ακμών Canny

Ο ανιχνευτής ακμών Canny, χρησιμοποιεί έναν πολυεπίπεδο αλγόριθμο, για να βρει ακμές σε εικόνες. Έχει χρησιμοποιηθεί ευρέως σε εφαρμογές και συστήματα ρομποτικής όρασης. Το φίλτρο αυτό χρησιμοποιεί τον υπολογισμό των μεταβλητών. Η τεχνική αυτήν βρίσκει μια συνάρτηση, η οποία βελτιστοποιεί μια δεδομένη συνάρτηση. Η λειτουργία του φίλτρου μπορεί να περιγραφεί με το άθροισμα των τεσσάρων εκθετικών όρων ή να προσεγγισθεί με την πρώτη παράγωγο της Gaussian. Το φίλτρο Canny, παρέχει αξιόπιστη ανίχνευση ακμών, λόγω των αυστηρά καθορισμένων μεθόδων που χρησιμοποιεί.

Η εκτέλεση του φίλτρου γίνεται στα πέντε ακόλουθα βήματα.

1) Εφαρμογή του Gaussian φίλτρου, για την εξαγωγή του θορύβου από την εικόνα
2) Εύρεση των παραγώγων της εικόνας
3) Εφαρμογή κατωφλίωσης, για την απαλλαγή από την ψευδή ανίχνευση των ακμών
4) Εφαρμογή διπλής κατωφλίωσης, για τον προσδιορισμό πιθανών ακμών
5) Ανίχνευση ακμών με την μέθοδο της υστέρησης, ολοκλήρωση της ανίχνευσης ακμών



Εικόνα 3.5.1 Παράδειγμα εφαρμογής του φίλτρου Canny

Πηγή : https://en.wikipedia.org/wiki/Canny_edge_detector

Οι υπολογισμοί που εκτελούνται σε αυτό το φίλτρο είναι αρκετά πολύπλοκοι. Για παράδειγμα, ο τύπος του Gaussian φίλτρου, για την εφαρμογή του βήματος 1 είναι ο ακόλουθος

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2} \right); 1 \leq i, j \leq (2k+1)$$

Ένα παράδειγμα εφαρμογής του τύπου αυτού, για περιοχή εικόνας 5x5, είναι ο ακόλουθος τύπος

$$\mathbf{B} = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

Μετά την αποθορυβοποίηση με το φίλτρο Gauss, υπολογίζεται η κλίση της εικόνας, μέσω των οριζόντιων και κάθετων παραγώγων G_x, G_y.

Αφού βρεθεί η κλίση κατά την οριζόντια και την κάθετη διάσταση, σε κάθε εικονοστοιχείο, οι τύποι για το μέτρο και την κλίση της παραγώγου της εικόνας δίνονται ως εξής:

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$
$$\mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x),$$

Στη συνέχεια, η κατωφλίωση γίνεται με βάση τους παραπάνω υπολογισμούς.

3.6 Το φίλτρο ανίχνευσης ακμών Sobel

Ο ανιχνευτής ακμών έχει ως σκοπό την εξαγωγή των βασικών χαρακτηριστικών της εικόνας με την παραγόμενη εικόνα να εμφανίζει το περίγραμμα των αντικειμένων της αρχικής εικόνας. Το φίλτρο Sobel χρησιμοποιείται στην επεξεργασία εικόνας και την ρομποτική όραση. Η εφαρμογή του φίλτρου επιτυγχάνεται με κατάλληλους αλγόριθμους οι οποίοι αποδίδουν τις ακμές των αντικειμένων. Η δημιουργοί του φίλτρου Sobel είναι ο Irwin Sobel και ο Gary Feldman [23]. Η γενική ιδέα του φίλτρου είναι να δημιουργηθεί ένα παράθυρο 3x3, το οποίο θα ολισθαίνει επάνω στα pixel της εικόνας, με σκοπό να παράγεται η εικόνα των ακμών μέσω των κατάλληλων μαθηματικών πράξεων που θα εκτελούνται. Για τις μαθηματικές πράξεις που απαιτούνται εφαρμόζονται οι ακόλουθοι πίνακες (εικόνα 3.6.1).

Υπολογισμοί για τον πίνακα τιμών X

X – Direction Kernel

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

Υπολογισμοί για τον πίνακα τιμών Y

Y – Direction Kernel

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

Εικόνα 3.6.1 Τιμές μεταβλητών για τους απαιτούμενους υπολογισμούς του φίλτρου Sobel

Ο πίνακας Gx εφαρμόζεται για τις οριζόντιες γραμμές των pixel της εικόνας και ο πίνακας Gy εφαρμόζεται για τις κάθετες γραμμές των pixel της εικόνας. Για το τελικό αποτέλεσμα και την εξαγωγή των ακμών θα προστεθούν μεταξύ τους οι απόλυτες τιμές που προκύπτουν από τους δύο αυτούς πίνακες.

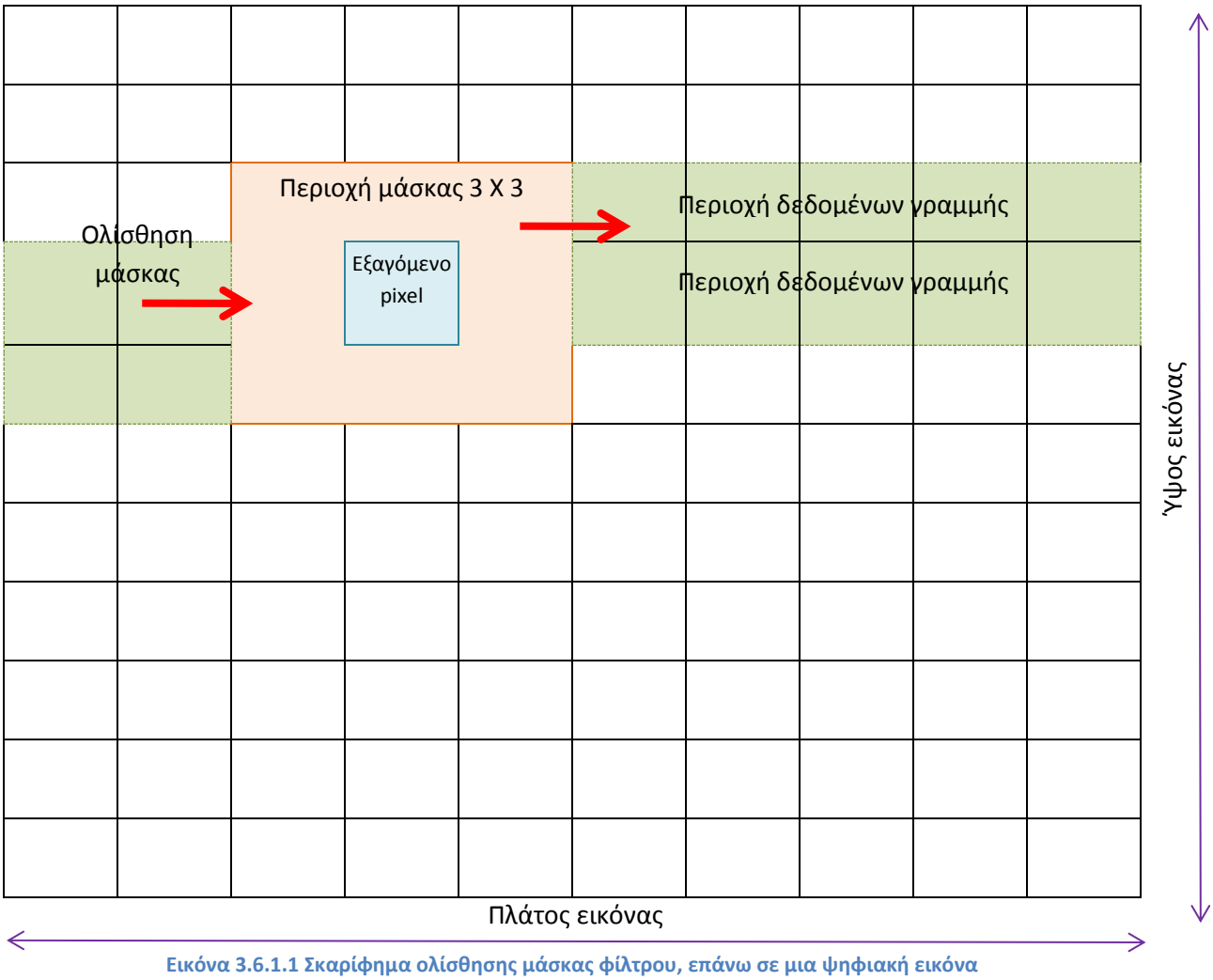
$$|G| = |Gx| + |Gy|$$

Η νέα παραγόμενη εικόνα είναι μικρότερη κατά δύο pixel της αρχικής εικόνας, στην οριζόντια διάσταση και στην κατακόρυφη διάσταση. Η νέα εικόνα αποδίδει τις ακμές των αντικειμένων με λευκή γραμμή σε μαύρο φόντο, ώστε να είναι ευδιάκριτες οι ακμές που έχουν παραχθεί.

3.6.1 Τρόπος υλοποίησης ανίχνευσης ακμών με το φίλτρο Sobel

Το φίλτρο Sobel, ως γνωστόν, εφαρμόζεται με τη βοήθεια ενός μικρού παραθύρου διάστασης 3x3, το οποίο ολισθαίνει οριζόντια επάνω στα pixel της εικόνας. Συγκεκριμένα, το παράθυρο αρχίζει να ολισθαίνει από την επάνω αριστερή γωνία του πίνακα δεδομένων της εικόνας και καταλήγει στο κάτω δεξί μέρος αυτής. Το παράθυρο αυτό, με τους συντελεστές που ορίζονται σε κάθε θέση, λέγεται αλλιώς και μάσκα. Έτσι, ολισθαίνει διαδοχικά σε όλες τις σειρές δεδομένων της εικόνας. Το παράθυρο κατά την ολίσθηση του επάνω στα pixel, εκτελεί μαθηματικές πράξεις, οι οποίες εξάγουν και την εικόνα ανίχνευσης ακμών. Ένα σκαρίφημα της ολίσθησης του παραθύρου επάνω στα pixel της εικόνας, φαίνεται στην εικόνα 3.6.1.1.

Για να είναι εφικτό να πραγματοποιηθούν οι μαθηματικές πράξεις όμως, θα πρέπει να αποθηκεύονται όλα τα δεδομένα που περιέχονται στις τρεις γραμμές, επάνω στις οποίες ολισθαίνει το παράθυρο. Τα δεδομένα αυτά θα πρέπει στην συνέχεια να είναι προσπελάσιμα, όταν το παράθυρο μετακινηθεί μια θέση δεξιά. Όταν τώρα το παράθυρο μεταβεί στις επόμενες τρεις γραμμές, θα πρέπει να αποθηκεύονται τα δεδομένα των επόμενων αυτών γραμμών. Επομένως χρειαζόμαστε για την εφαρμογή, ένα καταχωρητή ολίσθησης, ο οποίος θα είναι επιφορτισμένος με την αποθήκευση των δεδομένων των γραμμών αυτών.



Εικόνα 3.6.1.1 Σκαρίφημα ολίσθησης μάσκας φίλτρου, επάνω σε μια ψηφιακή εικόνα

Για τον υπολογισμό και την εξαγωγή της εικόνας ακμών, εκτελούνται δύο ξεχωριστοί υπολογισμοί στην περιοχή του παραθύρου, που στην συνέχεια τα αποτελέσματα τους προστίθενται με την μορφή απολύτων τιμών και τέλος το αποτέλεσμα συγκρίνεται με ένα νούμερο (κατώφλι). Κατά την σύγκριση, εάν το αποτέλεσμα είναι μεγαλύτερο από το συγκρινόμενο νούμερο, τότε εξάγεται ένα pixel το οποίο περιέχει τον δυαδικό αριθμό 11111111, μεγέθους 8 bit, δηλαδή ένα pixel χρώματος άσπρου. Στην αντίθετη περίπτωση, όταν δηλαδή το αποτέλεσμα είναι μικρότερο από τον συγκρινόμενο αριθμό, τότε εξάγεται ένα pixel χωρίς χρώμα μέσα του, μαύρο δηλαδή με δυαδικό αριθμό 00000000, μεγέθους 8 bit.

Όπως προαναφέρθηκε για τις μαθηματικές πράξεις απαιτούνται οι πράξεις που δείχνονται στου πίνακες των εικόνων 3.6.1.2 και 3.6.1.3που ακολουθύν. Οι πίνακες αυτοί μας δείχνουν τι πράξεις εκτελούνται, για το κάθε pixel του παραθύρου ξεχωριστά. Έτσι για

παράδειγμα για τον άξονα X, ο πίνακας μας δείχνει τι πράξεις θα γίνουν στα pixel της εικόνας. Στην εικόνα 3.6.1.2, φαίνονται οι απαραίτητες πράξεις για τον υπολογισμό του άξονα X. Όπου d1 έως και d9, είναι οι συντελεστές που περιέχονται στο pixel του παραθύρου ολίσθησης.

Πίνακας υπολογισμού άξονα X

-1 x d1	0 x d2	1 x d3
-2 x d4	0 x d5	2 x d6
-1 x d6	0 x d7	1 x d8

Εικόνα 3.6.1.2 Οι απαιτούμενες πράξεις για τον υπολογισμό του πίνακα X

Ομοίως για τον άξονα Y οι πράξεις που πρέπει να γίνουν φαίνονται στην εικόνα 3.6.1.3.

