

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ - ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ



UNIVERSITY OF
P A T R A S
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

Τομέας Ηλεκτρονικής & Υπολογιστών
Εργαστήριο Σχεδιασμού Ολοκληρωμένων Κυκλωμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας
Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

Χρήστου-Αλέξανδρου Νικολακάκη
Αριθμός Μητρώου: 7582

Θέμα

**Ανάπτυξη Αρχιτεκτονικών υλικού για αναγνώριση
ακμών σε εικόνες με τεχνικές σύνθεσης υψηλού
επιπέδου (High level synthesis)**

Επιβλέπων

Παλιούράς Βασίλειος
Αναπληρωτής Καθηγητής

Αριθμός Διπλωματικής Εργασίας:

Πάτρα, Οκτώβριος 2017

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η διπλωματική εργασία με θέμα

Ανάπτυξη Αρχιτεκτονικών υλικού για αναγνώριση ακμών σε εικόνες με τεχνικές σύνθεσης υψηλού επιπέδου (High level synthesis)

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας
Υπολογιστών

Χρήστου-Αλέξανδρου Νικολακάκη

(Α.Μ.: 7582)

παρουσιάστηκε δημόσια και εξετάστηκε στο τμήμα Ηλεκτρολόγων
Μηχανικών και Τεχνολογίας Υπολογιστών στις

18/10/2017

Ο Επιβλέπων

Ο Διευθυντής του Τομέα

Παλιουράς Βασίλειος
Αναπληρωτής Καθηγητής

Χούσος Ευθύμιος
Καθηγητής

Στοιχεία διπλωματικής εργασίας

Θέμα: Ανάπτυξη Αρχιτεκτονικών υλικού για αναγνώριση ακμών σε εικόνες με τεχνικές σύνθεσης υψηλού επιπέδου (High level synthesis)

Φοιτητής: Χρήστος-Αλέξανδρος Νικολακάκης

Ομάδα επίβλεψης:
Αναπληρωτής Καθηγητής Παλιουράς Βασίλειος

Εργαστήρια:
Εργαστήριο Σχεδιασμού Ολοκληρωμένων Κυκλωμάτων

Η εργασία αυτή γράφτηκε στο \LaTeX και χρησιμοποιήθηκε η γραμματοσειρά GFS Didot του Greek Font Society.

Περίληψη

Εφαρμογές με υψηλές υπολογιστικές απαιτήσεις, όπως για παράδειγμα αλγόριθμοι μηχανικής όρασης, απαιτούν επιδόσεις που δεν μπορούν να ικανοποιηθούν από ηλεκτρονικούς υπολογιστές χαμηλού κόστους καθώς απαιτούν σημαντικό υπολογιστικό φόρτο, εξαιτίας της αυξημένης πολυπλοκότητάς τους αλλά και του όγκου των δεδομένων που χρησιμοποιούν. Για το λόγο αυτό η υλοποίησή τους πραγματοποιείται σε υλικό ειδικού σκοπού, όπως για παράδειγμα σε application-specific integrated circuits (ASIC)s ή se Field Programmable Gate Array circuits (FPGA).

Η υλοποίηση σε εξειδικευμένο υλικό απαιτεί την απασχόληση πολυπληθούς και ιδιαίτερα εξειδικευμένου προσωπικού καθώς και μεγάλο χρόνο ανάπτυξης. Για το λόγο αυτό είναι απαραίτητο οι διαδικασίες υλοποίησής τους αφενός να συντομευτούν πολύ, αφετέρου να μειωθεί το κόστος ανάπτυξής τους.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη ενός συνδυασμένου συστήματος hardware/software και η υλοποίηση ενός ενσωματωμένου συστήματος για την υλοποίηση ενός τέτοιου απαιτητικού αλγορίθμου, συγκεκριμένα του Canny Edge Detector. Σε αντίθεση με το συνηθισμένο τρόπο αντιμετώπισης σχεδιασμού σε FPGA, θα αναπτύξουμε τον αλγόριθμο με τη βοήθεια του εργαλείου Vivado High-level Synthesis (HLS) της Xilinx, το οποίο μετασχηματίζει μία περιγραφή από γλώσσα γενικού σκοπού, υψηλού επιπέδου όπως η C σε μία περιγραφή υλικού (RTL).

Εκμεταλλευόμενοι τις δυνατότητες βελτιστοποίησης του Vivado HLS και γενικότερα της πλατφόρμας Vivado, μπορούμε να ελαχιστοποιήσουμε κατά πολύ το χρόνο ανάπτυξης και αποσφαλμάτωσης του αλγορίθμου μας και της υλοποίησης του συστήματος. Η συσκευή που χρησιμοποιήθηκε είναι το σύστημα ανάπτυξης της εταιρείας Xilinx, ZC702 με τον Zynq-7000 All Programmable SoC.

Λέξεις κλειδιά: FPGA, Σύνθεση υψηλού επιπέδου, Canny Edge Detector, Zynq-7000 All Programmable SoC, Ανίχνευση ακμών, Vivado HLS, Vivado, ARM

Abstract

High computational applications, like algorithms of Computer Vision, require a lot of performance that cannot be provided by low cost personal computers. Such applications, due to their high complexity and the size of the data they process, put a significant strain on the computing infrastructure executing them. Therefore, they are mainly implemented on application-specific integrated circuits (ASIC) or Field Programmable Gate Array Circuits (FPGA).

Their implementation requires a large amount of highly trained engineers and large amount of time. However, both the cost and the duration of their implementation must be minimized.

In this diploma thesis, we focus on providing a hardware/software design and an embedded system that implements a resource hungry Computer Vision algorithm: Canny Edge Detector. Comparing to the usual way of dealing with the implementation of such algorithms we chose to go with Xilinx's Vivado High-Level Synthesis (HLS). After developing the algorithm in a high-level programming language, for example C/C++, HLS synthesizes it and produces a RTL hardware design.

Taking advantage of HLS' optimizations and Xilinx's Vivado platform we observe that the development, debugging, and implementation times can be significantly reduced. The development board used in this thesis is the ZC702 that features a Zynq-7000 All Programmable SoC produced by Xilinx.

Keywords: FPGA, High-level Synthesis, Vivado HLS, Canny Edge Detector, Zynq-7000 All Programmable SoC, Edge Detection, Vivado HLS, Vivado, ARM

Ευχαριστίες

Θα ήθελα εξαρχής να ευχαριστήσω ιδιαίτερα τον Καθηγητή του Τμήματος κ. Παλιούρά Βασίλειο, τόσο για την εμπιστοσύνη του να μου αναθέσει την εκπόνηση της παρούσας Διπλωματικής Εργασίας όσο και τη συνεχή και αμέριστη βοήθεια και υποδείξεις του. Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή κ. Σκόδρα για την υποστήριξη και τη γνώση που μου παρείχε κατά τη διάρκεια των σπουδών μου σε συναφή εκπαιδευτικά αντικείμενα. Θερμότατες ευχαριστίες οφείλω στους φίλους μου για την ηθική συμπαράστασή τους αλλά και τη βοήθειά τους σε κρίσιμα σημεία της παρούσας εργασίας. Τέλος θα ήθελα από εδώ να ευχαριστήσω τους γονείς μου που ήταν δίπλα μου καθόλη τη διάρκεια της φοίτησής μου.

Περιεχόμενα

Περίληψη	vii
Abstract	ix
Περιεχόμενα	xiii
Κατάλογος Σχημάτων	xvii
Ακρωνύμια	xix
Ορολογία	xxi
1 Εισαγωγή	1
1.1 Εισαγωγή στα FPGA	1
1.1.1 Τι είναι το FPGA & Σύντομη ιστορική αναδρομή	1
1.1.2 Εφαρμογές	4
1.1.3 Σχεδιασμός υλικού & Προγραμματισμός FPGA	4
1.2 Περιγραφή & Στόχοι της εργασίας	5
1.3 Διάρθρωση της διπλωματικής	6
1.4 Behind the project	7
2 Υλικό & Λογισμικό	9
2.1 Αρχιτεκτονική των FPGA	9
2.1.1 LUT	11
2.1.2 Flip Flops	12
2.1.3 DSP48	12
2.1.4 BRAM & Άλλες μνήμες	14
2.2 Zynq-7000 ApSoc	15
2.2.1 Application Processing Unit (APU)	18
2.3 ZC702 Evaluation Kit	19
2.3.1 Διαμόρφωση συσκευής	21
2.3.2 USB-to-UART Bridge	21

2.3.3 AXI4	21
2.4 Software	22
2.4.1 Εισαγωγή	22
2.4.2 Γλώσσα προγραμματισμού VHDL	24
2.4.3 HLS	24
2.4.3.1 Επισκόπηση	24
2.4.3.2 Scheduling & Binding	26
2.4.3.3 Design Flow	27
2.4.3.4 Βελτιστοποίηση	28
2.4.3.5 Περιορισμοί HLS	32
3 Υπολογιστική όραση	35
3.1 Ορισμός	35
3.2 Ιστορική αναδρομή	36
3.3 Εφαρμογές	37
3.4 Μέθοδοι συστήματος υπολογιστικής όρασης	39
3.5 Υπολογιστική όραση & FPGAs	40
4 Αλγόριθμος Αναγνώρισης Ακμών Canny	43
4.1 Κατάτμηση εικόνων	43
4.2 Εισαγωγή στην αναγνώριση ακμών	43
4.2.1 Προσεγγίσεις	45
4.3 Ανιχνευτής ακμών Canny	45
4.3.1 Ανάπτυξη του αλγορίθμου	45
4.3.2 Διαδικασία ανίχνευσης ακμών	46
4.3.3 Παραδείγματα εφαρμογής αλγορίθμου	52
5 Υλοποίηση	55
5.1 Περιγραφή & Ανάλυση Προβλήματος	55
5.2 HLS	56
5.2.1 Δομή κώδικα & Top Function	57
5.2.2 Testbench	59
5.2.3 Δημιουργία Project & Solutions	59
5.2.4 Προσομοίωση με κώδικα C	60

5.2.5 Σύνθεση	60
5.2.6 RTL Co-Simulation	60
5.2.7 IP Export	62
5.2.8 TCL	62
5.3 Vivado IP Integrator	63
5.3.1 Δημιουργία Project	64
5.3.2 Δημιουργία Σχεδίου & Εισαγωγή IP	64
5.3.3 Παραμετροποίηση, Διασύνδεση & Επαλήθευση	65
5.3.4 Σύνθεση, bitstream & εξαγωγή στο SDK	67
5.4 SDK	70
5.4.1 Απαραίτητα αρχεία	70
5.4.2 Δομή κώδικα	71
6 Βελτιστοποίηση & Αποτελέσματα	75
6.1 Βελτιστοποίηση	75
6.1.1 Top συνάρτηση	76
6.1.2 Γκαουσιανό φίλτρο	77
6.1.3 grad	77
6.1.4 edgeID	78
6.2 Αποτελέσματα	78
6.2.1 Αποτελέσματα υλοποίησης	79
6.2.2 Αποτελέσματα επεξεργασίας	81
7 Συμπεράσματα	83
7.1 Συμπεράσματα	83
7.2 Μελλοντικές εργασίες & βελτιώσεις	84
Βιβλιογραφία	89