Πίνακας υπολογισμού άξονα Y

-1 x d1	-2 x d2	-1 x d3
0 x d4	0 x d5	0 x d6
1 x d6	2 x d7	1 x d8

Εικόνα 3.6.1.3 Οι απαιτούμενες πράξεις για τον υπολογισμό του πίνακα Y

Αφού εκτελεσθούν οι πράξεις των πινάκων X και Y ξεχωριστά, στην συνέχεια εξάγεται το αποτέλεσμα, από την πρόσθεση των απολυτών τιμών τους. Το αποτέλεσμα της πρόσθεσης αυτής συγκρίνεται με έναν αριθμό που ορίζουμε εμείς. Εάν το συγκρινόμενο αποτέλεσμα είναι μεγαλύτερο του ορισθέντα αριθμού, τότε το εξαγόμενο pixel θα λάβει την χρωματική τιμή του λευκού χρώματος, δηλαδή τιμή 11111111, σε σύστημα 8 bit. Σε αντίθετη περίπτωση το εξαγόμενο pixel θα λάβει την χρωματική τιμή του μαύρου χρώματος, δηλαδή τιμή 00000000, σε σύστημα 8 bit και πάλι.

Κεφάλαιο 4. Υλοποίηση συστήματος επιτάχυνσης της ανίχνευσης ακμών στο λογισμικό Vivado

4.1 Εισαγωγή

Η υλοποίηση του συστήματος επιτάχυνσης έγινε σε δύο στάδια. Στο πρώτο στάδιο της υλοποίησης, έγινε η ανάπτυξη του κώδικα του φίλτρου Sobel και η προσομοίωσή του στο λογισμικό Modelsim, με τη βοήθεια αποθηκευμένων εικόνων. Στο δεύτερο στάδιο, έγινε η σχεδίαση και ανάπτυξη του κατάλληλου κυκλώματος στο Vivado, για την εφαρμογή του φίλτρου Sobel στην προγραμματιζόμενη ψηφίδα του PYNQ-Z1.

4.2 Πρώτο στάδιο υλοποίησης

Στο πρώτο στάδιο υλοποίησης, η ανάπτυξη του κώδικα για το φίλτρο Sobel έγινε στο λογισμικό Quartus Prime. Για να γίνει η δοκιμή της λειτουργίας του φίλτρου Sobel που αναπτύχθηκε με τον κώδικα περιγραφής υλικού, έγινε η προσομοίωση αυτού στο Modelsim. Για την εκτέλεση της προσομοίωσης στο Modelsim, αναπτύχθηκε κατάλληλος κώδικας στο Quartus Prime, για την δημιουργία του αρχείου test bench, το οποίο θα οδηγήσει την προσομοίωση του φίλτρου Sobel. Για τον έλεγχο της ορθότητας λειτουργίας του φίλτρου Sobel κατά την προσομοίωση, χρησιμοποιήθηκε μια εικόνα για δοκιμή με διαστάσεις (πλάτος και ύψος) 1280 pixel x 720 pixel. Η εικόνα αυτή έπρεπε να μετατραπεί σε αρχείο μορφής .dat, πριν την εκτέλεση της προσομοίωσης στο Modelsim και στη συνέχεια, μετά το πέρας της προσομοίωσης να μετατραπεί εκ νέου από αρχείο μορφής dat σε εικόνα. Οι μετατροπές αυτές της εικόνας έγιναν μέσω του Matlab, με κώδικα που γράφηκε για αυτόν τον σκοπό.

4.2.1 Ο κώδικας των συστημάτων που δημιουργήθηκε στο Quartus

Για τον προγραμματισμό του φίλτρου Sobel σε γλώσσα VHDL, χρησιμοποιήθηκε το πρόγραμμα Quartus Prime. Επίσης, στο ίδιο πρόγραμμα φτιάχτηκε ο καταχωρητής ολίσθησης (shift register) και τέλος, το πρόγραμμα test bench για την διεξαγωγή της δοκιμής του

φίλτρου Sobel. Ο προγραμματισμός του φίλτρου Sobel στο Quartus Prime περιλαμβάνει τον ακόλουθο κώδικα.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity sobel is
	generic
		(
		 platos		: positive := 1280; -- platos eikonas
		 sobel_Threshold : integer  := 30;
		 word		: integer  := 8   -- ari8mos bit dedomenwn gia eisagwgh
		 );
	port
		(
		 datain	: in std_logic_vector (word -1 downto 0);
		 dataout	: out std_logic_vector (word -1 downto 0);
		 clock	: in std_logic;
		 pixl_clk_out : out std_logic;
		 reset_n	: in std_logic;
		 clken	: in std_logic
		);
end sobel;
architecture Behavioral of sobel is
component shift_line
generic
		(
		 depth : positive; -- platos eikonas
		 word  : integer   -- ari8mos bit dedomenwn gia eisagwgh
		 );
	port
		(
```

```
                shiftin   : in std_logic_vector (word -1 downto 0);
                shiftout  : out std_logic_vector (word -1 downto 0);
                clk       : in std_logic;
                resetn    : in std_logic
               );
end component;
signal data11 : std_logic_vector (word - 1 downto 0);
signal data12 : std_logic_vector (word - 1 downto 0);
signal data13 : std_logic_vector (word - 1 downto 0);
signal data21 : std_logic_vector (word - 1 downto 0);
signal data22 : std_logic_vector (word - 1 downto 0);
signal data23 : std_logic_vector (word - 1 downto 0);
signal data31 : std_logic_vector (word - 1 downto 0);
signal data32 : std_logic_vector (word - 1 downto 0);
signal data33 : std_logic_vector (word - 1 downto 0);
begin
buffer1:shift_line
generic map
        (
         depth => platos - 2,
         word => word
        )
port map
        (
         shiftin => data13,
         clk => clock,
         resetn => reset_n,
         shiftout => data21
        );
buffer2:shift_line
generic map
        (
```

```
        depth => platos - 2,
        word => word
       )
port map
       (
        shiftin => data23,
        clk => clock,
        resetn => reset_n,
        shiftout => data31
       );
p1: process(clock)

       variable Px    : integer;
       variable Py    : integer;
       variable total : integer;
       begin
       if rising_edge(clock) then
              if clken = '1' then
                     data12 <= data11;
                     data13 <= data12;
                     data22 <= data21;
                     data23 <= data22;
                     data32 <= data31;
                     data33 <= data32;
                -- pinakas X
                Px := abs(to_integer(unsigned(data23)) + to_integer(unsigned(data23)) +
to_integer(unsigned(data13)) + to_integer(unsigned(data33))-  (to_integer(unsigned(data21))
+ to_integer(unsigned(data21)) + to_integer(unsigned(data11)) +
to_integer(unsigned(data31))));
                -- pinakas Y
                Py := abs(to_integer(unsigned(data12)) + to_integer(unsigned(data12)) +
to_integer(unsigned(data11)) + to_integer(unsigned(data13)) - (to_integer(unsigned(data32))
```

```
+ to_integer(unsigned(data32)) + to_integer(unsigned(data31)) +
to_integer(unsigned(data33))));
                    -- Synolo pinakwn
                    total := Px + Py;
                  if (total > sobel_Threshold) then
                            dataout <= "11111111" ; -- didei timh 255
                  else
                            dataout <= "00000000" ; -- didei timh 0
                  end if;
                 end if;
          end if;
end process;
data11 <= datain;
pixl_clk_out <= clock;
end Behavioral;
```

Ο παραπάνω κώδικας χρησιμοποιεί ως πρότυπο τον κώδικα για ανίχνευση ακμών Prewitt που περιγράφεται στην παραπομπή [44].

Οι μεταβλητές platos, word και sobel_Threshold, έχουν δηλωθεί ως γενικές (generic), για να γίνεται η χρήση τους σε όλα τα τμήματα του κώδικα. Συγκεκριμένα, η μεταβλητή platos, αναφέρεται στο πλάτος της εικόνας που θα επεξεργασθούμε, η μεταβλητή sobel_Threshold δηλώνει το κατώφλι με το οποίο θα συγκριθεί το αποτέλεσμα των πράξεων, ώστε να παραχθεί η εικόνα ακμών και τέλος, η μεταβλητή word δηλώνει το μήκος της λέξης του εικονοστοιχείου σε bit, που σε αυτή την περίπτωση είναι 8 bit.

Όπως βλέπουμε από τον κώδικα που γράφηκε για το φίλτρο Sobel, ο καταχωρητής ολίσθησης είναι ενσωματωμένος στον προγραμματισμό του, με την αναφορά buffer1:shift_line και buffer2:shift_line, όπως φαίνεται στον κώδικα. Ο σκοπός είναι ο καταχωρητής ολίσθησης να συμβαδίζει με τις γραμμές στις οποίες βρίσκεται εκείνη την στιγμή το παράθυρο που ολισθαίνει για να αποθηκεύει τα δεδομένα και στη συνέχεια τα δεδομένα αυτά να ανακαλούνται από το φίλτρο Sobel, με σκοπό τον υπολογισμό των ακμών της εικόνας.

Ο κώδικας του καταχωρητή μνήμης που γράφηκε στο Quartus, είναι ο ακόλουθος. [44]

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity shift_line is
GENERIC
(
   depth : POSITIVE; --depth of delay line, nominally equal to image line
   word  : NATURAL
); -- bit width of shift register, nominally 8-bit
Port
(
   shiftin     : in STD_LOGIC_VECTOR(word - 1 downto 0);
   clk, resetn : in STD_LOGIC;
   shiftout    : out STD_LOGIC_VECTOR(word - 1 downto 0)
);
end shift_line;
architecture Behavioral of shift_line is
SUBTYPE sample IS STD_LOGIC_VECTOR(word - 1 downto 0);
TYPE OneD IS ARRAY(NATURAL RANGE <>) OF sample;
begin
PROCESS(clk, Resetn)
VARIABLE q : OneD(0 TO depth - 1);--Array q holds the outputs of FFs
BEGIN
IF Resetn='0' THEN
   gen0:          FOR i IN 0 TO depth - 1 LOOP
         q(i):= (OTHERS=>'0');
            END LOOP;
ELSIF clk' EVENT AND clk='1' THEN
   gen1:          FOR i IN depth - 1 DOWNTO 1 LOOP --see paragraph 6.9
         q(i):= q(i - 1);
               END LOOP;
```

```
        q(0):= shiftin;
END IF;
    shiftout <= q (depth - 1);
END PROCESS;
end Behavioral;
```

Όπως φαίνεται στον κώδικα, ο καταχωρητής ολίσθησης δημιουργεί έναν πίνακα (array), με σκοπό να αποθηκεύσει σε αυτόν, τα δεδομένα όλων των επόμενων pixels των γραμμών, στις οποίες βρίσκεται εκείνη την στιγμή το παράθυρο 3 x 3, του φίλτρου Sobel. Στην συνέχεια όταν το παράθυρο θα ολισθήσει μια θέση δεξιά, ο καταχωρητής μνήμης θα παράγει τα δεδομένα των επόμενων pixel, επάνω στο οποία βρίσκεται το παράθυρο, ώστε αυτά να υπολογιστούν από το φίλτρο Sobel. Όταν το παράθυρο 3 x 3, του φίλτρου Sobel, φτάσει στα τελευταία pixel στα δεξιά της εικόνας, τότε θα μετακινηθεί στις επόμενες γραμμές, και ο καταχωρητής μνήμης θα αποθηκεύσει τότε ξανά τα δεδομένα όλων των επόμενων pixel, των νέων γραμμών.

Τέλος γράφτηκε στο Quartus Prime, το αρχείο testbench, μέσω του οποίου θα γίνει η δοκιμή του αρχείου Sobel στην συνέχεια, στον εξομοιωτή ModelSim. Ο κώδικας που γράφηκε για το αρχείο test bench είναι ο ακόλουθος. [45]

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
----------------------------
entity sobel_tb is
end entity;
----------------------------
architecture sobel_tb_Arch of sobel_tb is
constant platos : POSITIVE := 1280;
component sobel is
generic
(
```

```
        row_width: POSITIVE:= 1280
);
port
(
    datain : in STD_LOGIC_VECTOR (7 downto 0);
    dataout : out STD_LOGIC_VECTOR(7 downto 0);
    clken : in STD_LOGIC;
    reset_n: in STD_LOGIC;
    clock : in STD_LOGIC
);
end component;
signal datain  : std_logic_vector (7 downto 0);
signal dataout : std_logic_vector (7 downto 0);
signal reset_n : std_logic;
signal clock   : std_logic;
signal clken   : std_logic;
begin
dut: sobel generic map
(
        row_width => platos
)
port map
(
        datain          => datain,
        dataout         => dataout,
        reset_n         => reset_n,
        clock                   => clock,
        clken     => clken
);
process
file rp: text open read_mode is "input_im.dat"; -- arxeio apo to opoio 8a diabasei ta dedomena
variable line_rd: line;
```

```
file wp: text open write_mode is "out_im.dat"; -- arxeio poy 8a dhmioyrgh8ei
variable line_wr: line;
variable a: integer range 0 to 2 ** 8 - 1;
variable i, j: integer;
begin
reset_n <= '0';
clock   <= '0';
clken   <= '0';
wait for 10 ns;
reset_n <= '1';
wait for 10 ns;
i := 1;
j := 1;
while(not endfile(rp)) loop
        -- dedomena apo eikona
        readline(rp, line_rd);
        read(line_rd, a);
        datain <= std_logic_vector(to_unsigned(a, 8));
        clock <= '1';
        clken <= '1'; -- pros8hkh den yphrxe sto paradeigma
        wait for 10 ns;
        -- paragomena dedomena
        if ((i >= 0) and (j >= 0)) then
                write(line_wr, to_integer(unsigned(dataout)));
                writeline(wp, line_wr);
        end if;
        clock <= '0';
        wait for 10 ns;
        if j < platos then
                j := j + 1;
        else
                j := 1;
```

```
                i := i + 1;
        end if;
end loop;
end process;
    end sobel_tb_Arch;
```

Ο κώδικας του test bench, παράγει τις κατάλληλες εισόδους κατά την προσομοίωση, όπως θα συνέβαινε σε συνθήκες πραγματικής λειτουργίας, με σκοπό τον έλεγχο της ορθότητας των αποτελεσμάτων. Ελέγχοντας το κύκλωμα στην εξομοίωση του υπολογιστή, μπορούμε να λάβουμε τα ίδια αποτελέσματα, με αυτά που θα λαμβάναμε εάν το κύκλωμα είχε αποτυπωθεί σε ένα FPGA.

Όπως παρατηρούμε στον κώδικα του test bench, δηλώθηκε το πλάτος της εικόνας, για την οποία θα ελέγξουμε τα αποτελέσματα, κατά την εξομοίωση του κυκλώματος του φίλτρου Sobel. Το πλάτος αυτό θα πρέπει να είναι ίδιο με το πλάτος που δηλώθηκε στον κώδικα του φίλτρου Sobel, ώστε να παραχθούν σωστά τα αποτελέσματα. Επίσης στον κώδικα του testbench, δίδουμε ρολόϊ (clock & clken) και reset στο κύκλωμα, ώστε να εξομοιώσουμε την λειτουργία του. Στην αρχή το clken και το reset έχουν την τιμή μηδέν (0) και όταν ξεκινάει να διαβάζει τα δεδομένα από την εικόνα, μέσω του αρχείου δεδομένων dat, οι τιμές τους γίνονται ένα (1), για όση χρονική διάρκεια υπάρχουν δεδομένα εισερχόμενα από το αρχείο dat (while(not endfile(rp)) loop). Τέλος τα αποτελέσματα από την επεξεργασία της εικόνας με το κύκλωμα του φίλτρου Sobel, αποθηκεύονται σε ένα αρχείο δεδομένων dat.

Ο κώδικας που γράφτηκε στο Quartus, για το κύκλωμα του φίλτρου Sobel και του καταχωρητή μνήμης μπορεί να λειτουργήσει σε οποιοδήποτε FPGA, ασχέτως του κατασκευαστή και του μοντέλου.

4.2.2 Ο κώδικας για την μετατροπή της εικόνας

Για να μπορέσει να γίνει η προσομοίωση και να λάβει τα δεδομένα της εικόνας το κύκλωμα που σχεδιάσαμε, θα πρέπει πρώτα η εικόνα να μετατραπεί σε εικόνα κλίμακας του γκρι. Στην συνέχεια, τα δεδομένα της φωτεινότητας του κάθε pixel της εικόνας, θα

αποθηκευθούν σε ένα αρχείο dat, έτσι ώστε τα δεδομένα για το κάθε pixel να έχει τιμές φωτεινότητας από τιμή μηδέν (0) έως την τιμή διακόσια πενήντα πέντε (255). Για το σκοπό αυτό, λοιπόν, δημιουργήθηκε κώδικας στο matlab, ο οποίος μετατρέπει την εικόνα σε εικόνα κλίμακας του γκρι και στην συνέχεια αποθηκεύει τα δεδομένα σε ένα αρχείο δεδομένων dat. Ο κώδικας που δημιουργήθηκε είναι ο ακόλουθος:

```
% Open a color image and convert it into grayscale
% and then into a 1D dat file for input in ModelSim
% image_name = 'TestImage.bmp';
image_name = 'tiger.jpg';
if (length(dir(image_name))>0)
  % read test image TestImage.bmp
%          orig_im= imread('TestImage.bmp', 'bmp');
  orig_im= imread(image_name);
  % convert to gray scale
  [nrws ncls colr] = size(orig_im);
  dbl=double(orig_im);
  gray_im      = zeros(nrws, ncls);
  gray_factors = [0.3; 0.59; 0.11];
  for j=1:ncls
     for i=1:nrws
        gray_im(i,j)= [dbl(i,j,1) dbl(i,j,2) dbl(i,j,3)] * gray_factors;
      end
  end
            % linearized array
  r1 = double(gray_im)+1;
  my_in_array = (1:ncls*nrws)';
  for i=1:nrws
     for j=1:ncls
        my_in_array((i-1)*ncls+j) = uint8(r1(i,j));
      end
   end
    save('input_im.dat', 'my_in_array', '-ASCII');
```

```
 else
  disp([' > Unable to locate the image TestImage.bmp']);
end;
```

Ο παραπάνω κώδικας επεξεργάζεται την εικόνα εισόδου, πολλαπλασιάζοντας την τιμή του κάθε χρώματος RGB, για το κάθε pixel ξεχωριστά με τους συντελεστές βαρύτητας 0.3, 0.59, 0.11, αντίστοιχα για κάθε χρώμα. Το πλάτος της προς μετατροπή εικόνας, λαμβάνεται αυτόματα από τον κώδικα. Η εικόνα με αυτόν τον κώδικα, μετατρέπεται σε εικόνα δεδομένων dat με σκοπό την επεξεργασία της, στην συνέχεια, μέσω της εξομοίωσης του φίλτρου Sobel στο ModelSin. Το αρχείο δεδομένων dat είναι ένα αρχείο σε μορφή πίνακα, με πλάτος μίας στήλης και πολλών σειρών. Συγκεκριμένα οι σειρές του πίνακα μέσα στο αρχείο dat είναι τόσες όσο και το αποτέλεσμα του πολλαπλασιασμού, του πλάτους της εικόνας με το ύψος αυτής, δηλαδή ο συνολικός αριθμός των pixel της εικόνας. Το αρχείο αυτό διευθετεί την εικόνα, όπως αυτή τοποθετείται στη μνήμη υπολογιστικού συστήματος.

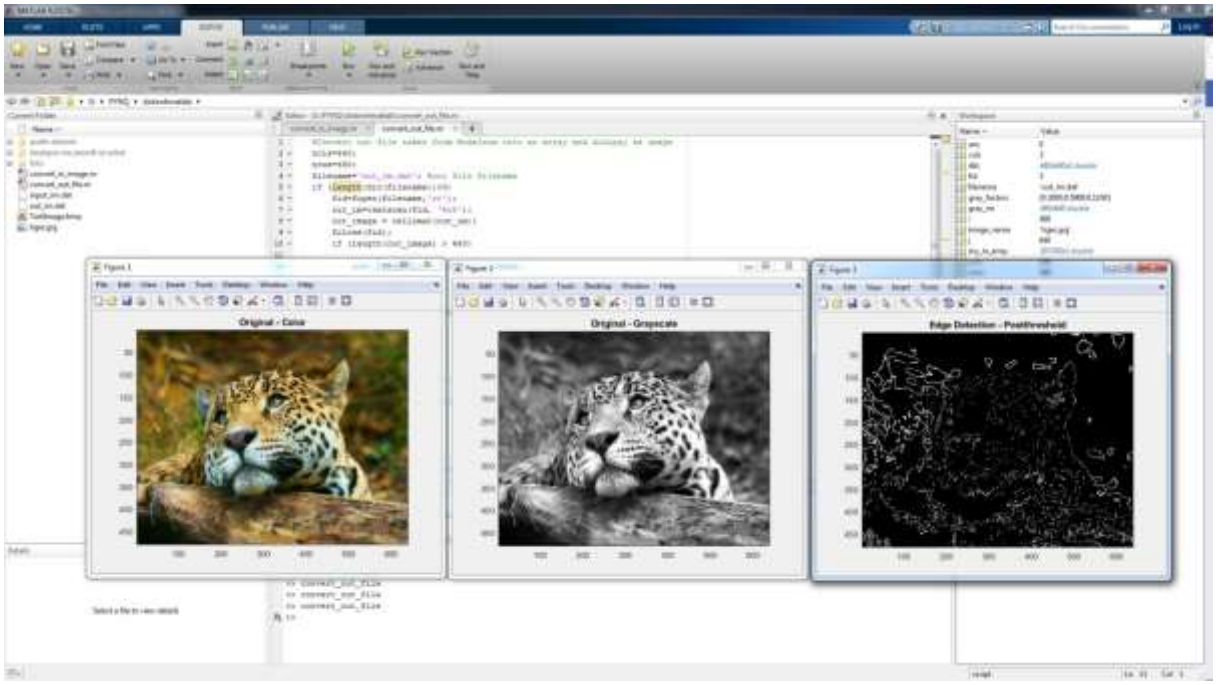
Στην συνέχεια δημιουργήθηκε ένα αρχείο ακόμα στο Matlab, με σκοπό να προβάλουμε την εξαγόμενη εικόνα από το αποτέλεσμα της εξομοίωσης του φίλτρου Sobel στο ModelSim. Η εικόνα που θα εξαχθεί από την εξομοίωση, είναι αποθηκευμένη σε αρχείο δεδομένων dat. Ο κώδικας που γράφηκε στο Matlab για την προβολή της εικόνας από το εξαγόμενο αρχείο δεδομένων dat, είναι ο ακόλουθος.

```
%Convert out file taken from Modelsim into an array and dislpay as image
ncls=1280;
nrws=720;
filename='out_im.dat'; %out file filename
if (length(dir(filename))>0)
        fid=fopen(filename,'rt');
   out_im=textscan(fid, '%u8');
   out_image = cell2mat(out_im);
   fclose(fid);
        if (length(out_image) > 1280)
                post_threshold  = uint8(out_image);
```

```
                for i=1:nrws
                   for j=1:ncls
                     post_thresh_im(i,j) = post_threshold((i-1)*ncls+j);
                   end
                end
                figure(1);
%               subplot(2,2,1);
                colormap('default');
                image(orig_im);
                title('Original - Color');
      figure(2);
%               subplot(2,2,2);
                colormap(gray);
                image(gray_im/3.5); % Scale data to use full color map
                title('Original - Grayscale');