Κατάλογος Σχημάτων

1.1	FPGA Eras	2
2.1	Βασική αρχιτεκτονική FPGA[1]	9
2.2	Δομή Configurable logic block (CLB) [2]	11
2.3	Δομή LUT [3]	11
2.4	Flip-Flop [4]	12
2.5	DSP48 Slice [5]	13
2.6	Dual Port BRAM [6]	14
2.7	Addressable Shift Register [6]	15
2.8	Programming Logic [8]	17
2.9	Zynq®-7000 All Programmable SoC Family [9]	17
2.10	Zynq APU [8]	18
2.11	APU Block diagram [8]	18
2.12	ZC702 Evaluation kit [10]	20
2.13	Λογικό διάγραμμα υψηλού επιπέδου [11]	20
2.14	Xilinx - Χρόνος σχεδιασμού vs. Απόδοση εφαρμογής με RTL [12]	23
2.15	Xilinx - Χρόνος σχεδιασμού vs. Απόδοση εφαρμογής με HLS [12]	24
2.16	Επίπεδα αφαιρετικότητας σε σχεδιασμούς FPGA [8]	25
2.17	High-level synthesis vs RTL στο Gasjki-Kuhn Y-chart [13]	26
2.18	Ροή scheduling & binding [8]	27
2.19	Ροή σχεδιασμού με HLS [8]	27
2.20	Διαμερισμός υπολογιστικών σταδίων σε μικρότερα μέσω pipelining [8]	29
2.21	Latency & ρυθμοαπόδοση μετά το pipelining [8]	30
2.22	dataflow βελτιστοποίηση [14]	31
3.1	Υπολογιστική όραση & σχετικοί τομείς [17]	36
4.1	Μετατροπή σε grayscale, μέσος όρος & μέθοδος Luma	47
4.2	Αποτελέσματα Canny για $thresh = 0.2$	49
4.3	Αποτελέσματα Canny για $thresh = 0.2$	52
4.4	Αποτελέσματα Canny για $thresh = 0.5$	52

5.1 Ροή αλγορίθμου Canny Edge Detection	57
5.2 125 × 125 Grayscale Lena	60
5.3 Ροή RTL CO-SIM [15]	61
5.4 Βασική ροή σχεδιασμού στο Vivado IPI	63
5.5 Βασικά IP Block	65
5.6 Διασύνδεση βασικών IP Blocks	66
5.7 Ρύθμιση AXI Direct Memory Access	67
5.8 Sources window	68
5.9 Πλήρες σύστημα	69
5.10 Ροή εφαρμογής ARM	70
6.1 Ποσοστό αξιοποίησης συνολικών πόρων	81
6.2 Εκτίμηση κατανάλωσης	81
6.3 Εικόνα #1	82
6.4 Εικόνα #2	82
6.5 Εικόνα #3	82

Ακρωνύμια

CLB	Configurable logic block xvii
FPGA	Field Programmable Gate Array 1
ASIC	Application-specific Integrated Circuit 1
PROM	Programmable Read-Only Memory 1
PLD	Programmable Logic Device 2
CMOS	Complementary metal–oxide–semiconductor 2
EPROM	Erasable Programmable Read-Only Memory 2
SOC	System-on-Chip 3
IP	Intellectual Property 3
AP SoC	All-Programmable SoC 3
EDA	Electronic design automation 4
HLS	High Level Synthesis 5
LUT	Look-up Table 10
PLL	Phase-locked loop 10
PS	Processing System 15
PL	Programmable Logic 15
ASSP	application specific standard produc 15
APU	Application processing unit 18
ACP	Accelerator Coherency Port 18
UART	Universal Asynchronous Receiver-Transmitter 19
MPSoC	multiprocessor system-on-chip 22

Ορολογία

netlist	Περιγραφή της συνδεσιμότητας ενός ηλεκτρονικού κυκλώματος 4
throughput	Μέγιστος ρυθμός παραγωγής ή μέγιστος ρυθμός επεξεργασίας δεδομένων 10
DSP	Ψηφιακή επεξεργασία σημάτων 12

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή στα FPGA

Στο παρόν κεφάλαιο θα παραθέσουμε μερικές πληροφορίες σχετικά με το τι είναι ένα Field Programmable Gate Array (FPGA), την ιστορική εξέλιξή του και θα αναφερθούμε στις διάφορες εφαρμογές του στη βιομηχανία. Θα επιχειρήσουμε ακόμη μια σύντομη ανασκόπηση στις διάφορες καινοτομίες που έχουν συμβεί τα τελευταία χρόνια και βασίζονται στις ιδιαίτερα επιτυχημένες μονάδες FPGAs.

1.1.1 Τι είναι το FPGA & Σύντομη ιστορική αναδρομή

Το FPGA είναι ένας τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος σχεδιασμένο έτσι ώστε να διαμορφώνεται από τον πελάτη μετά την κατασκευή του, εξού και το όνομά του - field programmable. Η διαμόρφωση του γίνεται συνήθως με κάποια γλώσσα περιγραφής υλικού, παρόμοια με αυτές που χρησιμοποιούνται στα Application-specific Integrated Circuit (ASIC).

Τα FPGA διαθέτουν έναν μεγάλο αριθμό τυποποιημένων προγραμματιζόμενων blocks και μια ιεραρχία από διαμορφώσιμες διασυνδέσεις που επιτρέπουν στα blocks να συνδέονται μεταξύ τους. Λογικά blocks μπορούν να διαμορφωθούν κατά τέτοιον τρόπο ώστε να πραγματοποιούν σύνθετες πράξεις ή απλές λογικές πύλες (AND, XOR, κά). Πολλά FPGA περιλαμβάνουν και διάφορες άλλες ψηφιακές λειτουργίες όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL, DSP slices κα. Κατά τον προγραμματισμό του FPGA ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία.

Η βιομηχανία των FPGA ξεκίνησε από την Programmable Read-Only Memory

(PROM) και τις Programmable Logic Devices (PLDs). Τόσο οι PROMs όσο και οι PLDs είχαν την επιλογή να προγραμματίζονται σε παρτίδες στο εργοστάσιο ή στο πεδίο. Η προγραμματιζόμενη λογική, ωστόσο, ήταν hard-wired μεταξύ των λογικών πυλών.

Η έννοια του FPGA αναπτύχθηκε στα τέλη της δεκαετίας του 1980 με την οικογένεια των XC2064TM FPGA της Xilinx. Την ίδια εποχή η Altera ανέπτυσσε τη δικιά της προγραμματιζόμενη συσκευή, την EP1200. Η τεχνολογία της Altera κατασκευαζόταν σε λιθογραφία 3-μμ Complementary metal–oxide–semiconductor (CMOS) Erasable Programmable Read-Only Memory (EPROM) η οποία απαιτούσε υπεριώδη ακτινοβολία για να επαναπρογραμματιστεί σε αντίθεση με της Xilinx που χρησιμοποιούσε static RAM (SRAM) αλλά απαιτούσε EPROM για την αποθήκευση του προγραμματισμού.

Ο ιδρυτής της Xilinx, Ross Freeman, υποστήριξε ότι με τη συνεχώς αναπτυσσόμενη τεχνολογία, η τιμή των τρανζίστορ θα μειωνόταν συνεχώς και θα μπορούσαν να προσφέρουν μεγαλύτερη ευελιξία στον προγραμματισμό των FPGA. Αυτό σηματοδότησε την αρχή της αγοράς των FPGA από μεγάλες εταιρίες όπως για παράδειγμα είναι οι Xilinx, Altera, Actel, IBM, Toshiba κα. Σχετικές έρευνες δείχνουν ότι το μερίδιο αγοράς των FPGA αυξάνεται συνεχώς. Το 2006 ήταν 4.5 δις δολλάρια με προβλέψεις ότι θα σπάσει το φράγμα των 10 δις μετά το 2020. Αυτή την εποχή η αγορά κυριαρχείται από 2 εταιρίες, τη Xilinx και την Altera. Το σημαντικό όμως είναι ότι το FPGA έχει ωριμάσει σε ένα πλήρες σύστημα και έχει ξεπεράσει τα ASICs. Η εξέλιξη των FPGA ήταν ραγδαία. Στον ακόλουθο πίνακα παρουσιάζονται οι τρεις διακριτές εποχές τους.

		Technology is limited
1984-1991	Invention	FPGAs are much smaller than the application problem size Design automation is secondary, architecture efficiency is key
1992-1993	Expansion	FPGA Size approaches the problem size Each of design becomes critical
2000-2007	Accumulation	FPGAs are larger than the typical problem size Logic capacity limited by I/O bandwidth

Σχήμα 1.1: FPGA Eras

Παροδοσιακά, τα FPGAs ήταν αργά, είχαν μεγάλη κατανάλωση ισχύος και παρείχαν μικρότερη λειτουργικότητα από τα ASICs. Ωστόσο, στη σημερινή εποχή έχουν εξελιχθεί δραματικά και παρέχουν λύσεις που τα καθιστούν ιδανικότερη λύση. Μπορούν να επιτύχουν:

- Χαμηλή κατανάλωση ισχύος
- Αυξημένη ταχύτητα
- Χαμηλό κόστος
- 'On-the-fly' διαμόρφωση

Η εξέλιξη αυτή δε θα μπορούσε να έχει επιτευχθεί χωρίς τη βοήθεια της ραγδαίας αύξησης του αριθμού των λογικών πυλών των FPGAs.

- 1982: 8192 πύλες
- 1987: 9,000 πύλες
- 1992: 600,000 πύλες
- Early 2000s: Εκατομμύρια

Ιδιαίτερη μνεία πρέπει να γίνει στη νέα αρχιτεκτονική που αναπτύσσεται τα τελευταία χρόνια, τα System-on-Chip (SOC) class FPGAs. Ένα SOC είναι ένα chip το οποίο περιλαμβάνει έναν ή περισσότερους πυρήνες – μικροεπεξεργαστές και/ή DSP μαζί με μνήμη και διάφορα περιφερειακά. Με τη πάροδο του χρόνου οι δυνατότητες των FPGA έχουν παρουσιάσει δραματική αύξηση και ένα σύγχρονο FPGA μπορεί να περιλαμβάνει χιλιάδες αθροιστές, DSP slices κα. όμως το πρόβλημα που εμφανίζεται είναι ότι πλέον δεν αντανακλούν τις δυνατότητες και τη λειτουργικότητα των σύγχρονων προγραμματιζόμενων συσκευών. Για το λόγο αυτό δημιουργήθηκαν τα SOC class FPGAs τα οποία μπορούν να περιέχουν έναν η περισσότερους soft/hard πυρήνες ARM μαζί με περιφερειακά και μνήμη συνδυάζοντας κατ' αυτόν τον τρόπο την απόδοση και την εξοικονόμηση ενέργειας των Intellectual Properties (IPs) με την ευελιξία του προγραμματισμού του επεξεργαστή ARM. Τα All-Programmable SoC (AP SoC) όπως τα ονομάζει η Xilinx είναι ιδανικά για:

- Ελαχιστοποίηση κατανάλωσης, κόστους, μεγέθους πλακέτας ενσωματώνοντας διακριτούς μικροεπεξεργαστές και συναρτήσεις DSP σε ένα FPGA
- Βελτιστοποίηση της απόδοσης του συστήματος με τη βοήθεια μίας high-bandwidth διασύνδεσης μεταξύ του επεξεργαστή και του FPGA
- Βελτίωση της διαφορετικότητας του τελικού προϊόντος προσαρμόζοντας κατάλληλα τόσο το υλικό όσο και το λογισμικό
- Ανάπτυξη εφαρμογών συμβατών με την αρχιτεκτονική ARM με απαράμιλλο

έλεγχο και παραγωγικότητα κάνοντας χρήση του FPGA-Adaptive debugging

1.1.2 Εφαρμογές

Τα FPGA μπορούν να επιλύσουν οποιοδήποτε πρόβλημα είναι δυνατόν να επιλυθεί υπολογιστικά. Η πρόταση αυτή έχει τεκμηριωθεί από το γεγονός ότι έχει επιτευχθεί ο σχεδιασμός ένος soft μικροεπεξεργαστή όπως για παράδειγμα ο Microblaze της Xilinx. Το πλεονέκτημα τους βρίσκεται στο ότι ορισμένες φορές είναι σημαντικά πιο γρήγοροι για συγκεκριμένες εφαρμογές λόγω της παραλληλίας και της βελτιστοποίησης που προσφέρουν. Ένα μεγάλο πλεονέκτημα των FPGA είναι ότι μπορούν να χρησιμοποιηθούν σαν επιταχυντές υλικού, όπου ο σχεδιαστής μπορεί να το χρησιμοποιήσει για να επιταχύνει **συγκεκριμένα** κομμάτια του αλγορίθμου και να μοιράσει μέρος του υπολογιστικού φόρτου με κάποιον μικροεπεξεργαστή γενικού σκοπού. Συνήθεις εφαρμογές των FPGA είναι στους τομείς που ακολουθούν:

- Aerospace & Defense
- Medical Electronics
- Audio
- Automotive
- Consumer Electronics
- Data Centers
- High Performance Computing
- Industrial κά.

1.1.3 Σχεδιασμός υλικού & Προγραμματισμός FPGA

Για την περιγραφή της συμπεριφοράς ενός FPGA, ο σχεδιαστής παρέχει ένα πρόγραμμα γραμμένο σε κάποια γλώσσα περιγραφής υλικού (HDL) όπως για παράδειγμα σε VHDL ή Verilog. Μετά τη συγγραφή του κώδικα, με τη βοήθεια Electronic design automation (EDA) εργαλείων ο χρήστης δημιουργεί ένα netlist το οποίο μεταφέρει στο FPGA, συνήθως μέσω ιδιόκτητου λογισμικού της εκάστοτε εταιρίας. Ο σχεδιαστής τότε θα επιβεβαιώσει τη σωστή λειτουργία πραγματοποιώντας χρονική ανάλυση, προσομοίωση και άλλες διαφορτικές μεθοδολογίες. Όταν επιβεβαιωθεί τε-

λικώς η σωστή λειτουργία, παράγεται το bitstream το οποίο χρησιμοποείται για τον (επανα)προγραμματισμό του FPGA. Το αρχείο αυτό μετφέρεται στο FPGA συνήθως μέσω της διεπαφής (JTAG).

Τα τελευταία χρόνια έχει αναπτυχθεί μία νέα τεχνική προγραμματισμού των FPGA για να αντικαταστήσει τον σχεδιασμό σε HDL που ονομάζεται High Level Synthesis (HLS). Τα εργαλεία HLS αναλύουν μία υπάρχουσα εφαρμογή γραμμένη σε γλώσσα υψηλού επιπέδου - συνήθως C, C++, systemC και παράγουν την αντίστοιχη HDL. Το HLS έχει προοπτικές για να είναι το επόμενο σημαντικό βήμα για την ανάπτυξη προγραμμάτων σε FPGA.

Στην υποενότητα αυτή έγινε μια σύντομη αναφορά στον σχεδιασμό και προγραμματισμό των FPGA. Στο επόμενο κεφάλαιο θα εμβαθύνουμε περισσότερο στη χρήση σχεδιαστικών μεθοδολογιών υψηλού επιπέδου.

1.2 Περιγραφή & Στόχοι της εργασίας

Όπως αναφέρθηκε στην προηγούμενη ενότητα ένας τρόπους ανάπτυξης σύνθετων εφαρμογών που απαιτούν μεγάλη επεξεργαστική ισχύ σε πραγματικό χρόνο με λογικό κόστος είναι η χρήση μονάδων FPGA με τον κατάλληλο προγραμματισμό τους σε ειδικές γλώσσες (VHDL, Verilog). Σήμερα όμως, σε όλους τους κλάδους της βιομηχανίας, προκύπτουν ολοένα και περισσότερες τέτοιες εφαρμογές που μάλιστα πρέπει να υλοποιούνται σε ελάχιστο χρόνο κυρίως λόγω του ανταγωνισμού αλλά και των απαιτήσεων των χρηστών. Η ανάπτυξη αυτών των εφαρμογών σε FPGA απαιτεί απασχόληση μεγάλου πλήθους απολύτως εξειδικευμένου προσωπικού για μεγάλο χρονικό διάστημα μέχρι την υλοποίησή τους, γεγονός που προκάλεσε την ανάπτυξη της πλατφόρμας Vivado HLS και τον προγραμματισμό σε κάποια διαδεδομένη γλώσσα υψηλού επιπέδου.

Διαφαίνεται έτσι ότι η λύση με την υλοποίηση της εφαρμογής μέσω της πλατφόρμας Vivado HLS εμφανίζει πολλά πλεονεκτήματα (μικρός χρόνος ανάπτυξης, χαμηλότερο κόστος κλπ). Αυτό σημαίνει ότι είναι απαραίτητη η όσο το δυνατόν μεγαλύτερη διάδοση και εξοικείωση των νέων μηχανικών στην όλη διαδικασία χρήσης της πλατφόρμας HLS αλλά και η ολοκλήρωση (integration) των παραγόμενων αποτελεσμάτων σε FPGA μέσω κατάλληλου υλικού και λογισμικού. Η παρούσα εργασία αυτόν

ακριβώς το στόχο έχει. Να περιγράψει:

- την ανάπτυξη και την end-to-end υλοποίηση μιας απαιτητικής εφαρμογής Υπολογιστικής Όρασης σε FPGA με τη χρήση της πλατφόρμας Vivado HLS
- τη διαπίστωση και καταγραφή τυχόν προβλημάτων και δυσκολιών που έχουν προκύψει
- τον έλεγχο των παραγόμενων αποτελεσμάτων και την υλοποίηση τεχνικών βελτίωσης της λειτουργίας του συστήματος
- τα σημεία στα οποία η εργασία μπορεί να αναπτυχθεί ή να βελτιωθεί μελλοντικά

1.3 Διάρθρωση της διπλωματικής

Το κείμενο της διπλωματικής είναι διαχωρισμένο σε 8 κεφάλαια. Αρχικά, γίνεται μια εισαγωγή στα FPGA, στη συνέχεια παρουσιάζεται η υπολογιστική όραση και στο τέλος γίνεται αναλυτική περιγραφή της υλοποίησης και παρουσίαση των αποτελεσμάτων και συμπερασμάτων.

Κεφάλαιο 1

Γίνεται μια εισαγωγή στα FPGA και στην τεχνολογία τους, στις πολλαπλές εφαρμογές τους καθώς και στον τρόπο προγραμματισμού τους. Παρουσιάζονται επίσης οι στόχοι της διπλωματικής εργασίας.

Κεφάλαιο 2

Παρουσιάζεται η βασική αρχιτεκτονική των σύγχρονων FPGA καθώς και της πλατφόρμας που θα χρησιμοποιηθεί κατα τη διάρκεια της εργασίας. Αναλύεται η ανάπτυξη λογισμικού για την περιγραφή του υλικού με έμφαση στη High Level Synthesis.

Κεφάλαιο 3

Γίνεται μία βάση ανάλυση της υπολογιστικής όρασης και παρουσιάζονται οι εφαρμογές της, η ροή ανάπτυξης εφαρμογών και η σχέση της με τα FPGA.

Κεφάλαιο 4

Αρχικά, γίνεται μία εισαγωγή στην αναγνώριση ακμών και έπειτα αναλύεται η μέθοδος αναγνώρισης ακμών με τη μέθοδο του Canny. Παρουσιάζονται όλα τα απαραίτητα βήματα και προσεγγίσεις.

Κεφάλαιο 5

Παρουσιάζεται η πλήρης υλοποίηση του ενσωματωμένου συστήματος. Γίνεται περιγραφή της ροής στο HLS, ο σχεδιασμός του συστήματος με τα απαιτούμενα περιφερειακά και τέλος αναλύεται η εφαρμογή του επεξεργαστή και η εκτέλεση του προγράμματος.

Κεφάλαιο 6

Παρατίθενται διάφορες βελτιστοποιήσεις που έγιναν κατά την ανάπτυξη του αλγορίθμου και τα αποτέλεσματα της υλοποίησης.

Κεφάλαιο 7

Στο κεφάλαιο αυτό αναλύουμε τα συμπεράσματα που προέκυψαν από την υλοποίηση και δίνονται απαντήσεις σε ερωτήματα που προέκυψαν κατά τη διάρκεια της εργασίας. Γίνεται επίσης αναφορά στις διάφορες δυσκολίες που εμφανίστηκαν. Τέλος, γίνεται λόγος και στις μελλοντικές εργασίες που μπορούν να πραγματοποιηθούν.

1.4 Behind the project

Όπως αναφέρθηκε και στην προηγούμενη υποενότητα, το HLS είναι μια συνεχώς αναπτυσσόμενη τεχνολογία που μπορεί να αντικαταστήσει τον παραδοσιακό τρόπου σχεδιασμού και προγραμματισμού των FPGA. Η ευκαιρία να μελετήσουμε σε βάθος και να βγάλουμε συμπεράσματα για την τεχνολογία αυτή αποτέλεσε και το βασικότερο κίνητρο για την πραγματοποίηση της συγκεκριμένης διπλωματικής εργασίας.

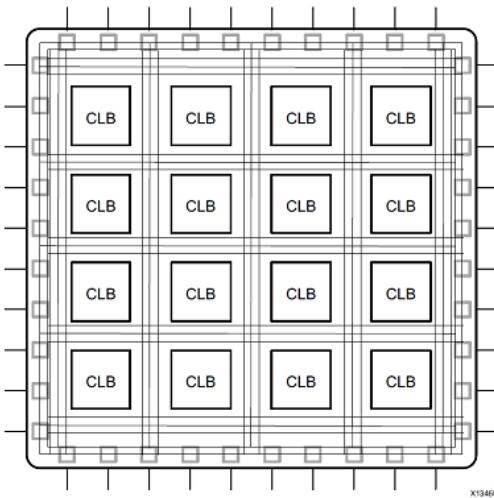
Ελπίζουμε ότι η εργασία αυτή θα αποτελέσει κίνητρο και το πρώτο σκαλί για λεπτομερέστερη μελέτη της διαδικασίας προγραμματισμού των FPGAs μέσω της γλώσσας υψηλού επιπέδου HLS ώστε η ανάπτυξη σύνθετων ηλεκτρονικών μονάδων να γίνει απλούστερη και οικονομικότερη.

Κεφάλαιο 2

Υλικό & Λογισμικό

2.1 Αρχιτεκτονική των FPGA

Μια γενική άποψη ενός FPGA board αποκαλύπτει αρκετά ηλεκτρονικά εξαρτήματα που συνεργάζονται για να υλοποιήσουν τα επιθυμητά ψηφιακά κυκλώματα. Η κύρια λειτουργική μονάδα ενός FPGA είναι το CLB. Αυτά είναι οι κύριοι λογικοί πόροι που επιτρέπουν την υλοποίηση ακολουθιακών αλλά και συνδυαστικών κυκλωμάτων. Κάθε πλακέτα περιλαμβάνει έναν μεγάλο αριθμό CLB, τα οποία οργανώνονται σε μία διδιάστατη διάταξη συνδεδεμένη μέσω οριζόντιων και κάθετων καναλιών.



Σχήμα 2.1: Βασική αρχιτεκτονική FPGA[1]

Υπάρχουν επίσης αρκετά I/O blocks που επιτρέπουν στην συσκευή να επικοινωνεί με το περιβάλλον.

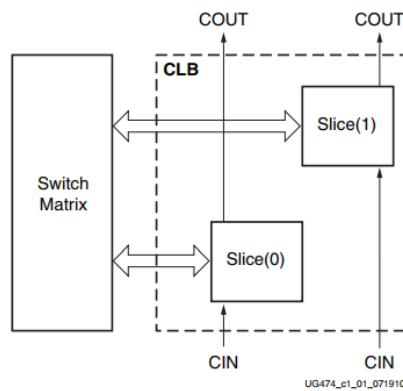
Κάθε CLB αποτελείται από slices, καθένα από τα οποία περιλαμβάνει κάποιον αριθμό λογικών κελιών. Ο αριθμός αυτός εξαρτάται από το εκάστοτε μοντέλο FPGA και κατά συνέπεια τον κατασκευαστή του. Κάθε λογικό κελί αποτελείται από τα ακόλουθα:

- Look-up Table (LUT) - Το στοιχείο αυτό πραγματοποιεί λογικές πράξεις.
- Flip-Flop (FF) - Είναι καταχωρητές που αποθηκεύουν τα αποτελέσματα των LUTs.
- Διασυνδέσεις (καλώδια)
- Input/Output (I/O) pads - Είναι φυσικές πύλες για τη μεταφορά δεδομένων από-/προς το FPGA.