%               subplot(2,2,3);
%               colormap(gray);
%               image(pre_thresh_im/3.5);
%               title('Edge Detection - Prethreshold');
%               %pause(1);
                figure(3);
%               subplot(2,2,3);
                colormap(gray);
                image(post_thresh_im/3.5);
                title('Edge Detection - Postthreshold');
   else
                disp([' > Incomplete simulation. Unable to display the image.']);
         end
else
         disp([' > Unable to locate the image TestImage.bmp']);
end
```

Είναι σημαντικό και εδώ να δηλώσουμε το πλάτος και το ύψος της εικόνας που θα προβάλουμε. Το πλάτος και το ύψος θα πρέπει να είναι το ίδιο με το πλάτος της αρχικής εικόνας, δηλαδή 1280 pixel x 720 pixel. Ο κώδικας αυτός, λαμβάνει τα δεδομένα από το αρχείο δεδομένων dat και στην συνέχεια μετατρέπει τον μονοδιάστατο πίνακα του αρχείου dat, σε εικόνα και την προβάλει. Συγκεκριμένα λαμβάνει τις πρώτες 1280 σειρές του πίνακα, δημιουργώντας έτσι την πρώτη σειρά δεδομένων της εικόνας. Στην συνέχεια μεταβαίνει στην δεύτερη σειρά της εικόνας και προσθέτει άλλα 1280 pixel σε αυτήν και συνεχίζει έτσι έως ότου συμπληρωθούν και οι 720 σειρές τις εικόνας. Με την ολοκλήρωση αυτής της διαδικασίας, γίνεται και η προβολή της παραγόμενης εικόνας ακμών από την εξομοίωση του κυκλώματος.



Εικόνα 4.2.2.1 Η αρχική εικόνα, η κλίμακας του γκρι και η παραγόμενη εικόνα ακμών, στο περιβάλλον του Matlab.

4.2.3 Η προσομοίωση στο Modelsim

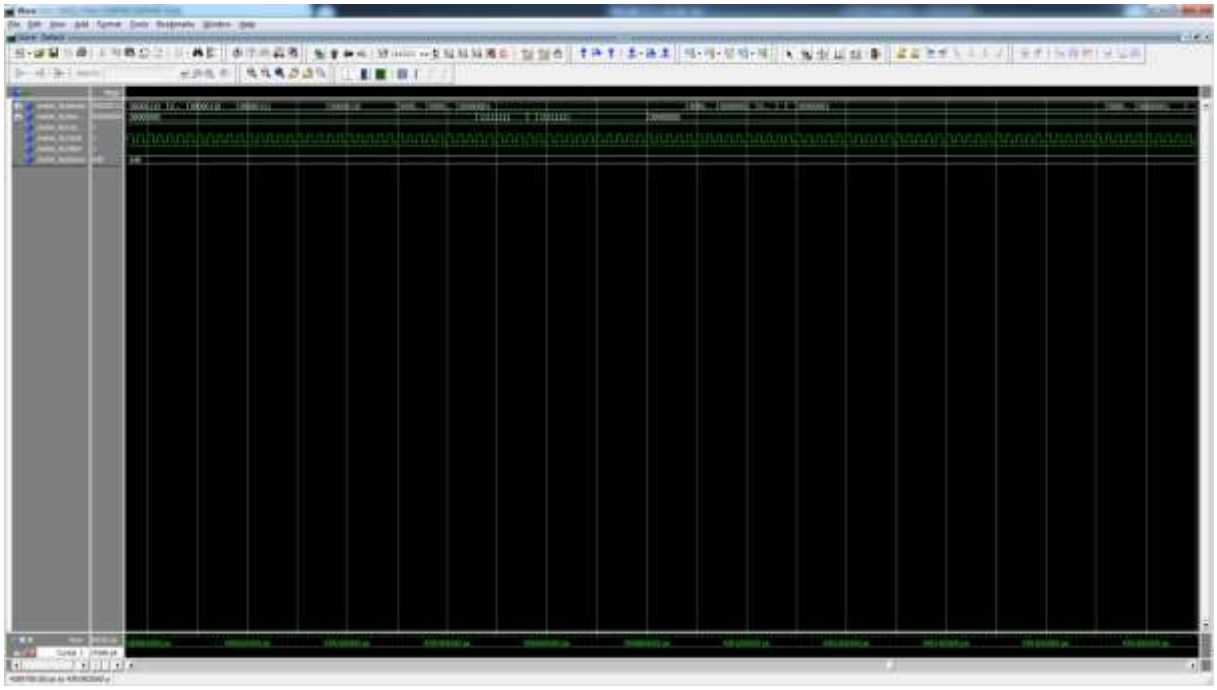
Στο λογισμικό του Modelsim, εκτελέσθηκε η προσομοίωση του κυκλώματος που σχεδιάστηκε για το φίλτρο Sobel. Για την εκτέλεση της προσομοίωσης, δηλώθηκε στο

ModelSim ο φάκελος στον οποίο περιεχόταν τα αρχεία VHD του φίλτρου Sobel, ο καταχωρητής ολίσθησης, το αρχείου testbench και τέλος, το αρχείο δεδομένων dat της εικόνας, η οποία δίνεται ως είσοδος στην προσομοίωση. Ως χρόνο προσομοίωσης επιλέγουμε τα 18432020 ns. Ο χρόνος αυτός επιλέχθηκε λόγω του ότι η εικόνα μας έχει πλάτος 1280 pixel και ύψος 720 pixel. Έτσι, ο χρόνος υπολογίζεται ως ακολούθως:

(1280 pixel x 720 pixel x 20 ns) + 20 ns = 18432020 ns.

Στην παραπάνω σχέση, το γινόμενο των pixels πολλαπλασιάζεται με τα 20 ns, διότι αυτός ο χρόνος απαιτείται για να ολοκληρωθεί μια περίοδος ρολογιού στο περιβάλλον της προσομοίωσης. Επιπροσθέτως, δίνουμε επιπλέον ένα περιθώριο της τάξης των 20 ns, διότι η προσομοίωση δεν ξεκινά από τον πρώτο χτύπο του ρολογιού. Για αυτό τον λόγο, η προσομοίωση ορίστηκε για χρόνο 18432020 ns. Ο χρόνος αυτός είναι εικονικός και δεν ανταποκρίνεται σε πραγματικό χρόνο επεξεργασίας των δεδομένων, κατά την προσομοίωση. Ο πραγματικός χρόνος της προσομοίωσης, εξαρτάται πάντοτε από τις επιδόσεις του υπολογιστικού συστήματος, στο οποίο αυτή εκτελείται. Έτσι, στην πραγματικότητα ο χρόνος που απαιτείται για την ολοκλήρωση της προσομοίωσης στο ModelSim είναι μεγαλύτερος από τον ορισθέντα.

Αφού ολοκληρωθεί η προσομοίωση του κυκλώματος, εξάγεται από αυτήν ένα νέο αρχείο δεδομένων dat στον φάκελο που έχει δηλωθεί πριν την έναρξη της προσομοίωσης.



Εικόνα 2.8.2 Το επιτυχές αποτέλεσμα της προσομοίωσης, όπως φαίνεται στο γραφικό περιβάλλον του ModelSim

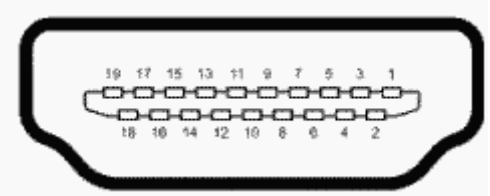
4.3 Δημιουργία νέου project στο Vivado

Στο περιβάλλον του Vivado, θα πρέπει να δημιουργηθεί ένα νέο project, ώστε να δηλωθεί σε αυτό η αναπτυξιακή πλακέτα PYNQ-Z1 και να δημιουργηθεί το κατάλληλο κύκλωμα, για την εξαγωγή της εικόνας ακμών, μέσω της εφαρμογής του φίλτρου Sobel. Για να δημιουργηθεί ένα νέο project στο Vivado, επιλέγουμε στην αρχική επιφάνεια του προγράμματος, από το μενού Quick Start, την επιλογή Create Project. Στην συνέχεια, δηλώνουμε το όνομα του project (στην συγκεκριμένη περίπτωση το όνομα που δόθηκε είναι το video_in_out_test2_project), την θέση αποθήκευσής του στον υπολογιστή και στην συνέχεια το δηλώνουμε ως RTL Project, με σκοπό να μπορούμε να εισάγουμε σε αυτό υπάρχοντα κυκλώματα σχεδιασμένα σε VHDL, από βιβλιοθήκες. Τέλος, δηλώνουμε την αναπτυξιακή πλακέτα PYNQ-Z1, στην οποία θα δουλέψουμε. Στην συνέχεια θα αντλήσουμε από τις βιβλιοθήκες πνευματικής περιουσίας (IPs) τις απαραίτητες βαθμίδες για το project. [47]

4.4 Εισαγωγή βαθμίδας IP HDMI στο project

4.4.1 Η θύρα HDMI

Για την επικοινωνία των θυρών HDMI in και HDMI out του PYNQ-Z1, θα φορτώσουμε στο project που δημιουργήσαμε τη βιβλιοθήκη που περιέχει τις βαθμίδες πνευματικής περιουσίας (IPs) για τον έλεγχο των θυρών. Η θύρα HDMI αποτελείται από δεκαεννέα (19) pins. Από αυτά τα δεκαεννέα (19) pins, τα οκτώ (8) χρησιμεύουν στην μεταφορά εικόνας και ήχου. Τα οκτώ (8) αυτά pins, αποτελούν τα λεγόμενα TMDS (Transition Minimized Differential Signaling) σήματα. Το ένα TMDS από τα τέσσερα χρησιμοποιείται για τον συγχρονισμό του σήματος μεταφοράς και τα υπόλοιπα τρία χρησιμοποιούνται για την μεταφορά των χρωμάτων RGB.

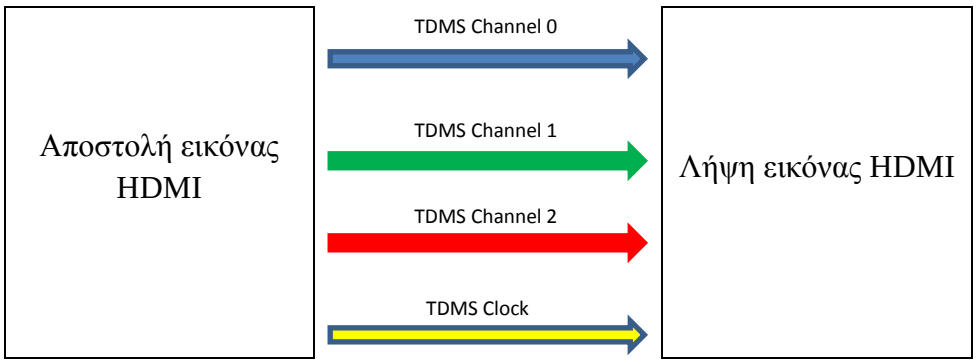


Εικόνα 4.4.1.1 Η θύρα HDMI και οι ακροδέκτες της

Στην εικόνα 4.4.1.1, φαίνεται η θύρα HDMI και οι ακροδέκτες της, σύμφωνα με την κατασκευή της. Τα τέσσερα TMDS που αναφέρθηκαν, με την αντιστοιχία τους στα pins της θύρας, είναι τα ακόλουθα.

- Pin 1 TMDS Data 2 +
- Pin 3 TMDS Data 2 –
- Pin 4 TMDS Data 1 +
- Pin 6 TMDS Data 1 –
- Pin 7 TMDS Data 0 +
- Pin 9 TMDS Data 0 -
- Pin 10 TMDS Clock +
- Pin 12 TMDS Clock –

Το κανάλι Data 0, αναλαμβάνει την μεταφορά του χρώματος μπλε της εικόνας, το κανάλι Data 1 αναλαμβάνει την μεταφορά του πράσινου χρώματος της εικόνας και τέλος, το κανάλι Data 2 αναλαμβάνει την μεταφορά του κόκκινου χρώματος της εικόνας. Με αυτόν τον τρόπο μεταφέρεται μέσω της θύρας HDMI η χρωματική πληροφορία RGB της εικόνας ή του βίντεο.



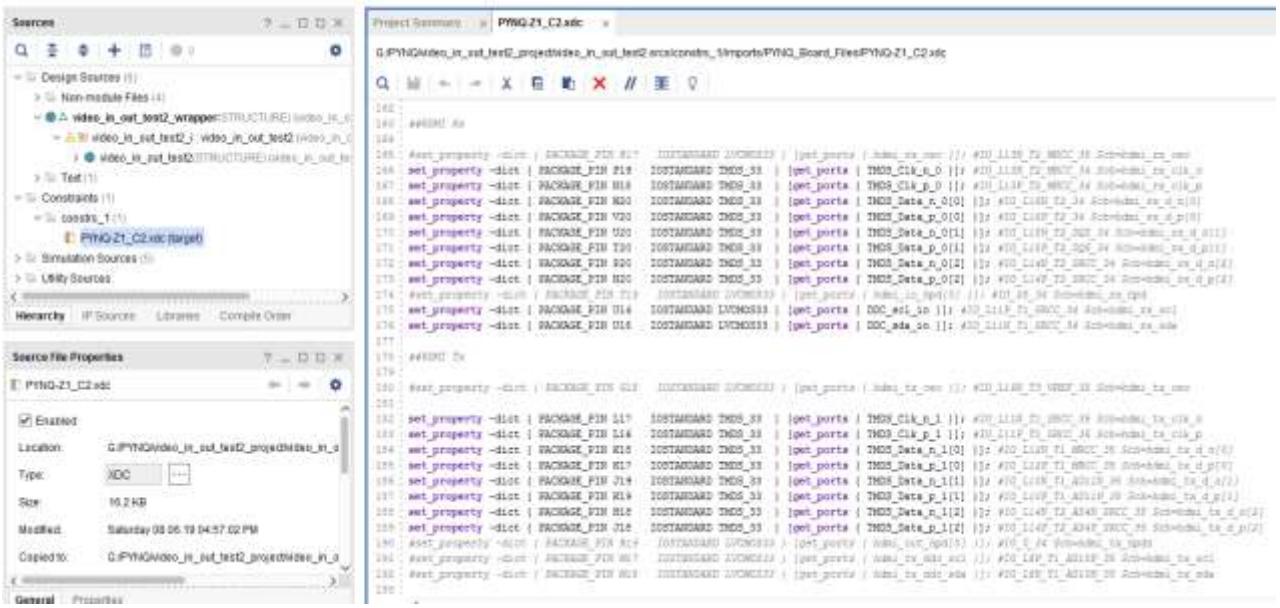
Εικόνα 4.4.1.2 Σκαρίφημα διάταξης TMDS

Ο συγχρονισμός του TDMS Clock, εξαρτάται από την ανάλυση και τη συχνότητα ανανέωσης που θέλουμε να έχουμε στην οθόνη. Για παράδειγμα, για ανάλυση 640 x 480, ο συγχρονισμός γίνεται στα 250 MHz. Για μεγαλύτερες αναλύσεις αυτός ο συγχρονισμός αλλάζει προς τα επάνω, με αποτέλεσμα κάποιες φορές να μην μπορεί να επιτευχθεί, λόγω της υπέρβασης του ορίου συγχρονισμού από το χρησιμοποιούμενο FPGA. Για παράδειγμα, η μέγιστη συχνότητα που μπορεί να δημιουργήσει το PYNQ-Z1, είναι τα 650 MHz για τον επεξεργαστή του και τα 525 MHz για την μνήμη DDR3 που διαθέτει [11].

4.4.2 Οι βαθμίδες IP που ελέγχουν τις θύρες HDMI

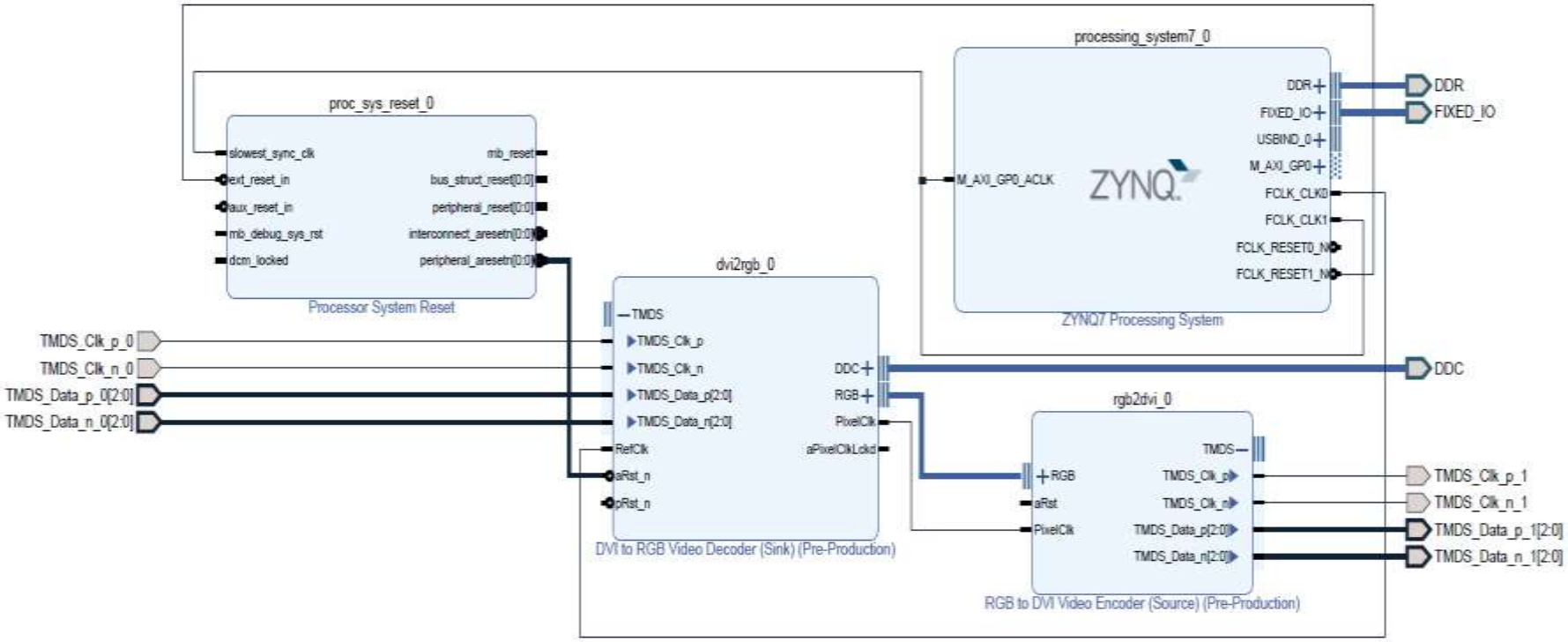
Οι βαθμίδες IP που θα χρησιμοποιηθούν στο project περιέχουν όλες τις απαραίτητες συνδέσεις και πληροφορίες που απαιτούνται, ώστε να είναι εφικτό το PYNQ-Z1 να δεχθεί εικόνα από την είσοδο HDMI και να την εξάγει στην συνέχεια στην έξοδο HDMI. Τα μπλοκ της βιβλιοθήκης είναι κατάλληλα σχεδιασμένα για την είσοδο και την έξοδο του HDMI, καθώς επίσης περιέχονται και όλες οι απαραίτητες ρυθμίσεις που απαιτούνται για την ενεργοποίηση των ακροδεκτών HDMI εισόδου και εξόδου. Οι αντιστοιχήσεις των ακροδεκτών γίνονται με τη βοήθεια του αρχείου PYNQ-Z1_C2.xdc. Το αρχείο PYNQ-Z1_C2.xdc είναι ένας χάρτης για όλες τους ακροδέκτες εισόδου και εξόδου που υπάρχουν επάνω στο PYNQ-Z1. Μέσω του αρχείου αυτού, μπορούμε να συνδέσουμε τις εισόδους και τις εξόδους του συστήματος που σχεδιάσαμε με τους ακροδέκτες της διάταξης.

Στην εικόνα 4.4.2.1, μπορούμε να δούμε τις αντιστοιχήσεις ακροδεκτών, που έχουν συμπεριληφθεί στο αρχείο PYNQ-Z1_C2.xdc, για τους ακροδέκτες HDMI της εισόδου (HDMI Px) και το HDMI της εξόδου (HDMI Tx) του PYNQ-Z1.



Εικόνα 4.4.2.1 Τα ενεργοποιημένα pin του αρχείου PYNQ-Z1_C2.xdc

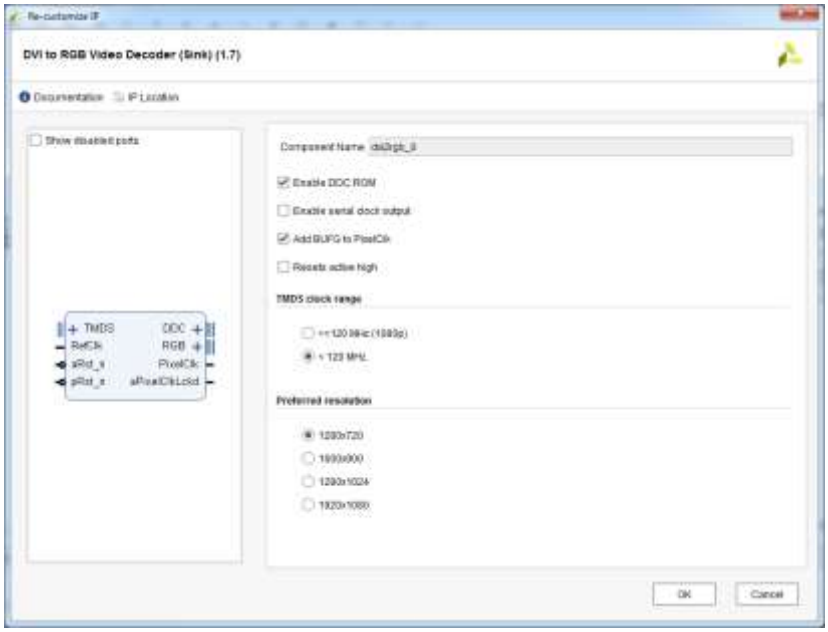
Οι βαθμίδες βιβλιοθήκης (IPs) που χρησιμοποιήσαμε, φαίνονται στην παρακάτω εικόνα του Vivado, όπου εμφανίζεται το διάγραμμα βαθμίδων του συστήματος. Τα IPs της βιβλιοθήκης είναι το DVI to RGB Video Decoder, το οποίο ελέγχει την είσοδο της εικόνας στο HDMI εισόδου, το RGB to DVI Video Encoder, το οποίο ελέγχει την έξοδο της εικόνας στο HDMI εξόδου, το ZYNQ Processing System το οποίο μας δίνει τον απαραίτητο συγχρονισμό και τέλος το Processor System Reset.



Εικόνα 4.4.2.2 Τα IP της βιβλιοθήκης στο διάγραμμα βαθμίδων του Vivado

Πιο συγκεκριμένα, το ZYNQ Processing System ελέγχει τα περιφερειακά του PYNQ-Z1 και τον επεξεργαστή ARM. Αυτό μας παρέχει το σήμα αναφοράς για τον συγχρονισμό της εικόνας εισόδου (200 MHz), το ρολόϊ για τον χρονισμό της μνήμης DDR3 στα 525 MHz και του επεξεργαστή στα 650 MHz. Η συχνότητα χρονισμού της μνήμης και του επεξεργαστή, είναι οι μέγιστες δυνατές συχνότητες που μπορεί να παρέχει η αναπτυξιακή πλακέτα PYNQ-Z1, για τον σκοπό αυτό. Επίσης, το ZYNQ Processing System, ελέγχει και την λειτουργία του Processor System Reset, το οποίο είναι υπεύθυνο για την παραγωγή των σημάτων επανεκκίνησης (reset) των διαφόρων βαθμίδων.

Στο IP DVI to RGB Video Decoder, μπορούμε να επιλέξουμε την ανάλυση της εικόνας εισόδου και το εύρος του συγχρονισμού του TMDS clock. Για την υλοποίηση της εργασίας μας, επιλέγεται ανάλυση 1280 x 720 και εύρος συγχρονισμού των pixels μικρότερο από 120 MHz. Στην περίπτωση μας, για μέγεθος εικόνας εισόδου 1280 x 720 με ρυθμό ανανέωσης 60 Hz, το ρολόι συγχρονισμού των pixel είναι στα 74,25 MHz και ο συγχρονισμός του καναλιού



Εικόνα 4.4.2.3 Οι ρυθμίσεις του IP DVI to RGB Video Decoder

TMDS είναι στα 742,50 MHz. [37] Ωστόσο, υπενθυμίζουμε ότι το ρολόϊ αναφοράς για την αποσειριοποίηση των σημάτων έχει ορισθεί στα 200 MHz.

4.5 Δημιουργία βοηθητικών κυκλωμάτων

Για να μπορέσουμε να επεξεργασθούμε την εισερχόμενη εικόνα μέσω του φίλτρου Sobel στο διάγραμμα βαθμίδων του συστήματος, θα πρέπει να εισάγουμε το φίλτρο μαζί με τον καταχωρητή μνήμης (shift register) και τέλος, να δημιουργήσουμε μερικά ακόμα κυκλώματα με κώδικα VHDL. Τα κυκλώματα που πρέπει να δημιουργήσουμε είναι :

α) μετατροπή της εικόνας από έγχρωμη σε εικόνα αποχρώσεων του γκρι

β) μετατροπή της εικόνας από κλίμακα του γκρι σε έγχρωμη εικόνα RGB

Για την μετατροπή της εικόνας εισόδου από έγχρωμη σε αποχρώσεις του γκρι, θα δημιουργηθούν δύο κυκλώματα. Το πρώτο, περιέχει κώδικα για τον διαχωρισμό των καναλιών χρώματος RGB της εικόνας εισόδου. Το κάθε κανάλι χρώματος έχει μέγεθος 8 bit. Το δεύτερο θα περιέχει τον κατάλληλο κώδικα για την μετατροπή της σε κλίμακας του γκρι εικόνα.

Για την δημιουργία των τριών νέων κυκλωμάτων, επιλέγουμε στην καρτέλα Sources, το εικονίδιο Add sources (+) και στο παράθυρο διαλόγου που εμφανίζεται επιλέγουμε, Add or create design sources. Αφού ονοματίσουμε το κάθε νέο κύκλωμα ξεχωριστά, προχωρούμε έπειτα με την γραφή του κώδικα περιγραφής του κάθε κυκλώματος.

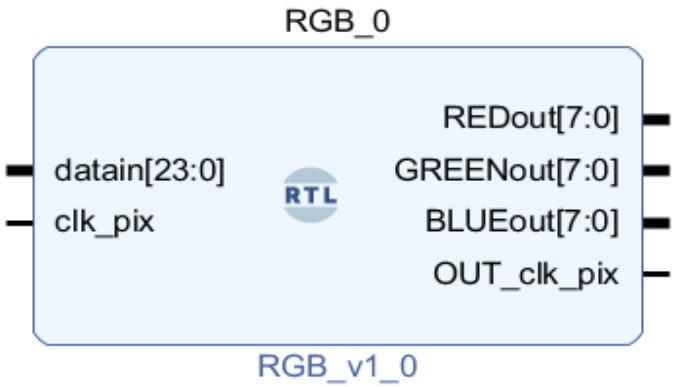
Ξεκινάμε με την μετατροπή του σήματος της εικόνας μεγέθους 24 bit που λαμβάνουμε από τη βαθμίδα HDMI in, σε τρία ξεχωριστά κανάλια χρώματος RGB των 8 bit το καθένα. Το κάθε εικονοστοιχείο περιέχει και τα τρία κανάλια χρώματος, έτσι ώστε να περιγράφεται με σαφήνεια το χρώμα του. Ο κώδικας για την δημιουργία των τριών διαφορετικών χρωματικών καναλιών της εικόνας, είναι ο ακόλουθος.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity RGB is
generic
(
   word_width: integer := 8     -- dhlwsh bit
```

```
);
Port
(
   datain     : in std_logic_vector ((word_width * 3) - 1 downto 0); -- dedomena eisodoy
   REDout     : out std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Red
   GREENout   : out std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Green
   BLUEout    : out std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Blue
         clk_pix    : in STD_LOGIC;
         OUT_clk_pix: out STD_LOGIC
);
end entity;
architecture Behavioral of RGB is
begin
RGBSplit:
process(clk_pix)
begin
         if rising_edge(clk_pix) then
REDout   <= datain ((word_width * 3) - 1 downto word_width * 2);
GREENout <= datain ((word_width * 2) - 1 downto word_width);
BLUEout  <= datain (word_width - 1 downto word_width - word_width);
         end if;
end process RGBSplit;
OUT_clk_pix <= clk_pix;
end Behavioral;
```