Παρ' όλο που η δομή αυτή επαρκεί για την υλοποίηση οποιουδήποτε αλγορίθμου, η αποδοτικότητα της υλοποίησης είναι περιορισμένη από άποψη υπολογιστικού throughput, απαιτούμενων πόρων καθώς και συχνότητας του ρολογιού. Για το λόγο αυτό, οι σύγχρονες αρχιτεκτονικές FPGA ενσωματώνουν τα βασικά στοιχεία μαζί με επιπλέον υπολογιστικά και blocks αποσθήκευσης μνήμης που αυξάνουν την υπολογιστική ισχύ καθώς και την απόδοση της. Τα στοιχεία αυτά, τα οποία αναλύονται στην επόμενη ενότητα, είναι:

- Ενσωματωμένες μνήμες για κατανεμημένη αποθήκευση δεδομένων
- Phase-locked loop (PLL) για την επίτευξη διαφορετικών συχνοτήτων ρολογιού του FPGA fabric
- Σειριακοί πομποδέκτες υψηλής ταχύτητας
- Ελεγκτές μνήμης εκτός του chip
- Multiply-accumulate blocks

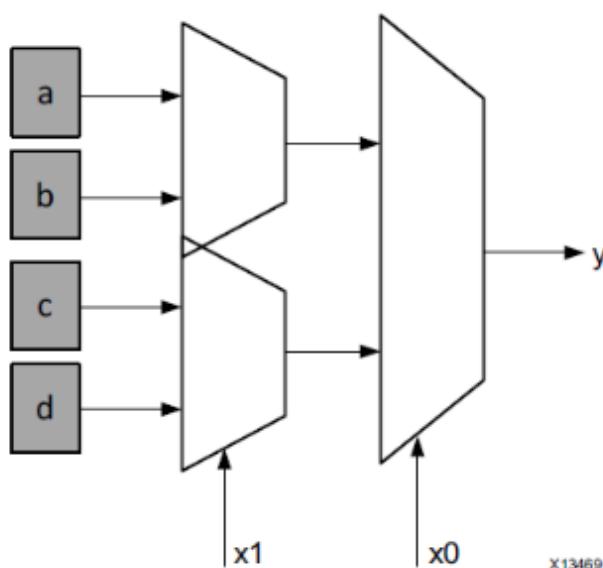
Ο συνδυασμός όλων των CLBs είναι υπεύθυνος για την υλοποίηση οποιασδήποτε εφαρμογής. Υπεύθυνα όμως για την πραγματοποίηση πολλαπλών σχεδίων είναι τα ‘switch matrices’, που διαμορφώνονται κάθε φορά ανάλογα με το σχέδιο που υλοποείται. Μια λεπτομερέστερη άποψη των CLB και των switch matrices φαίνεται στο ακόλουθο σχήμα:



Σχήμα 2.2: Δομή CLB [2]

2.1.1 LUT

Το LUT είναι το βασικό κατασκευαστικό block των FPGA και είναι ικανό να υλοποιεί οποιαδήποτε λογική συνάρτηση N boolean μεταβλητών. Βασικά, το στοιχείο αυτό αποτελεί έναν πίνακα αληθείας όπου διάφοροι συνδυασμοί των εισόδων υλοποιούν διαφορετικές πράξεις και παράγουν αντίστοιχα αποτελέσματα στην έξοδο τους. Καθώς το όριο των εισόδων στον πίνακα αληθείας είναι N , οι μέγιστες τιμές εξόδου που μπορεί να υπολογίσει το LUT είναι 2^N , που αντιστοιχούν στις θέσεις μνήμης που μπορεί να προσπελάσει. Τυπική τιμή του N για τα FGPA της Xilinx είναι 6.

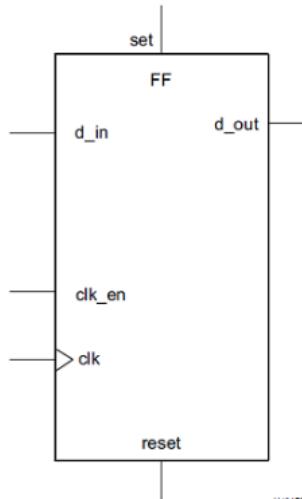


Σχήμα 2.3: Δομή LUT [3]

Η υλοποίηση στο υλικό ενός LUT μπορεί να θεωρηθεί σα μία συλλογή από κελιά μνήμης συνδεδεμένα σε ζευγάρια πολυπλεκτών. Οι είσοδοι των LUT δρουν σαν selector bits του πολυπλέκτη ο οποίος επιλέγει το αποτέλεσμα σε κάθε δεδομένη χρονική στιγμή. Είναι πολύ σημαντικό να έχουμε στο νου μας αυτή την αναπαράσταση καθώς ένα LUT μπορεί να χρησιμοποιηθεί τόσο για τον υπολογισμό κάποιας συνάρτηση όσο και ως στοιχείο μνήμης.

2.1.2 Flip Flops

Η βασική δομή ενός Flip-Flop περιλαμβάνει μία είσοδο δεδομένων, έναν παλμό ρολογιού, ένα σήμα reset και μία έξοδο δεδομένων. Κατά την κανονική λειτουργία του, οποιαδήποτε τιμή στην είσοδο του είναι μανδαλωμένη και μεταφέρεται στην έξοδο σε κάθε παλμό του ρολογιού. Ο σκοπός του clock enable pin είναι να επιτρέπει στο Flip-Flop να διατηρεί μια συγκεκριμένη τιμή για περισσότερους από έναν παλμούς ρολογιού. Νέα δεδομένα στην είσοδο απλώς μανδαλώνονται και περνούν στην έξοδο μόνο όταν το ρολόι αλλά και το clock enable είναι ίσα με 1.

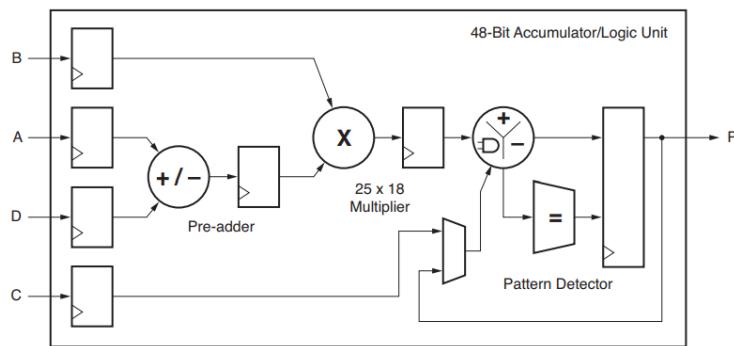


Σχήμα 2.4: Flip-Flop [4]

2.1.3 DSP48

Τα FPGA είναι αποδοτικά στην εκτέλεση εφαρμογών DSP καθώς μπορούν να υλοποιήσουν κατά-παρραγελία, πλήρως παραλληλοποιημένους αλγορίθμους. Οι εφαρμογές DSP χρησιμοποιούν πολλούς δυαδικούς πολλαπλασιαστές και συσσωρευτές οι

οποίοι είναι καλύτερα υλοποιήσιμοι στα DSP Slices. Όλα τα σύγχρονα FPGA της Xilinx διαθέτουν πολλά αποκλειστικά, πλήρως διαμορφώσιμα και χαμηλής κατανάλωσης DSP Slices τα οποία συνδυάζουν υψηλή ταχύτητα με μικρό μέγεθος ενώ διατηρούν την ευελιξία του σχεδιασμού του συστήματος σε υψηλό επίπεδο. Τα DSP Slices ενισχύουν την ταχύτητα και την απόδοση πολλών εφαρμογών πέρα από την σκοπιά της Ψηφιακής Επεξεργασίας Σημάτων, όπως για παράδειγμα σε wide dynamic bus shifters, memory address generators, wide bus multiplexers και memory-mapped I/O registers. Η βασική λειτουργικότητα των DSP48 slices φαίνεται στο ακόλουθο σχήμα.



Σχήμα 2.5: DSP48 Slice [5]

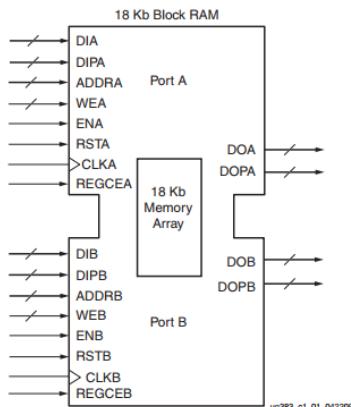
Μερικά χαρακτηριστικά της λειτουργικότητας των DSP Slices είναι:

- 25x18 πολλαπλασιαστές συμπληρώματος ως προς 2:
Dynamic bypass
- Συσσωρευτής 48bit:
Μπορεί να χρησιμοποιηθεί σαν ασύγχρονος απαριθμητής
- Pre-adder χαμηλής κατανάλωσης:
Βελτιστοποιεί εφαρμογές συμμετρικών φίλτρων και μειώνει την κατανάλωση των DSP Slices
- Single-instruction-multiple-data (SIMD) αριθμητική μονάδα:
Dual 24-bit ή quad 12-bit add/subtract/accumulate
- Προαιρετική λογική μονάδα:
Μπορεί να παράξει οποιαδήποτε από τις δέλτα λογικές πράξεις δύο τελεστών
- Ανιχνευτής προτύπων
- Προχωρημένα χαρακτηριστικά:
Προαιρετικό pipelining

2.1.4 BRAM & Άλλες μνήμες

Το FPGA Fabric περιλαμβάνει ενσωματωμένα στοιχεία μνήμης τα οποία μπορούν να χρησιμοποιηθούν ως μνήμες RAM, ROM ή καταχωρητές ολίσθησης. Τα στοιχεία αυτά είναι block RAMs (BRAMs), LUT και καταχωρητές ολίσθησης αντίστοιχα.

Το BRAM είναι ένα στοιχείο RAM δύο θυρών ενσωματωμένο στο FPGA fabric για να παρέχει αποθηκευτικό χώρο για μεγάλο αριθμό δεδομένων. Οι δύο τύποι BRAM που είναι διαθέσιμοι σε μια συσκευή μπορούν να αποθηκεύσουν είτε 18k είτε 36k bits και το σύνολο του αποθηκευτικού χώρου διαφέρει με την συσκευή. Λόγω των δύο θυρών που διαθέτουν, επιτρέπεται η παράλληλη και στον ίδιο παλμό ρολογιού πρόσβαση δεδομένων σε διαφορετικές θέσεις.

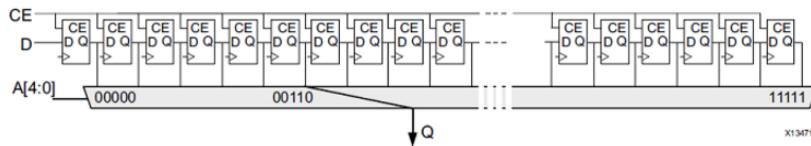


Σχήμα 2.6: Dual Port BRAM [6]

Όπως αναφέρθηκε και προηγουμένως, το LUT είναι μια μικρή μνήμη στην οποία εγγράφονται τα περιεχόμενα ενός πίνακα αληθείας. Λόγω της ευελιξίας των LUT στα FPGA της Xilinx, τα blocks αυτά μπορούν να χρησιμοποιηθούν και ως 64bit μνήμες και αναφέρονται συνήθως σαν κατανεμημένη μνήμη. Είναι ο πιο γρήγορος διαθέσιμος τύπος μνήμης στα FPGA, καθώς μπορεί να ενσωματωθεί σε οποιοδήποτε σημείο του fabric που βελτιώνει την απόδοση του κυκλώματος προς υλοποίηση.

Ο shift register (καταχωρητής ολίσθησης) μπορεί να θεωρηθεί ως μία αλυσίδα καταχωρητών συνδεδεμένων ο ένας με τον άλλον. Ο σκοπός της δομής αυτής είναι να παρέχει τη δυνατότητα της επαναχρησιμοποίησης των δεδομένων κατά μήκος ενός υπολογιστικού μονοπατιού, π.χ. κατά την κατασκευή ενός φίλτρου. Για παράδειγμα, ένα βασικό φίλτρο αποτελείται από μια σειρά πολλαπλασιαστών που πολλαπλασιά-

ζουν ένα δείγμα δεδομένων με ένα σετ συντελεστών. Χρησιμοποιώντας έναν shift register για την αποθήκευση των δεδομένων εισόδου, μία ενσωματωμένη δομή μεταφοράς μεταφέρει τα δείγματα στον επόμενο πολλαπλασιαστή στην αλυσίδα σε κάθε παλμό του ρολογιού.



Σχήμα 2.7: Addressable Shift Register [6]

2.2 Zynq-7000 ApSoc

Η οικογένεια Zynq®-7000 της Xilinx είναι βασισμένη στην αρχιτεκτονική ‘Xilinx All Programmable SoC’. Τα προϊόντα αυτά περιέχουν έναν διπύρηνο ή μονοπύρηνο ARM® Cortex™-A9 που αποτελεί το Processing System (PS) και την Programmable Logic (PL) κατασκευασμένη στα 28nm. Η ARM Cortex-A9 CPU είναι το κέντρο του PS και περιλαμβάνει επίσης on-chip μνήμη, εξωτερικές διασυνδέσεις μνήμης και ένα πλούσιο σετ από διεπαφές περιφερειακών. Όλα αυτά καθιστούν την οικογένεια Zynq®-7000 να διαθέτει πολύ γρήγορη αλληλεπίδραση κατά τη διάρκεια επιτάχυνσης εφαρμογών. [7]

Η οικογένεια Zynq-7000 προσφέρει την ευελιξία και την επεκτασιμότητα ενός FPGA, παρέχοντας παράλληλα απόδοση, ισχύ και ευκολία χρήσης που συσχετίζεται συνήθως με ASIC και application specific standard products (ASSPs). Το φάσμα των συσκευών της οικογένειας Zynq-7000 επιτρέπει στους σχεδιαστές να στοχεύσουν ευαίσθητες ως προς το κόστος καθώς και εφαρμογές υψηλής απόδοσης από μια ενιαία πλατφόρμα χρησιμοποιώντας εργαλεία που βασίζονται σε βιομηχανικά πρότυπα. Ενώ κάθε συσκευή της οικογένειας Zynq-7000 περιέχει το ίδιο PS, οι πόροι των PL και I/O διαφέρουν μεταξύ των συσκευών. Ως αποτέλεσμα, τα Zynq-7000S SoCs είναι σε θέση να εξυπηρετούν ένα ευρύ φάσμα εφαρμογών, όπως:

- Βοήθεια οδηγού, πληροφορίες και ψυχαγωγία
- Κάμερα εκπομπής

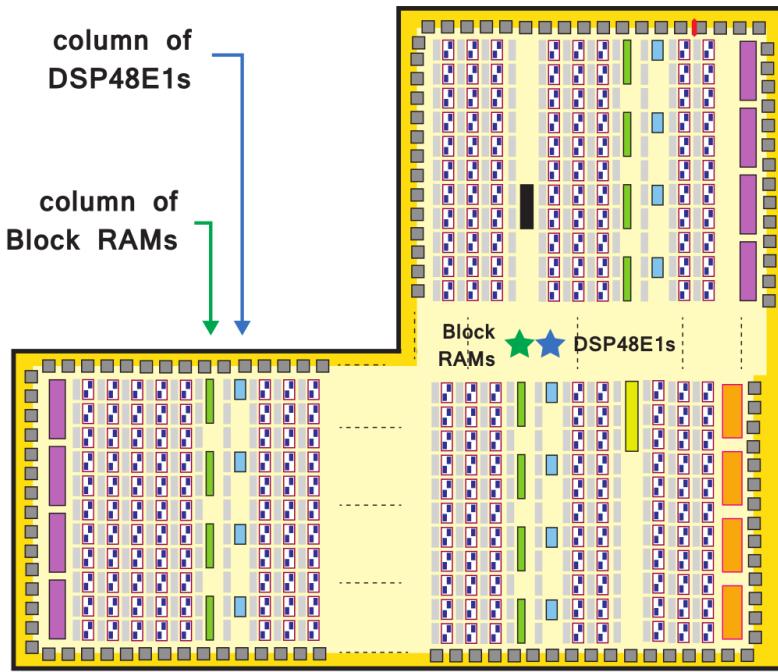
- Έλεγχος βιομηχανικού κινητήρα, βιομηχανική δικτύωση και μηχανική όραση
- IP και έξυπνη κάμερα
- LTE radio και baseband
- Ιατρική διάγνωση και απεικόνιση
- Εξοπλισμός βίντεο και νυχτερινή όραση

Η αρχιτεκτονική Zynq-7000 επιτρέπει την υλοποίηση προσαρμοσμένης λογικής στο PL και προσαρμοσμένο λογισμικό στο PS για την πραγματοποίηση μοναδικών και διαφοροποιημένων λειτουργιών του συστήματος. Η ενσωμάτωση του PS με το PL επιτρέπει επίπεδα απόδοσης που δε μπορούν να επιτευχθούν από ένα σύστημα 2 chips (πχ ένα ASSP με ένα FPGA) λόγω της περιορισμένης εύρους ζώνης I/O, της καθυστέρησης παραγωγής δεδομένων και του προϋπολογισμού ισχύος.

Η Xilinx προσφέρει ένα μεγάλο αριθμό IPs για την οικογένεια Zynq-7000. Προγράμματα οδήγησης συσκευών τόσο αυτόνομα όσο και για Linux είναι διαθέσιμα για τα περιφερειακά του PS και του PL. Το περιβάλλον ανάπτυξης Vivado® Design Suite επιτρέπει γρήγορη ανάπτυξη προϊόντων. Η υιοθέτηση του βασισμένου σε ARM υπολογιστικού συστήματος συνδυάζει ένα ευρύ φάσμα εργαλείων τρίτων με το υπάρχον οικοσύστημα PL της Xilinx. Η συμπερίληψη ενός επεξεργαστή εφαρμογών επιτρέπει την υποστήριξη υψηλού επιπέδου λειτουργικών συστημάτων, όπως το Linux. Άλλα τυπικά λειτουργικά συστήματα που χρησιμοποιούνται με τον επεξεργαστή Cortex-A9 είναι επίσης διαθέσιμα για την οικογένεια Zynq-7000.

Το PS και το PL βρίσκονται σε χωριστούς τομείς ισχύος, επιτρέποντας στον χρήστη αυτών των συσκευών να απενεργοποιήσουν την PL εάν απαιτείται για να επιτύχουν καλύτερη διαχείριση ισχύος. Οι επεξεργαστές στο PS πάντα εκκινούν πρώτα, επιτρέποντας μια εστιασμένη στο λογισμικό προσέγγιση για τη διαμόρφωση του PL. Τη διαμόρφωση PL διαχειρίζεται λογισμικό που εκτελείται στη CPU.

Η οικογένεια Zynq-7000 της Xilinx περιλαμβάνει ένα μεγάλο αριθμό συσκευών με διαφορετικό αριθμό πόρων. Μπορούμε να τις διαχωρίσουμε σε δύο υποκατηγορίες: Cost-optimized και- Mid-Range. Στο ακόλουθο σχήμα φαίνονται οι συσκευές και τα χαρακτηριστικά τους. [8]



Σχήμα 2.8: Programming Logic [8]

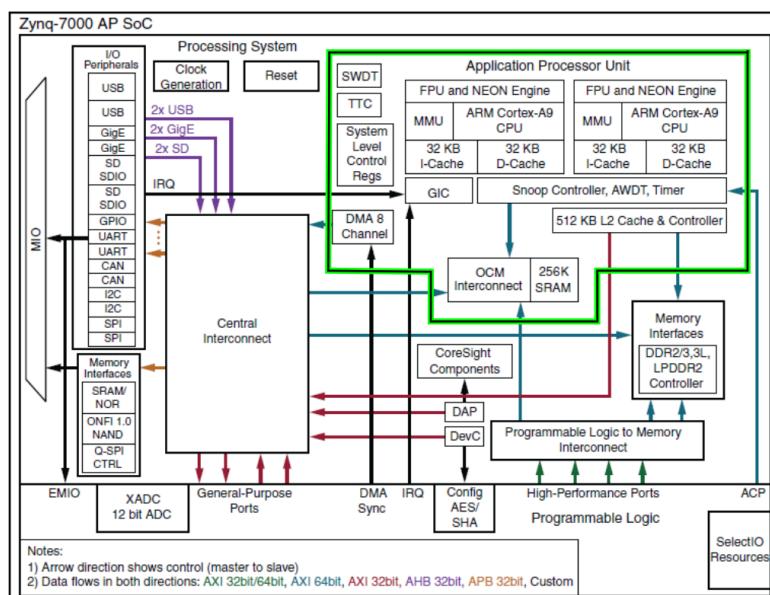
	Cost-Optimized Devices					Mid-Range Devices				
	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz					Dual-Core ARM Cortex-A9 MPCore Up to 866MHz				Dual-Core ARM Cortex-A9 MPCore Up to 1GHz ^[1]
Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor					32KB Instruction, 32KB Data per processor				
L1 Cache	512KB					256KB				
L2 Cache	256KB					512KB				
On-Chip Memory	DDR3, DDR3L, DDR2, LPDDR2					2x Quad-SPI, NAND, NOR				
External Memory Support ^[2]	2x Quad-SPI, NAND, NOR					8 (4 dedicated to PL)				
External Static Memory Support ^[2]	2x Quad-SPI, NAND, NOR					2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO				
DMA Channels	8 (4 dedicated to PL)					2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO				
Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO					RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot				
Peripherals w/ built-in DMA ^[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO					2x AXI 32b Master, 2x AXI 32b Slave				
Security ^[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot					4x AXI 64b/32b Memory				
Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	4x AXI 64b ACP					AXI 64b ACP				16 Interrupts
Programmable Logic (PL)										
7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
DSP Slices	66	120	170	80	160	220	400	900	900	2,020
PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
Analog Mixed Signal (AMS) / XADC ^[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
Security ^[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
Speed Grades	Commercial	-1		-1		-1		-1		-1
	Extended	-2		-2,-3		-2,-3		-2,-3		-2
	Industrial	-1, -2		-1, -2, -1L		-1, -2, -2L		-1, -2, -2L		-1, -2, -2L

Σχήμα 2.9: Zynq®-7000 All Programmable SoC Family [9]

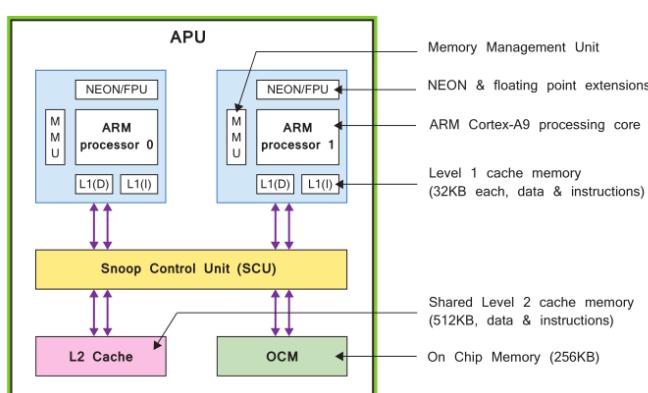
2.2.1 Application Processing Unit (APU)

Το PS του Zynq εκτός από τους πυρήνες ARM περιλαμβάνει και ένα set υπολογιστικών πόρων, σχηματίζοντας έτσι ένα Application processing unit (APU), το οποίο φαίνεται στο ακόλουθο σχήμα.

Το APU αποτελείται κυρίως από τους δύο ARM πυρήνες, καθένας από τους οποίους περιλαμβάνει: έναν NEON Media Processing Engine και Floating Point Unit, μία Memory Management Unit καθώς και μνήμες cache. Τέλος, η μονάδα Snoop Control δημιουργεί μία διασύνδεση μεταξύ των ARM, των μνημών και του PL. Η SCU διαχειρίζεται όλες τις συναλλαγές μεταξύ των PS και PL μέσω της Accelerator Coherency Port (ACP).