Στον κώδικα δηλώνουμε ως γενική μεταβλητή το μέγεθος της πληροφορίας των 8 bit, του κάθε καναλιού χρώματος. Για την δημιουργία των καναλιών, λαμβάνουμε από το εισερχόμενο σήμα της εικόνας το οποίο είναι μεγέθους 24 bit, τα πρώτα 8 bit για το χρωματικό κανάλι του κόκκινου, τα επόμενο 8 bit για το χρωματικό κανάλι του πράσινου και τέλος τα εναπομείναντα 8 bit για το χρωματικό κανάλι του μπλε. Επίσης, στον κώδικα δηλώνουμε το ρολόι συγχρονισμού clk_pix, το οποίο θα δουλεύει συγχρονισμένα με το ρολόι PixelClk, της θύρας HDMI in. Στο τέλος του κώδικα, δηλώνουμε και την έξοδο του ρολογιού

συγχρονισμού εισόδου clk_pix, στο ρολόι OUT_clk_pix, έτσι ώστε να συνδεθεί με το επόμενο IP, που θα σχεδιάσουμε στον κώδικα που θα γράψουμε. Στην συνέχεια, μέσω της αυτόματης διαδικασίας μετατροπής σε RTL block του Vivado, εισάγουμε το νέο RTL block (εικόνα 4.5.1), που αφορά την εξαγωγή των καναλιών χρώματος της εικόνας, στο παράθυρο του block design diagram.



Εικόνα 4.5.1 Το RTL block του κυκλώματος εξαγωγής των καναλιών RGB της εικόνας

Στην συνέχεια πληκτρολογούμε τον κώδικα της μετατροπής της εικόνας σε εικόνα κλίμακας του γκρι, με βάση τις εξόδους των τριών καναλιών χρώματος του προηγούμενου κώδικα. Όσον αφορά το ρολόι συγχρονισμού pixel_clock δηλαδή το ρολόι με το οποίο μεταδίδονται τα pixel στο σήμα native video, εργαζόμαστε ομοίως με τον προηγούμενο κώδικα.

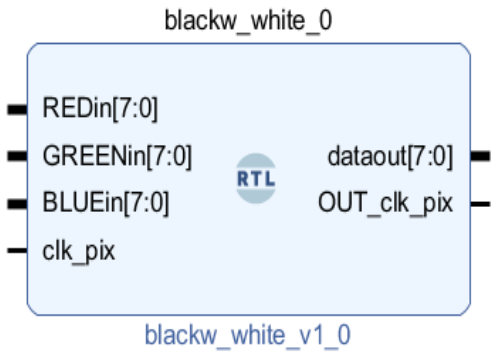
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity blackw_white is
generic
(
   word_width: integer := 8     -- dhlwsh bit
);
```

```
Port
(
    REDin      : in std_logic_vector  (word_width - 1 downto 0); -- dedomena eisodoy
   GREENin    : in std_logic_vector  (word_width - 1 downto 0); -- dedomena eisodoy
   BLUEin     : in std_logic_vector  (word_width - 1 downto 0); -- dedomena eisodoy
   dataout    : out std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy
  clk_pix    : in STD_LOGIC;
  OUT_clk_pix: out STD_LOGIC
);
end entity;
architecture Behavioral of blackw_white is
SIGNAL grayPixel_u   : UNSIGNED((word_width * 2) - 1 downto 0);
begin
VideoEdition:

process(clk_pix)
begin
  if rising_edge(clk_pix) then
            grayPixel_u <= (x"4c"*unsigned(REDin) + x"97"*unsigned(GREENin) +
x"1C"*unsigned(BLUEin)); --weighted color
  end if;
end process VideoEdition;
dataout <= std_logic_vector(grayPixel_u((word_width * 2) - 1 downto word_width));
OUT_clk_pix <= clk_pix;
end Behavioral;
```

Ο παραπάνω κώδικας παράγει μια έξοδο των 8 bit, η οποία περιέχει την πληροφορία για την απόχρωση του κάθε εικονοστοιχείου της εικόνας σε κλίμακα του γκρι. Η πληροφορία αυτή θα τροφοδοτήσει την είσοδο του φίλτρου Sobel. Για την μετατροπή της εικόνας σε

εικόνα κλίμακας του γκρι με μέγεθος πληροφορίας του κάθε εικονοστοιχείου της 8 bit, θα πρέπει πρώτα, να το κάθε κανάλι χρώματος ξεχωριστά να πολλαπλασιαστεί με έναν παράγοντα βαρύτητας και στην συνέχεια να προστεθούν τα αποτελέσματα. Συγκεκριμένα, για να γίνει αυτό, το κάθε χρωματικό κανάλι δηλώνεται ως σήμα Unsigned, για να γίνει πολλαπλασιασμός αυτού με τον παράγοντα βαρύτητας. Για το κανάλι χρώματος κόκκινο ο συντελεστής αυτός είναι το "4c", για το κανάλι πράσινο είναι το "97" και τέλος για το κανάλι μπλε είναι το "1C", στο δεκαεξαδικό σύστημα αρίθμησης. Το άθροισμα των συντελεστών βαρύτητας πρέπει να ισούται με την μονάδα ή αλλιώς το 100 %. Ο επιμερισμός για τον κάθε συντελεστή έχει ως ακολούθως, ο συντελεστής 4c αντιστοιχεί στο 29,9 %, ο συντελεστής 97 στο 58,7 % και τέλος ο συντελεστής 1C στο 14,4 %. Τα αποτελέσματα αυτά προστίθενται στο τέλος και μας δίνουν μια τιμή για την φωτεινότητα του κάθε εικονοστοιχείου της εικόνας, σε κλίμακα του γκρι. Η τιμή αυτήν έχει εύρος από το μηδέν (0) έως το διακόσια πενήντα πέντε (255), χωρίς ποτέ να ξεπερνάει την μέγιστη αυτήν τιμή. Για την μετατροπή σε RTL block, χρησιμοποιούμε ξανά την αυτόματη διαδικασία μετατροπής σε RTL block του Vivado, εισάγουμε το νέο RTL block (εικόνα 4.5.2). Τέλος το νέο RTL block εισάγεται στο παράθυρο του διαγράμματος βαθμίδων.

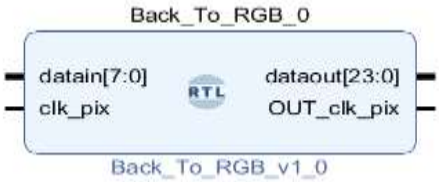


Εικόνα 4.5.2 Το RTL block του κυκλώματος εξαγωγής της κλίμακας του γκρι εικόνας

Τέλος, δημιουργούμε τον τρίτο και τελευταίο κώδικα για την περιγραφή του κυκλώματος, που θα κάνει την μετατροπή της εικόνας εξόδου από το φίλτρο Sobel, από pixels εύρους 8 bit, σε εικόνα εξόδου 24 bit RGB. Η έξοδος της εικόνας των 24 bit, θα τροφοδοτήσει το IP της θύρας HDMI out.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Back_To_RGB is
generic
(
   word_width: integer := 8     -- dhlwsh bit
);
Port
(
   REDin     : in std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Red
   GREENin   : in std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Green
   BLUEin    : in std_logic_vector (word_width - 1 downto 0); -- dedomena e3odoy Blue
   dataout     : out std_logic_vector ((word_width * 3) - 1 downto 0) -- dedomena e3odoy Red
         clk_pix    : in STD_LOGIC;
         OUT_clk_pix: out STD_LOGIC
);
end entity;
architecture Behavioral of Back_To_RGB is
begin
dataout  (23 downto 16) <= REDin (15 downto 8);
dataout  (15 downto 8)  <= BLUEin (15 downto 8);
dataout  (7 downto 0)   <= GREENin (15 downto 8);
OUT_clk_pix <= clk_pix;
end Behavioral;
```

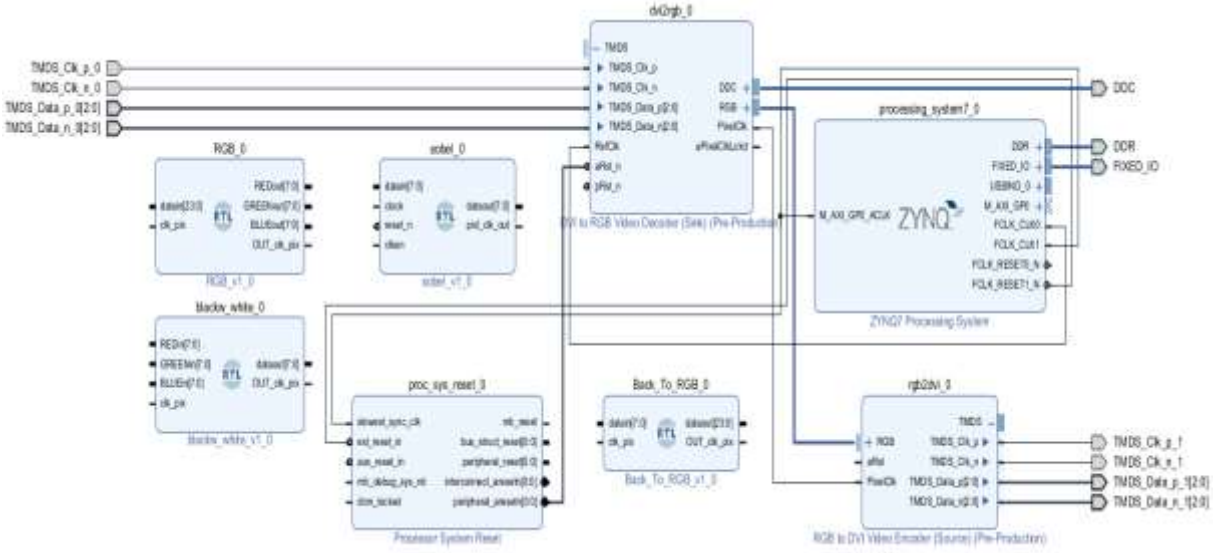
Σε αυτόν τον κώδικα λαμβάνεται το σήμα εισόδου από την έξοδο του φίλτρου Sobel, σε μορφή εικόνας της κλίμακας γκρι, με μέγεθος 8 bit και μετατρέπεται σε έγχρωμη εικόνα μεγέθους 24 bit. Το PixelClk, είναι συνδεδεμένο σε σειρά με το κύκλωμα του φίλτρου Sobel, από όπου λαμβάνεται και η παραχθείσα εικόνα ακμών. Για την μετατροπή και αυτού του σχεδιαζόμενου κυκλώματος σε μορφή RTL block, έγινε χρήση της αυτόματης διαδικασίας μετατροπής σε RTL block του Vivado. Στην συνέχεια το νέο RTL block (εικόνα 4.5.3), εισήχθη στο παράθυρο του διαγράμματος βαθμίδων (block design diagram).



Εικόνα 4.5.3 Το RTL block του κυκλώματος εξαγωγής της έγχρωμης εικόνας, από την εικόνα κλίμακας του γκρι

4.6 Δημιουργία του project στο block diagram

Αφού δημιουργήσουμε τα τρία αυτά αρχεία και τα βάλουμε όλα μέσα στο διάγραμμα βαθμίδων με την μέθοδο που περιγράφηκε προηγουμένως, η τελική μορφή στο παράθυρο του block design diagram του Vivado θα είναι όπως αυτή που φαίνεται στην εικόνα 4.6.1.



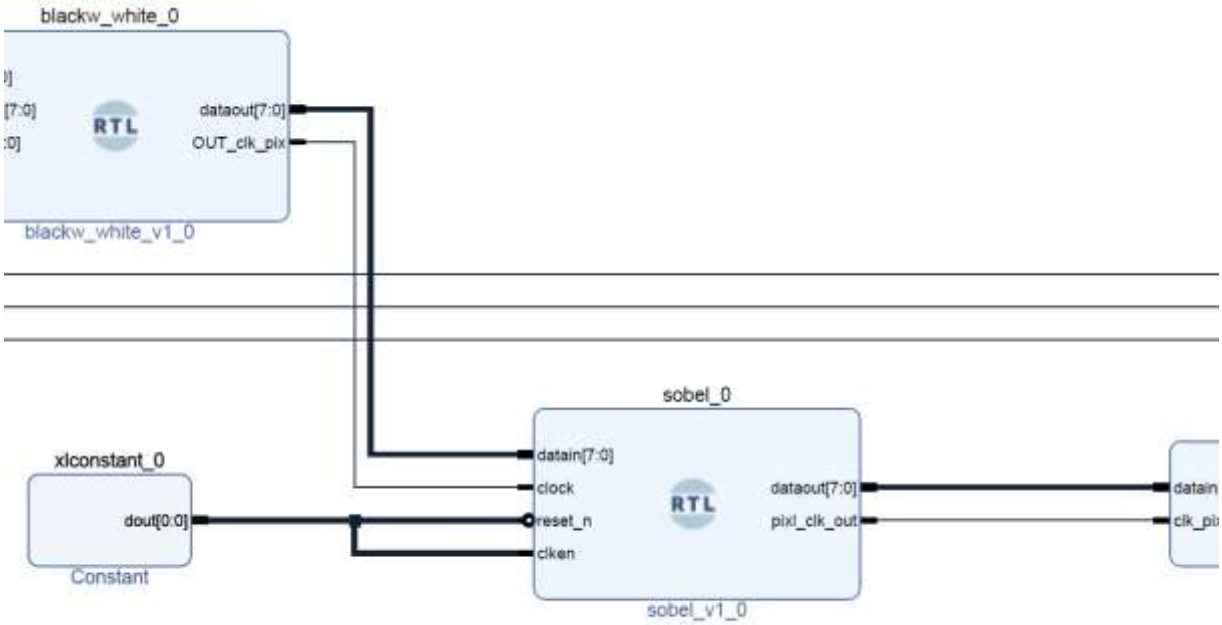
Εικόνα 4.6.1 Το παράθυρο του block design diagram του Vivado

Για να μπορέσουμε να ενσωματώσουμε στο block design, τα τέσσερα αρχεία που προσθέσαμε, θα πρέπει να δημιουργήσουμε τις συνδέσεις μεταξύ αυτών των βαθμίδων και των υπολοίπων βαθμίδων, DVI to RGB Video Decoder και RGB to DVI Video Encoder. Στην αρχή, θα διαγράψουμε τις υπάρχουσες συνδέσεις μεταξύ του DVI to RGB Video Decoder και του RGB to DVI Video Encoder και στην συνέχεια θα κάνουμε τις νέες

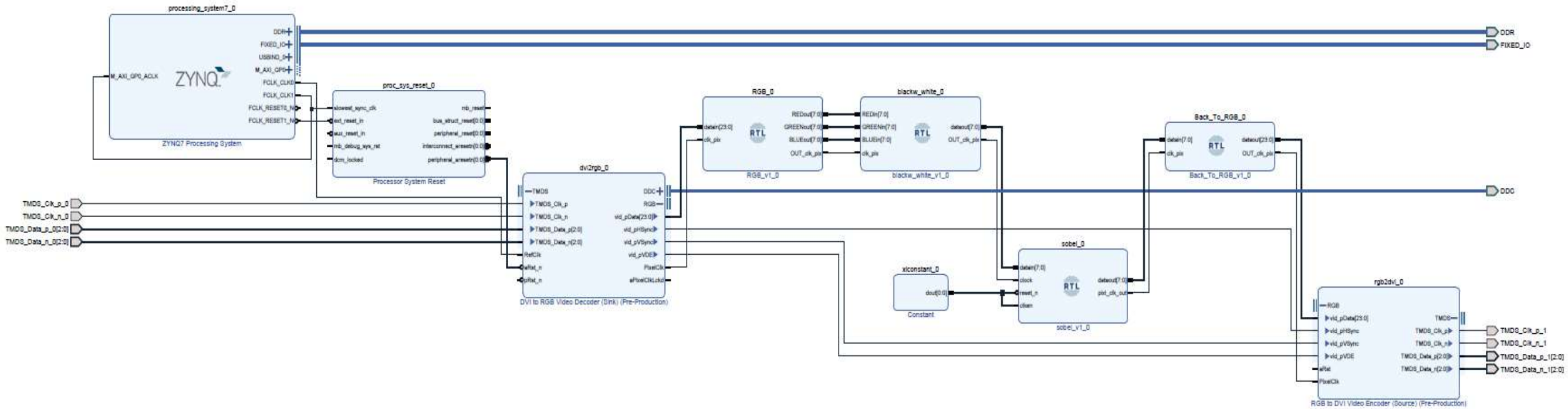
συνδέσεις, συμπεριλαμβάνοντας τα RTL blocks που δημιουργήσαμε. Η σειρά των RTL blocks πρέπει να έχει ως εξής: μετά την είσοδο HDMI, θα πρέπει να τοποθετήσουμε το αρχείο μετατροπής της έγχρωμης εικόνας σε εξόδους καναλιών RGB, στη συνέχεια το αρχείο για την μετατροπή της εικόνας σε κλίμακα του γκρι, έπειτα το φίλτρο Sobel και τέλος πριν την έξοδο του HDMI το αρχείο για την μετατροπή της εικόνας από κλίμακας του γκρι των 8 bit, σε εικόνα των 24 bit. Ο λόγος που δημιουργήθηκαν διαφορετικά μπλοκ για την κάθε διεργασία κατά την μετατροπή της εισερχόμενης εικόνας σε εικόνα ακμών είναι για τον καλύτερο έλεγχο της κάθε διαδικασίας. Με αυτόν τον τρόπο, μπορεί εύκολα να βρεθεί το σφάλμα σε περίπτωση δυσλειτουργίας, κατά τον έλεγχο του συστήματος, με την διεξαγωγή προσομοίωσης σε πραγματικές συνθήκες.