Σχήμα 2.10: Zynq APU [8]

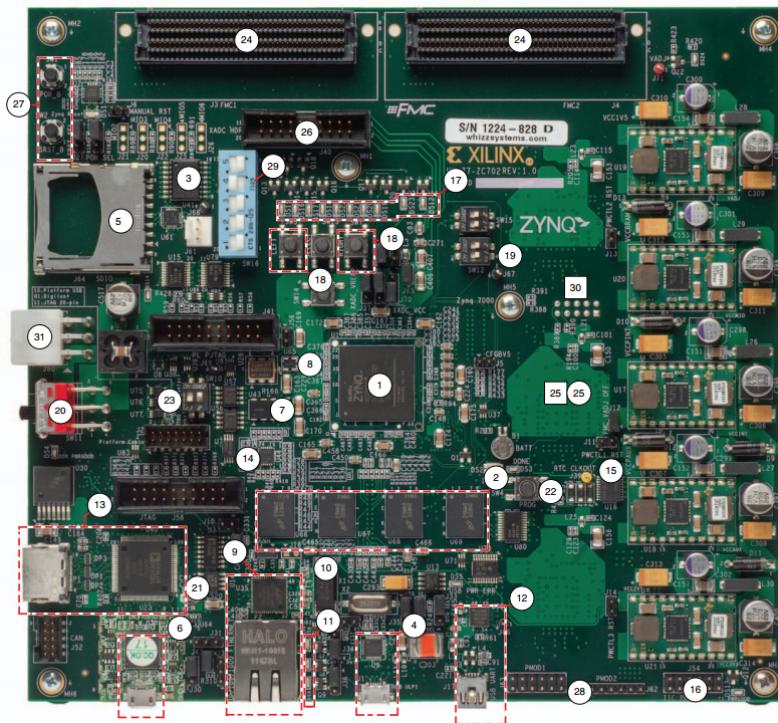


Σχήμα 2.11: APU Block diagram [8]

2.3 ZC702 Evaluation Kit

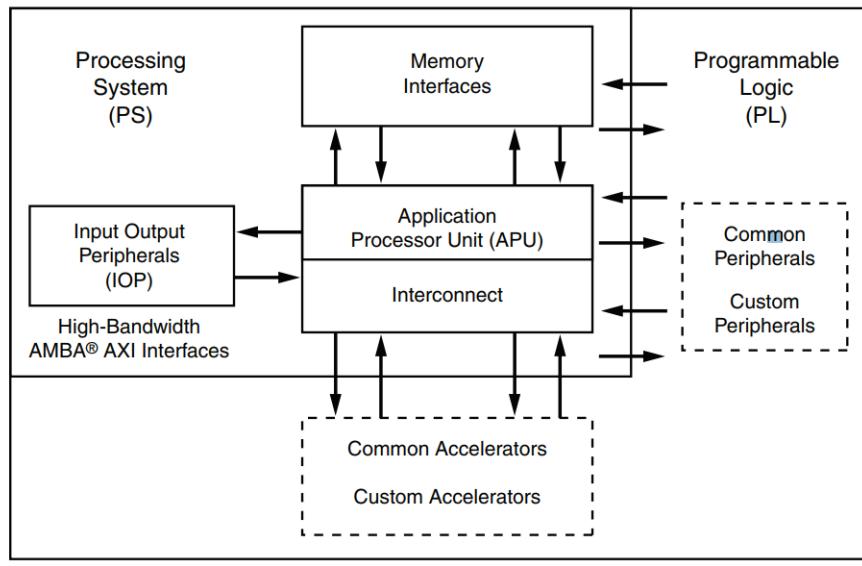
Η συσκευή που χρησιμοποιήθηκε στην παρούσα διπλωματική είναι το ZC702 Evaluation Kit της Xilinx που περιλαμβάνει το Zynq-7000 XC7Z020-1CLG484C AP SoC και είναι αντίστοιχο με ένα Artix-7 FPGA. Ανήκει στο low-end κομμάτι των προϊόντων της αρχιτεκτονικής Zynq-7000 με 85K9 Logic Cells, 53.200 LUT, 106.400 Flip-Flops και 220 DSP slices με 560KB BRAM. Το ZC702 evaluation board παρέχει ένα περιβάλλον υλικού για την ανάπτυξη και την αξιολόγηση σχεδίων. Το PS περιλαμβάνει δύο πυρήνες ARM® Cortex™-A9 MPCore™, διασύνδεση AMBA®, εσωτερική μνήμη, εξωτερικές διεπαφές μνήμης καθώς και περιφερειακά όπως: USB, Ethernet, SPI, SD/SDIO, I2C, CAN, Universal Asynchronous Receiver-Transmitter (UART) και GPIO. Στο 2.12 φαίνεται το board ανάπτυξης ZC702. Τα κυριότερα χαρακτηριστικά του είναι:

- Zynq XC7Z020-1CLG484C
- 1 GB DDR3
- 128 Mb Quad SPI flash memory
- USB 2.0 ULPI
- Secure Digital (SD) connector
- USB JTAG
- USB-to-UART
- HDMI codec
- I²C
- Status LEDs
- User I/O
- VITA 57.1 FMC LPC connectors
- Dual 12-bit 1 MSPS XADC analog-to-digital front end



Σχήμα 2.12: ZC702 Evaluation kit [10]

Στο σχήμα 2.13 φαίνεται το λογικό διάγραμμα υψηλού επιπέδου με τις διασυνδέσεις του υπολογιστικού συστήματος με την PL.



Σχήμα 2.13: Λογικό διάγραμμα υψηλού επιπέδου [11]

Στη συνέχεια θα γίνει αναφορά στις διάφορες συσκευές και περιφερειακά που χρησιμοποιήθηκαν στην παρούσα εργασία.

2.3.1 Διαμόρφωση συσκευής

Η συσκευή χρησιμοποιεί μια διαδικασία εκκίνησης πολλαπλών σταδίων που υποστηρίζει τόσο ασφαλή όσο και μη ασφαλή εκκίνηση. Το υπολογιστικό σύστημα είναι ο κύριος της διαδικασίας εκκίνησης και διαμόρφωσης. Για ασφαλή εκκίνηση, το PL πρέπει να είναι ενεργοποιημένο για να επιτρέψει τη χρήση του του μπλοκ ασφαλείας το οποίο παρέχει αποχρυπτογράφηση/πιστοποίηση ταυτότητας 256 bit AES & SHA. Οι επιλογές διαμόρφωσης είναι οι ακόλουθες:

- Ρύθμιση PS: Μνήμη flash Quad SPI
- Διαμόρφωση PS: Εκκίνηση συστήματος επεξεργαστή από κάρτα SD ή μέσω SDK
- Διαμόρφωση PL: Θύρα διαμόρφωσης USB JTAG ή Platform cable JTAG

2.3.2 USB-to-UART Bridge

Η πλακέτα ZC702 περιέχει μια USB-to-UART συσκευή της Silicon Labs (CP2103GM) που επιτρέπει τη σύνδεση με έναν κεντρικό υπολογιστή μέσω θύρας USB. Οι ακροδέκτες του - TX & RX - είναι συνδεδεμένοι στο μπλοκ IP του UART_1 που βρίσκεται εντός των περιφερειακών του XC7Z020 AP SoC.

2.3.3 AXI4

Το AXI είναι κομμάτι του ARM AMBA, μίας οικογένεις μικροελεγκτών και διασυνδέσεων που έκανε την εμφάνισή της το 1996. Η τελευταία έκδοση κυκλοφόρησε το 2010 και περιλαμβάνει τη δεύτερη έκδοση του AXI, AXI4. Υπάρχουν 3 τύποι διεπαφών AXI4:

- AXI4 - high-performance memory-mapped requirements
- AXI4-Lite - simple, low-throughput memory-mapped communication
- AXI4-Stream - high-speed streaming data

Στην εργασία για τη μεταφορά δεδομένων, θα χρησιμοποιηθεί κυρίως το AXI4-Stream με το S_AXI_HP το οποίο είναι μία διεπαφή υψηλής απόδοσης ανάμεσα στο PL και το PS. Επιτρέπει ένα κανάλι μεγάλου throughput και αποτελεί έναν από του βέλτιστους τρόπους μεταφοράς μεγάλων όγκων δεδομένων σε εφαρμογές video, υλοποίησης φίλτρων κάτια.

2.4 Software

2.4.1 Εισαγωγή

Ανάλογα με την απόδοση και την ευκολία προγραμματισμού του συστήματος, ο μηχανικός είναι επιφορτισμένος με τον καθορισμό της καλύτερης πλατφόρμας υλοποίησης ώστε να βγάλει το προϊόν στην αγορά. Για να φέρει το έργο του εις πέρας, έχει τη βοήθεια των διάφορων τεχνικών προγραμματισμού και μιας ποικιλίας πλατφορμών επεξεργασίας υλικού.

Στον τομέα του προγραμματισμού, τις τελευταίες δεκαετίες έχουν γίνει σημαντικές βελτιώσεις στον αντικειμενοστραφή προγραμματισμό και στην παράλληλη επεξεργασία ώστε να επιτευχθεί αύξηση της απόδοσης. Οι πρόοδοι στις γλώσσες προγραμματισμού επέτρεψαν στο μηχανικό να δημιουργεί και να δοκιμάζει προσεγγίσεις με πολύ γρήγορο ρυθμό. Η ανάγκη όμως αυτή, για γρήγορη ανάπτυξη συστημάτων οδήγησε στο ερώτημα του που θα εκτελείται ο αλγόριθμος.

Σχετικά με την πλατφόρμα στην οποία θα εκτελείται ο αλγόριθμος, υπάρχουν αρκετές επιλογές για έναν μηχανικό, όπως για παράδειγμα ASIC, FPGA, multiprocessor system-on-chip (MPSoC). Τονίζουμε ότι υπάρχει μια συνεχώς αυξανόμενη εστίαση στην παραλληλοποίηση και ταυτόχρονη εκτέλεση εντολών.

Έχουν γίνει σημαντικές πρόοδοι στην κατασκευή των ASIC βελτιώνοντας κατά πολύ την απόδοση, την κατανάλωση και την υπολογιστική ρυθμοαπόδοση τους. Το κόστος για την ανάπτυξη του κυκλώματος όμως παραμένει ακριβό και η χρονοβόρα διαδικασία μετάφρασης του αλγορίθμου σε υλικό τα καθιστά μία οικονομικά βιώσιμη λύση μόνο όταν αναφερόμαστε σε εφαρμογές που θα παραχθούν σε πολύ μεγάλα νούμερα (της τάξης των εκατομμυρίων).

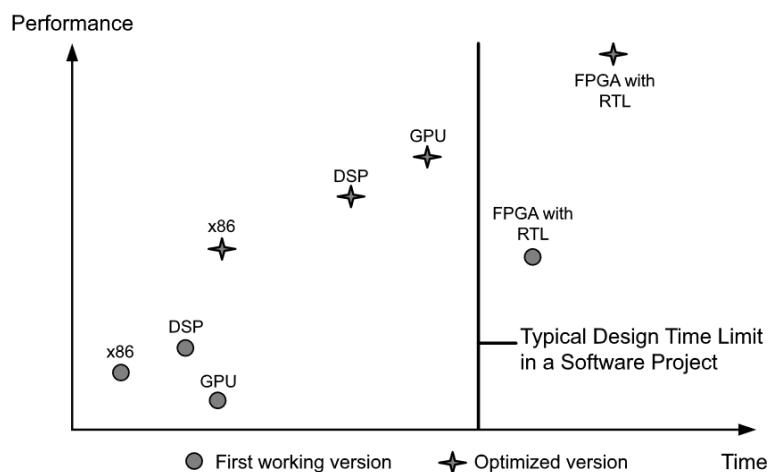
Τα FPGA επιτρέπουν στον σχεδιαστή να υλοποιεί ειδικές υλοποιήσεις του αλ-

γορίθμου χρησιμοποιώντας ‘off-the-shelf’ εξαρτήματα βασικής προγραμματιζόμενης λογικής. Η πλατφόρμα αυτή προσφέρει χαμηλή κατανάλωση και μεγάλη απόδοση χωρίς να έχει το αυξημένο κόστος των ASICs.

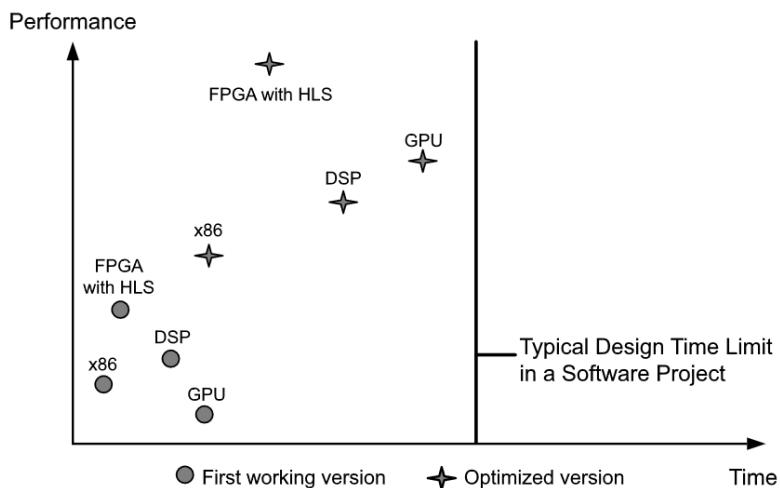
Για τον προγραμματισμό των FPGA παραδοσιακά χρησιμοποιούνται γλώσσες περιγραφής υλικού (HDL - Hardware Description Languages). Οι γλώσσες αυτές παρέχουν ειδικά χαρακτηριστικά που διευκολύνουν το σχεδιασμό ακολουθιακής ή συνδυαστικής λογικής. Ενώ χρησιμοποιούνται κατά κόρον τις τελευταίες δεκαετίες, παρουσιάζουν αρκετά μειονεκτήματα σε σχέση με γλώσσες προγραμματισμού υψηλού επιπέδου, όπως για παράδειγμα μεγάλος χρόνος ανάπτυξης και προσομοίωσης.

Μια νέα τεχνική έχει κάνει την εμφάνιση της τις δύο τελευταίες δεκαετίες για να αντικαταστήσει τον προγραμματισμό σε VHDL/Verilog, η οποία καλείται High Level Synthesis HLS. Η σύνθεση υψηλού επιπέδου, μερικές φορές αναφέρεται και ως σύνθεση C, είναι μια αυτοματοποιημένη διαδικασία σχεδιασμού που ερμηνεύει μία αλγορίθμική περιγραφή μίας επιθυμητής συμπεριφοράς και δημιουργεί υλικό που την υλοποιεί. Η σύνθεση αρχίζει με μια προδιαγραφή υψηλού επιπέδου για το πρόβλημα, όπου η συμπεριφορά αποσυνδέεται γενικά από π.χ. χρονοδιάγραμμα σε επίπεδο ρολογιού.

Στα ακόλουθα σχήματα, φαίνεται ότι η μέθοδος σχεδιασμού σε επίπεδο καταχωρητών μπορεί να οδηγήσει σε περιορισμούς σχετικά με το χρόνο υλοποίησης και την επιτευχείσα απόδοση, σε αντίθεση με την προσέγγιση μέσω HLS η οποία μπορεί να οδηγήσει σε ικανοποιητικά αποτελέσματα σε σημαντικά μικρότερο χρονικό διάστημα.



Σχήμα 2.14: Xilinx - Χρόνος σχεδιασμού vs. Απόδοση εφαρμογής με RTL [12]



Σχήμα 2.15: Xilinx - Χρόνος σχεδιασμού vs. Απόδοση εφαρμογής με HLS [12]

2.4.2 Γλώσσα προγραμματισμού VHDL

Στον τρόπο προγραμματισμού που έχει επικρατήσει τις τελευταίες δύο δεκαετίες, ο σχεδιαστής υλικού παρέχει χειροκίνητα μία περιγραφή σε επίπεδο καταχωρητή, η οποία εφαρμόζεται στο υλικό μέσω HDL. Στη συνέχεια η περιγραφή RTL προσομοιώνεται με τη βοήθεια ενός προγράμματος που καλείται testbench. Εαν δεν υπάρχουν σφάλματα στη σχεδίαση δημιουργείται το netlist. Στις περισσότερες περιπτώσεις, τα FPGA συνοδεύονται από μια πλατφόρμα λογισμικού του κατασκευαστή που χρησιμοποιείται για την παραγωγή του netlist.

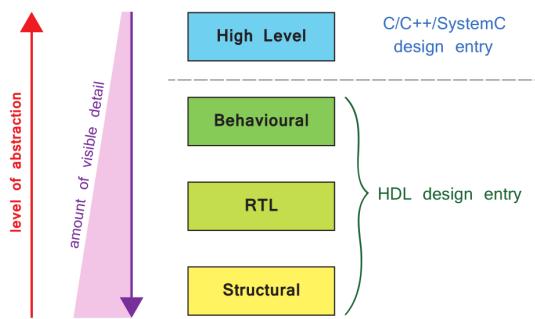
Η περιγραφή αυτή στη συνέχεια υλοποιείται στο FPGA μέσω μιας ειδικής διαδικασίας που καλείται place-and-route και τελικά παράγεται το δυαδικό αρχείο (bitstream) που απαιτείται για τον προγραμματισμό του. Το bitstream φορτώνεται στο FPGA μέσω διεπαφής JTAG ή εξωτερικής μνήμης.

2.4.3 HLS

2.4.3.1 Επισκόπηση

Ο κώδικας αναλύεται, περιορίζεται σε επίπεδο αρχιτεκτονικής και προγραμματίζεται ώστε να παραχθεί μια γλώσσα περιγραφής υλικού σε επίπεδο καταχωρητών, η οποία με τη σειρά της συνήθως συντίθεται σε επίπεδο πύλης με τη χρήση ενός λογικού εργαλείου σύνθεσης. Ο στόχος του HLS είναι να επιτρέψει στους σχεδιαστές υλικού

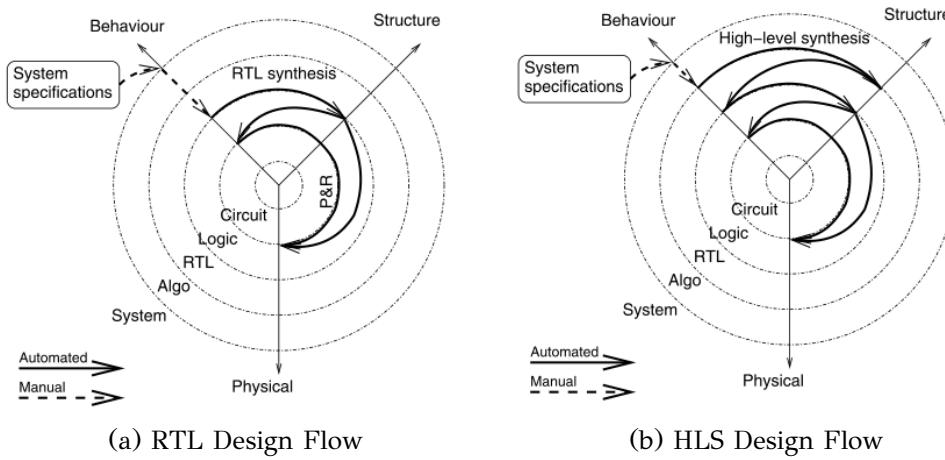
να δημιουργήσουν και να ελέγξουν αποτελεσματικά το υλικό, παρέχοντάς του τον καλύτερο έλεγχο στη βελτιστοποίηση της αρχιτεκτονικής επιτρέποντας του παράλληλα να περιγράφει το σχέδιο σε υψηλότερο επίπεδο αφαίρεσης. Η επαλήθευση του RTL αποτελεί σημαντικό μέρος της διαδικασίας.



Σχήμα 2.16: Επίπεδα αφαιρετικότητας σε σχεδιασμούς FPGA [8]

Ενώ η λογική σύνθεση χρησιμοποιεί μια RTL περιγραφή του σχεδιασμού, η σύνθεση υψηλού επιπέδου λειτουργεί σε υψηλότερο επίπεδο αφαίρεσης, ξεκινώντας με μια αλγορίθμική περιγραφή σε μια γλώσσα υψηλού επιπέδου όπως η SystemC και η ANSI C/C++. Ο σχεδιαστής αναπτύσσει τη λειτουργικότητα της μονάδας και το πρωτόκολλο διασύνδεσης και στη συνέχεια τα εργαλεία σύνθεσης υψηλού επιπέδου χειρίζονται την μικρο-αρχιτεκτονική και μετασχηματίζουν τον λειτουργικό κώδικα σε πλήρως χρονομετρημένες υλοποιήσεις RTL, δημιουργώντας αυτόματα λεπτομέρειες για τον κύκλο εκτέλεσης του υλικού.

Το ακόλουθο σχήμα (2.17) διαθέτει τρεις άξονες που αναπαριστούν τις διαφορετικές όψεις του σχεδιασμού: συμπεριφορά (τί κάνει το υποκύκλωμα), δομή (πώς είναι δομημένο το κύκλωμα - netlist) και γεωμετρία (πώς είναι δομημένο σε φυσικό επίπεδο το κύκλωμα - layout). Υπάρχουν επίσης πέντε ομόκεντροι κύκλοι που αντιπροσωπεύουν τα επίπεδα της αφαιρετικότητας: κυκλώματος, λογικής, RTL, αλγορίθμου και συστήματος.



Σχήμα 2.17: High-level synthesis vs RTL στο Gasjki-Kuhn Y-chart [13]

2.4.3.2 Scheduling & Binding

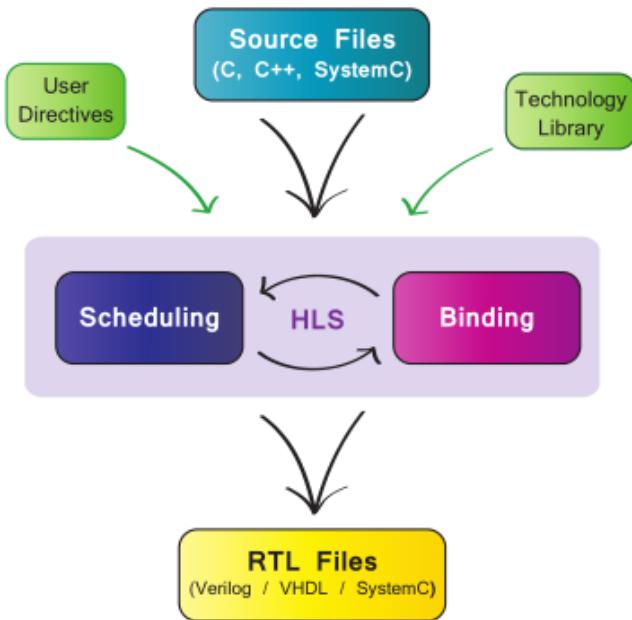
Το HLS αποτελείται από δύο κύριες διεργασίες: scheduling και binding. Αυτές πραγματοποιούνται σε επαναληπτική βάση όπως φαίνεται στο επόμενο σχήμα καθώς η μία επηρεάζει την άλλη. Οι λειτουργίες των διεργασιών αυτών αναλύονται στη συνέχεια:

Scheduling

Είναι η διαδικασία της μετάφρασης των RTL δηλώσεων που μεταφράστηκαν από τον κώδικα C σε ένα σύνολο εντολών, καθένα με την αντίστοιχη διάρκεια εκφρασμένη σε κύκλους ρολογιού. Οι αποφάσεις που πραγματοποιούνται σε αυτό το στάδιο επηρεάζονται από τη συχνότητα του ρολογιού και την αβεβαιότητα του, την τεχνολογία της συσκευής που θα χρησιμοποιηθεί και τις ντιρεκτίβες που εφαρμόζονται από το χρήστη.