Το φίλτρο Sobel, έχει ως εισόδους το datain, το clock, το reset_n και τέλος το clken. Το datain δέχεται την εικόνα κλίμακας του γκρι. Το clock είναι συνδεδεμένο με το PixelClk, μέσω των κυκλωμάτων για την μετατροπή της εικόνας από έγχρωμη σε κλίμακας του γκρι. Το reset_n και το clken, είναι συνδεδεμένα με μια σταθερή μεταβλητή με τιμή 1, όπως φαίνεται στην εικόνα 4.6.2. Αυτό συμβαίνει για να είναι το reset_ και το clken, πάντα στο λογικό 1, για λόγους ορθής λειτουργίας και απλότητας. Όταν το clken έχει τιμή 1, τότε ξεκινάει και η ανάγνωση των δεδομένων από το φίλτρο και τον καταχωρητή μνήμης. Διαφορετικά, για να γίνει η ανάγνωση των δεδομένων από το φίλτρο και τον καταχωρητή μνήμης, θα μπορούσε να συνδεθεί το clken με το σήμα συγχρονισμού του video enable. Τότε η ενεργοποίηση του ρολογιού θα δινόταν από την θύρα HDMI εισόδου.

Εικόνα 4.6.2 Η μεταβλητή που συνδέεται με το reset_n και το clken



Εικόνα 4.6.3 Λεπτομέρεια με τις συνδέσεις του φίλτρου Sobel



Εικόνα 4.6.4 Οι συνδέσεις των κυκλωμάτων

Αφού δημιουργηθεί όλο το μπλοκ και γίνουν όλες οι απαραίτητες συνδέσεις στο Vivado, στη συνέχεια εκτελούμε την εντολή Refresh Changed Modules και έπειτα την εντολή Generate Bitstrean, από το μενού PROGRAM AND DEBUG. Μόλις ολοκληρωθεί με επιτυχία ο έλεγχος και η σύνθεση, τότε μπορούμε να δούμε πληροφορίες για τις επιδόσεις του συστήματος που σχεδιάστηκε.


Design Timing Summary

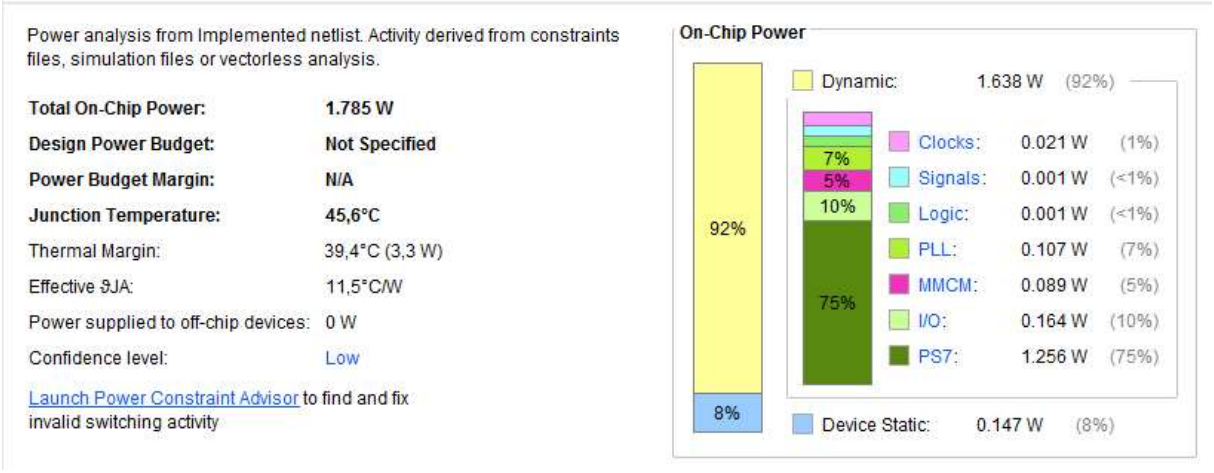
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1,419 ns	Worst Hold Slack (WHS):	0,059 ns	Worst Pulse Width Slack (WPWS):	0,264 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4209	Total Number of Endpoints:	4209	Total Number of Endpoints:	3482

All user specified timing constraints are met.

Εικόνα 4.6.5 Στοιχεία για τους χρόνους επίτευξης του συστήματος

Στην εικόνα 4.6.5, διακρίνονται οι χρόνοι απόκρισης του συστήματος. Σύμφωνα με τα αναγραφόμενα αποτελέσματα, βλέπουμε ότι οι χρόνοι που επετεύχθησαν είναι της τάξης των ns και ο μέγιστος χρόνος παρουσιάζεται είναι 1,419 ns.

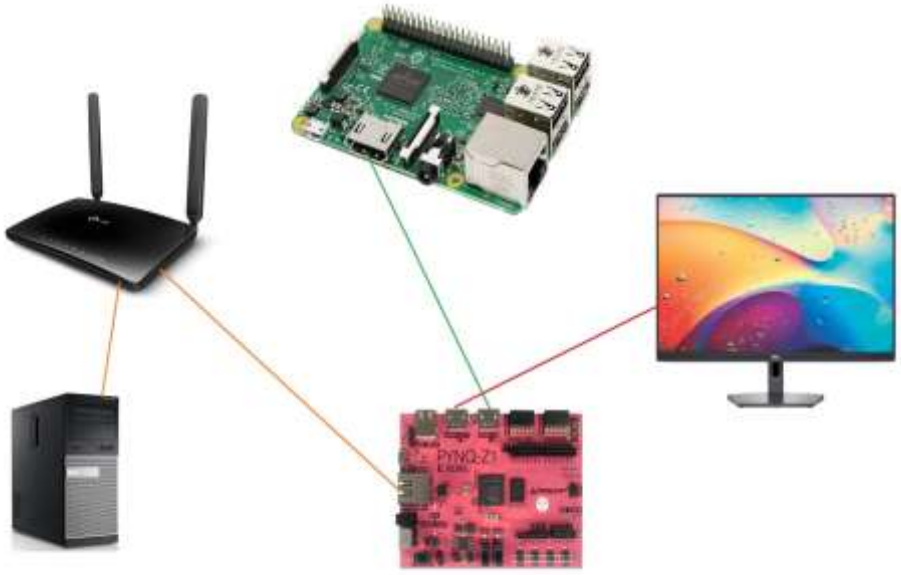
Στην εικόνα 4.6.6, εμφανίζονται τα στοιχεία για την ενεργειακή κατανάλωση του συστήματος και την θερμοκρασία που θα αναπτυχθεί από την λειτουργία του συστήματος. Η μέγιστη ενεργειακή κατανάλωση εμφανίζεται από το ZYNQ7 Processing System, με 1,256 Watt ή αλλιώς το 75%. Η συνολική κατανάλωση ενέργειας του συστήματος ανέρχεται στα 1,785 Watt και η αναλογία Watt και θερμοκρασίας είναι 11,5 ^{o}C ανά Watt.



Εικόνα 4.6.6 Στοιχεία για την ενεργειακή κατανάλωση του συστήματος

4.7 Δοκιμή του project

Στην συνέχεια, παράγουμε το αρχείο διαμόρφωσης (.bit), για ναδιαμορφώσουμε τη διάταξη FPGA. Αυτό το αρχεία παράγεται από το μενού File → Export → Export Bitstream File, εφόσον έχουμε ανοιχτό το παράθυρο του block diagram. Στην συνέχεια κάνουμε την κατάλληλη συνδεσμολογία με το PYNQ-Z1, το Raspberry Pi3, τον υπολογιστή, το μόντεμ και την οθόνη, ώστε να δοκιμάσουμε το φίλτρο Sobel (εικόνα 4.7.1).

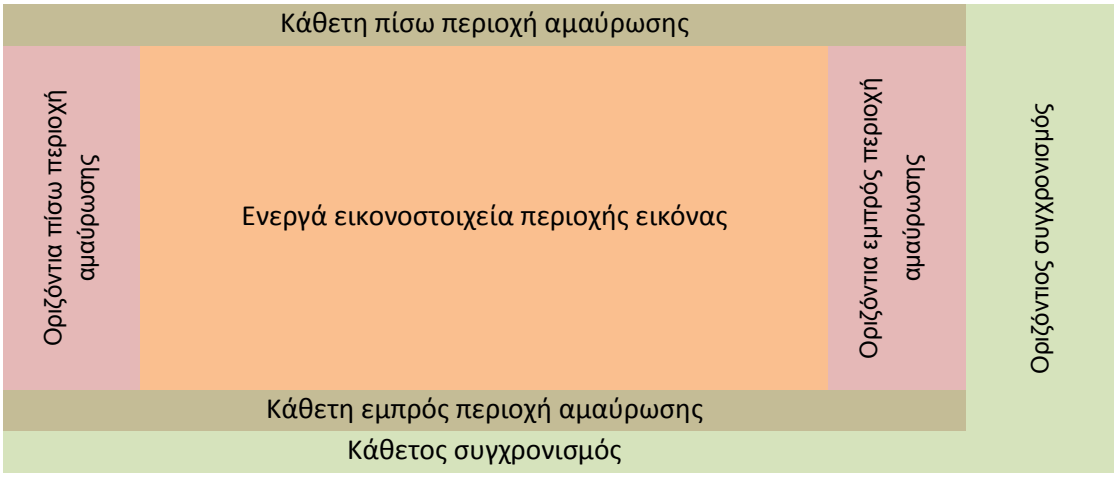


Εικόνα 4.7.1 Οι απαιτούμενες συνδέσεις για την δοκιμή του φίλτρου

Στην συνέχεια αφού έχουμε ρυθμίσει την ανάλυση στο Raspberry Pi3, σε μέγεθος 1280 x 720 στα 60 Hz, κάνουμε τον προγραμματισμό της διάταξης, ώστε να γίνει η διαμόρφωση του κυκλώματος στο FPGA και λαμβάνουμε αποτελέσματα στην οθόνη, η οποία είναι συνδεδεμένη με το PYNQ-Z1. Η παραγόμενη εικόνα που λαμβάνουμε φαίνεται στην εικόνα 4.7.2. Όπως διακρίνουμε δεν παράχθηκε σωστά η εικόνα.

Εικόνα 4.7.2 Το τελικό αποτέλεσμα της εικόνας, χωρίς την χρήση σημάτων συγχρονισμού

Αυτό οφείλεται στο ότι δεν χρησιμοποιήσαμε στην σχεδίαση σήματα συγχρονισμού και έτσι το σήμα της εικόνας περιέχει περιοχές αμαύρωσης. Οι περιοχές αυτές διακρίνονται στην εικόνα οριζόντια αλλά και κάθετα σε αυτήν, πριν και μετά τα ενεργά pixel της. [37]



Εικόνα 4.7.3 Γραφική αναπαράσταση των περιοχών αμαύρωσης

Το σήμα της εικόνας περιέχει και διαστήματα για τον οριζόντιο συγχρονισμό του, όπως βλέπουμε και στην εικόνα 4.7.3. Αυτά τα κενά διαστήματα ονομάζονται Horizontal Back Porch και Front Porch και βρίσκονται αριστερά και δεξιά από την περιοχή των pixels της εικόνας. Επίσης υπάρχουν και αντίστοιχα κενά για τον κάθετο συγχρονισμό της εικόνας. Λόγω της ύπαρξης αυτών των κενών θα πρέπει να αλλάξουμε το πλάτος της εικόνας στον κώδικα του φίλτρου Sobel και στον κώδικα του καταχωρητή μνήμης. Η νέα τιμή πλάτους που θα μπει για ανάλυση εικόνας 1280 x 720, θα είναι η τιμή 1650 (Total Pixels), σύμφωνα με τον ακόλουθο πίνακα (Πίνακας 1)

Πίνακας 1 Αριθμός pixels ανά περιοχή της εικόνας, για ανάλυση 1280 x 720, 60 Hz

Horizontal Timings	
Active Pixels	1280
Front Porch	110
Sync Width	40
Back Porch	220
Blanking Total	370
Total Pixels	1650
Sync Polarity	pos

Πίνακας 1. Πηγή : https://projectf.io/posts/video-timings-vga-720p-1080p/

Το ρολόϊ των εικονοστοιχείων (pixel clock) πρέπει να λειτουργεί στα 74.250MHz, ώστε να υποστηρίζεται συχνότητα ανανέωσης πλαισίων 60Hz.

Μετά την αλλαγή της τιμής της μεταβλητής του πλάτους της εικόνας, από 1280 σε 1650, στον κώδικα του φίλτρου Sobel και του καταχωρητή μνήμης, εκτελούμε εκ νέου την εντολή Refresh Changed Modules στο Vivado και έπειτα την εντολή Generate Bitstream, από το μενού PROGRAM AND DEBUG. Μόλις ολοκληρωθούν οι απαραίτητες διεργασίες, εκτελούμε ξανά την δοκιμή. Η συνδεσμολογία της διάταξης δοκιμής παραμένει ως έχει σύμφωνα με την εικόνα 4.4.1. Η παραγόμενη εικόνα που λαμβάνουμε μετά από τη διόρθωση της μεταβλητής του πλάτους φαίνεται στην εικόνα 4.7.5, ενώ η αρχική εικόνα εισόδου φαίνεται στην εικόνα 4.7.4.

Εικόνα 4.7.4 Η αρχική εικόνα

Εικόνα 4.7.5 Η παραγόμενη εικόνα ακμών

Στην συνέχεια εκτελέσθηκε δοκιμή με παραγόμενο βίντεο από το Raspberry Pi3 και διαπιστώθηκε ότι το PYNQ-Z1, παρήγαγε εικόνα ακμών του εισερχόμενου βίντεο από το Raspberry Pi3, με ρυθμό ανανέωσης 60 Hz.

Στο ίδιο αποτέλεσμα φτάνουμε, αν ενεργοποιούμε το ρολόϊ των εικονοστοιχείων μόνο στην περιοχή των ενεργών pixels της εικόνας. Αυτό επιτυγχάνεται με τη σύνδεση του ακροδέκτη VDE (video enable) στον ακροδέκτη clock enable (clken), Στην περίπτωση αυτή, το ρολόϊ του συστήματος που σχεδιάσαμε ενεργοποιείται μόνον στην περιοχή των ενεργών εικονοστοιχείων και όχι στις περιοχές συγχρονισμού και αμαύρωσης.

Κεφάλαιο 5. Συμπεράσματα και μελλοντικές προοπτικές έρευνας

5.1 Συμπεράσματα

Η παρούσα διπλωματική εργασία είχε ως σκοπό την διερεύνηση της επιτάχυνσης της παραγωγής εικόνας ακμών, με την εφαρμογή του φίλτρου Sobel σε πραγματικό χρόνο, με κώδικα σε γλώσσα περιγραφής υλικού και υλοποίηση στο σύστημα PYNQ-Z1. Η είσοδος της εικόνας, έγινε στο PYNQ-Z1, από την είσοδο HDMI που διαθέτει το σύστημα και η έξοδος της εικόνας ακμών έγινε από την έξοδο HDMI που διαθέτει.

Για την υλοποίηση της εργασίας χρησιμοποιήθηκαν τα προγράμματα, Quartus, Matlab, ModelSim και το Vivado. Αυτό έδωσε την ευκαιρία της εξοικείωσης και του εμπλουτισμού των γνώσεων επάνω σε αυτά τα προγράμματα. Επίσης κατανοήθηκε καλύτερα η δομή, η λειτουργία του FPGA, καθώς και η λογική του προγραμματισμού του.

Σημαντικό επίσης είναι ότι μέσω της εργασίας αυτής, έγινε κατανοητό με ποιον τρόπο συνδέεται ο προγραμματισμός του FPGA με τις περιφερειακές θύρες εξόδου και εισόδου που διαθέτει η κάρτα PYNQ-Z1.

Η επεξεργασία της εικόνας των ακμών στην παρούσα εργασία, επιτυγχάνεται με ρυθμό 60 πλαισίων το δευτερόλεπτο, για εικόνα ανάλυσης 1280x720. Ο χρόνος επεξεργασίας είναι ίδιος, είτε η εισερχόμενη εικόνα ήταν μια στατική εικόνα, είτε ήταν παραγόμενο βίντεο. Από αυτό, γίνεται αντιληπτό το τι δυνατότητες παρέχει ένα FPGA και το πόσο αναγκαίο είναι για την πρόοδο της τεχνολογίας, σύμφωνα με τις δυνατότητες που μπορεί να προσφέρει σε διάφορες εφαρμογές.

Τέλος, η εργασία αυτή αναδεικνύει την πλακέτα PYNQ-Z1 ως ένα εύχρηστο εργαλείο ανάπτυξης επιταχυντών αλγορίθμων επεξεργασίας video. Είναι αξιοσημείωτο ότι η κάρτα PYNQ-Z1 είναι σχετικά χαμηλού κόστους, καθώς η τελική τιμή της δεν ξεπερνά τα 200€ και η ακαδημαϊκή τιμή της είναι ακόμη χαμηλότερη.

Ας σημειωθεί, ότι η πλακέτα PYNQ-Z1 προσδίδει και επιπλέον δυνατότητες επεξεργασίας, μέσω του PYNQ-project, που επιτρέπει τον συσχεδιασμό υλικού/λογισμικού, με χρήση της γλώσσας Python.

5.2 Μελλοντική επέκταση

Η μέθοδος εξαγωγής ακμών με την χρήση του φίλτρου Sobel είναι ευρέως διαδεδομένη και αποτελεσματική. Οι αλγόριθμοι ανίχνευσης ακμών τροποποιούνται και εξελίσσονται. Στις ημέρες μας, οι εφαρμογές ανίχνευσης ακμών σε πραγματικό χρόνο εμπλέκουν και την μηχανική ευφυΐα, άλλοτε για εξαγωγή αποτελεσμάτων (π.χ. αναγνώριση αντικειμένων) και άλλοτε για την λήψη αποφάσεων από το σύστημα. Η νέα βιομηχανία που αναπτύσσεται, αυτή των αυτόνομων οχημάτων, με σκοπό την εμπορική χρήση τους, οδηγεί τις εξελίξεις στον τομέα της μηχανικής όρασης. Ο τομέας αυτός είναι δυναμικά αναπτυσσόμενος και αποτελεί τον θεμέλιο λίθο της αυτοκινητοβιομηχανίας των αυτόνομων οχημάτων. Η μηχανική όραση μέσω τις επεξεργασίας της εικόνας, είναι πολύ σημαντική για την λήψη των αποφάσεων ενός αυτόνομου οχήματος.

Ως μελλοντική επέκταση της παρούσας διπλωματικής εργασίας, θα μπορούσε να γίνει η δοκιμή ασύρματης σύνδεσης με ένα υπολογιστή απομακρυσμένο, με σκοπό την περεταίρω επεξεργασία της παραγόμενης εικόνας ακμών για τη λήψη αποφάσεων. Η λήψη αποφάσεων αυτή θα μπορούσε να χρησιμοποιηθεί, είτε για την κίνηση ενός αυτόνομου οχήματος, είτε για την αναγνώριση αντικειμένων, είτε για την αναγνώριση πινακίδων αυτοκινήτων, είτε για χαρτογράφηση χώρου και λοιπές εφαρμογές. Η λήψη της εικόνας θα μπορούσε να γίνει από μια κάμερα απλή ή στερεοσκοπική, η οποία θα είναι τοποθετημένη επάνω σε ένα αυτόνομο όχημα, ή σε ένα στατικό σημείο ή ακόμα και σε ένα drone.

Βιβλιογραφικές Πηγές

[1] E-Class TEI Πειραιά

http://eclass.teipir.gr/openeclass/modules/document/file.php/HYS100/IV.%20%CE%A3%CF%85%CF%83%CF%84%CE%AE%CE%BC%CE%B1%CF%84%CE%B1%20%CE%A0%CF%81%CE%B1%CE%B3%CE%BC%CE%B1%CF%84%CE%B9%CE%BA%CE%BF%CF%8D%20%CE%A7%CF%81%CF%8C%CE%BD%CE%BF%CF%85.pdf

[2] Wikipedia Website

https://en.wikipedia.org/wiki/Whirlwind_I

[3] Wikipedia Website

https://en.wikipedia.org/wiki/Semi-Automatic_Ground_Environment

[4] Wikipedia Website

https://el.wikipedia.org/wiki/%CE%95%CF%80%CE%B5%CE%BE%CE%B5%CF%81%CE%B3%CE%B1%CF%83%CE%AF%CE%B1_%CE%B5%CE%B9%CE%BA%CF%8C%CE%BD%CE%B1%CF%82

[5] Σημειώσεις μαθήματος ρομποτικής όρασης Α εξαμήνου, προγράμματος μεταπτυχιακών σπουδών στην ρομποτική

[6] Μ Δασυγένης, 2015

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiA_pXbqd3vAhWugP0HHUsYBLgQFjAAegQIBhAD&url=https%3A%2F%2Frepository.kallipos.gr%2Fbitstream%2F11419%2F2251%2F1%2Fchapter04.pdf&usg=AOvVaw39Ssx24DfkOFqdCl1CoxeV

[7] Wikipedia site

https://en.wikipedia.org/wiki/Application-specific_integrated_circuit

[8] Wikipedia site

https://el.wikipedia.org/wiki/VHDL

[9] Wikipedia site

https://en.wikipedia.org/wiki/Xilinx_Vivado

[10] Chipstimate

https://www.chipestimate.com/Xilinx-Unveils-Vivado-Design-Suite-for-the-Next-Decade-of-/Xilinx/Technical-Article/2012/06/12

[11] Digilent Website

https://reference.digilentinc.com/programmable-logic/pynq-z1/reference-manual

[12] Wikipedia site

https://el.wikipedia.org/wiki/MATLAB

[13] Intel Website

https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/model-sim.html

[14] Pynq Website

http://www.pynq.io/

[15] YantaVision Website

https://www.yantravision.com/pynq-on-zynq/

[16] Wikipedia site

https://en.wikipedia.org/wiki/Graphics_processing_unit

[17] Stemmer Imaging site

https://www.stemmer-imaging.com/en-pl/technical-tips/image-processing-on-the-graphics-card/

[18] Imaging machine vision Europe site

https://www.imveurope.com/feature/multicore-image-processing

[19] Packt> site

https://subscription.packtpub.com/book/data/9781800207219/1/ch01lvl1sec03/single-board-computers

[20] Packt> site

https://www.ximea.com/support/wiki/apis/Jetson_Image_Processing

[21] Intel Website

https://www.intel.com/content/www/us/en/products/programmable/fpga/new-to-fpgas/resource-center/overview.html

[22] Xilinx Website

https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[23] Irwin Sobel, 2014, History and Definition of the Sobel Operator

[24] Wikipedia Website

https://en.wikipedia.org/wiki/Canny_edge_detector

[25] Wikipedia Website

https://en.wikipedia.org/wiki/CUDA

[26] Wikipedia Website

https://en.wikipedia.org/wiki/OpenCL

[27] Wikipedia Website

https://el.wikipedia.org/wiki/%CE%93%CE%BB%CF%8E%CF%83%CF%83%CE%B1_%CF%80%CE%B5%CF%81%CE%B9%CE%B3%CF%81%CE%B1%CF%86%CE%AE%CF%82_%CF%85%CE%BB%CE%B9%CE%BA%CE%BF%CF%8D

[28] Wikipedia Website

https://en.wikipedia.org/wiki/Prewitt_operator

[29] Image Analysis, Edge Detection, Christophoros Nikou

http://www.cs.uoi.gr/~cnikou/Courses/Image_Analysis/02_Edge_Detection.pdf

[30] Edge Detection, Trucco, Chapt 4 AND Jain et al., Chapt 5

https://eclass.hmu.gr/modules/document/file.php/TP249/AV05_08_EdgeDetection.pdf

[31] Intel Website

https://www.intel.com/content/www/us/en/products/programmable/cyclone-series.html

[32] Intel Website

https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-10/gx.html

[33] Xilinx Website

https://www.xilinx.com/products/boards-and-kits.html

[34] Wikipedia Website

https://en.wikipedia.org/wiki/Field-programmable_gate_array

[35] ElectronicDesign Website, Maria Guerra, May 20, 2016

https://www.electronicdesign.com/technologies/fpgas/article/21801527/the-principles-of-fpgas

[36] Xilinx Website

https://www.xilinx.com/products/design-tools/vivado.html

[37] Project F – FPGA Development Website

https://projectf.io/posts/video-timings-vga-720p-1080p/

[38] Xilinx Website

https://www.xilinx.com/products/boards-and-kits/1-elhabt.html

[39] Xilinx Website

https://www.xilinx.com/products/boards-and-kits/1-hydd4z.html

[40] Pynq Website

http://www.pynq.io/board.html

[41] Aldec Website

https://www.aldec.com/en/company/blog/144--introduction-to-zynq-architecture

[42] Xilinx Website

https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[43] Xilinx Website

https://www.xilinx.com/support/documentation/ip_documentation/processing_system7/v5_3/pg082-processing-system7.pdf

[44] Ioannis Kalomiros, "Basic Video processing with PYNQ-Z1 board", IHU, 2021 (personal communication)

[45] Σημειώσεις μαθήματος «Σχεδίαση Συστημάτων Υψηλών Επιδόσεων», του μεταπτυχιακού προγράμματος στην Ρομποτική. ΔΙ.ΠΑ.Ε. Σέρρες

[46] Mathworks Website

https://www.mathworks.com/?s_tid=gn_logo

[47] Basic Video in-out with PYNQ-Z1_part1, Ioannis Kalomiros Dpt. of Computers, Informatics and Telecommunications, International Hellenic University, February 2021, v 1.0