Binding

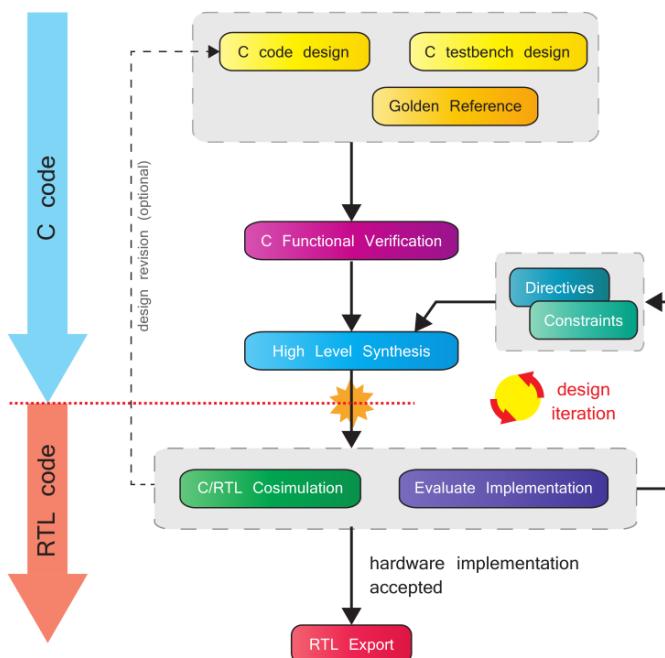
Binding είναι η διαδικασία του συσχετισμού των προγραμματισμένων λειτουργιών με τους φυσικούς πόρους της συσκευής. Τα χαρακτηριστικά της λειτουργικότητας και του χρονισμού αυτών των πόρων μπορούν να επηρεάσουν τον προγραμματισμό των εργασιών, επομένως πληροφορίες σχετικές με το binding ανατροφοδοτούνται στη διαδικασία του scheduling.



Σχήμα 2.18: Ροή scheduling & binding [8]

2.4.3.3 Design Flow

Αναλυτικότερα, η πλήρης ροή σχεδιασμού με το HLS φαίνεται στο ακόλουθο σχήμα. Τα στάδια αυτά θα περιγραφούν στη συνέχεια.



Σχήμα 2.19: Ροή σχεδιασμού με HLS [8]

Δεδομένα εισόδου στο HLS

Τα πρωταρχικά δεδομένα εισόδου στο HLS είναι μία C/C++/systemC συνάρτηση καθώς και ένα αρχείο testbench το οποίο έχει αναπτυχθεί για να γίνει επαλήθευση της λειτουργίας της συνάρτησης. Το αρχείο αυτό περιέχει ‘golden data’ τα οποία θα συγκριθούν με τα δεδομένα εξόδου της συνάρτησης.

Λειτουργική επαλήθευση

Αρχικά, είναι απαραίτητη η επαλήθευση της λειτουργική ακεραιότητας του κώδικα μας προτού παραχθεί ο κώδικας RTL με τη βοήθεια του testbench.

Σύνθεση

Επόμενο βήμα είναι η ίδια η σύνθεση υψηλού επιπέδου, η οποία περιλαμβάνει ανάλυση και επεξεργασία του κώδικα, των ντιρεκτίβων και περιορισμών ώστε να δημιουργηθεί μία περιγραφή RTL του υλικού.

Προσομοίωση & Αξιολόγηση

Μετά τη δημιουργία του μοντέλου RTL, δοκιμάζεται η σωστή λειτουργία του (σε επίπεδο RTL) με τη βοήθεια του πρωτότυπου testbench. Εκτός της προσομοίωση πρέπει να γίνει όμως και μία αξιολόγηση των αποτελεσμάτων συγκριτικά με τις προδιαγραφές που είχαν οριστεί αρχικά.

Εξαγωγή RTL

Τελευταίο βήμα είναι η εξαγωγή του υλικού για τη χρήση του σε μεγαλύτερα συστήματα. Η μέθοδος της εξαγωγής σε IP είναι από τις πλέον κατάλληλες μορφές ώστε να εισαχθεί το υλικό στον IP Integrator, XPS ή System Generator.

2.4.3.4 Βελτιστοποίηση

Μεγάλη βαρύτητα πρέπει να δοθεί στις μετρήσεις και εκτιμήσεις απόδοσης του υλικού που σχεδιάστηκε. Πιο συγκεκριμένα πρέπει να εστιάσουμε:

- Στους πόρους του FPGA που χρειάζονται για την υλοποίηση του σχεδίου μας και πως αυτοί συγκρίνονται με τους διαθέσιμους.
- Στη ρυθμοαπόδοση, δηλαδή σε τι ρυθμό μπορεί να επεξεργάζεται δεδομένα το

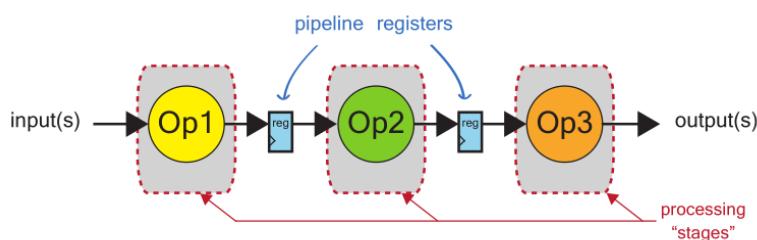
υλικό.

- Στη μέγιστη συχνότητα ρολογιού που μπορεί να λειτουργεί το σχέδιο.
- Στο latency, δηλαδή πόσοι κύκλοι απαιτούνται για να παραχθεί η έξοδος.
- Στην κατανάλωση ισχύος του συστήματος και
- στις απαιτήσεις των απαιτούμενων διεπαφών.

To Vivado HLS προσπαθεί από μόνο του να πετύχει το καλύτερο δυνατό σχεδιασμό RTL αν και τις περισσότερες φορές δίνει μεγαλύτερη έμφαση στην εξοικονόμηση πόρων θυσιάζοντας μέρος της απόδοσης του υλικού. Μας δίνεται όμως η δυνατότητα να πραγματοποιήσουμε βελτιστοποιήσεις κατά τη διάρκεια συγγραφής του κώδικα ώστε να επιτύχουμε την επιθυμητή απόδοση ή μία ισορροπία ανάμεσα στην απόδοση και την εξοικονόμηση πόρων. Στη συνέχεια αναλύονται οι βασικότερες μέθοδοι βελτιστοποίησης.

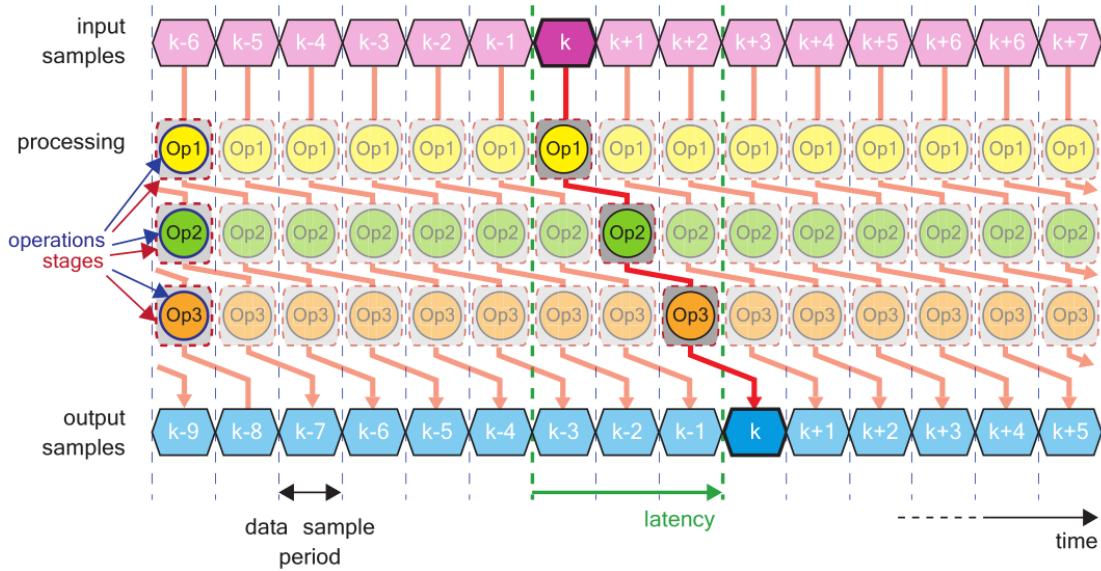
Pipelining

Το pipeline είναι μία από τις πιο συνηθισμένες τεχνικές βελτίωσης του χρόνου καθυστέρησης διεργασιών. Διαιρώντας τα στάδια της επεξεργασίας σε πολλά μικρότερα τα οποία μπορούν να επεξεργαστούν διαφορετικά δεδομένα εισόδου ταυτόχρονα, επιτυγχάνεται η παραλληλοποίηση των διεργασιών. Σε όρους υλικού, αυτό επιτυγχάνεται με την εισαγωγή καταχωρητών ανάμεσα στα νέα μικρότερα στάδια, επιτρέποντας έτσι τη διατήρηση στη μνήμη των δειγμάτων εισόδου.



Σχήμα 2.20: Διαμερισμός υπολογιστικών σταδίων σε μικρότερα μέσω pipelining [8]

Με την τεχνική αυτή, η συνολική ρυθμοαπόδοση αυξάνεται σημαντικά και αρκετά συχνά παρατηρείται αύξηση στη μέγιστη συχνότητα λειτουργίας του υλικού. Αυτό συμβαίνει διότι τα τρία στάδια εντολών είναι πλέον ελεύθερα να επεξεργάζονται ταυτόχρονα συνεχόμενα δείγματα εισόδου όπως φαίνεται στο ακόλουθο σχήμα.



Σχήμα 2.21: Latency & ρυθμοαπόδοση μετά το pipelining [8]

Πρέπει να δοθεί προσοχή σε ένα ακόμη μέγεθος, το Iteration Interval το οποίο εκφράζει τον αριθμό των κύκλων ανάμεσα στην είσοδο ενός νέου δείγματος εισόδου προς επεξεργασία. Χωρίς τη χρήση pipeline, το latency και το iteration interval μπορεί να ταυτίζοται καθώς το HLS βελτιστοποιεί το υλικό δίνοντας έμφαση στην εξοικονόμηση πόρων. Η στρατηγική χρήση pipeline μπορεί να τα μειώσει δραματικά. Όμως, η μείωση αυτή μπορεί να επιφέρει αυξημένη κατανάλωση πόρων επομένως χρειάζεται κάποιος συμβιβασμός.

Loop Unrolling

Οι βρόχοι χρησιμοποιούνται εκτεταμένα στον προγραμματισμό και αποτελούν μία φυσική μέθοδο για την αναπαράσταση διεργασιών που εκτελούνται επαναληπτικά. Με τη βοήθεια του Vivado HLS ο σχεδιαστής μπορεί να προτρέψει τους βρόχους να μετασχηματιστούν με διαφορετικούς τρόπους. Από προεπιλογή, οι βρόχοι είναι ‘rolled’, μοιράζονται δηλαδή ένα ελάχιστο σετ υλικού το οποίο αποτελεί το κύριο σώμα του, επηρεάζοντας αρνητικά το συνολικό latency.

Οι βρόχοι επομένως μπορούν να ξετυλιχθούν (unrolled) κατά κάποιο βαθμό Ν ώστε να δημιουργηθούν Ν επαναλήψεις του υλικού επιτρέποντας κατ’ αυτόν τον τρόπο την παράλληλη εκτέλεση των διεργασιών. Το μεγαλύτερο μειονέκτημα αυτής της μεθόδου είναι ότι μπορεί να οδηγήσει σε μεγάλα σχέδια υλικού. Και σε αυτή την περίπτωση πρέπει να γίνει κάποιος συμβιβασμός. Για παράδειγμα ο ακόλουθος

κώδικας,

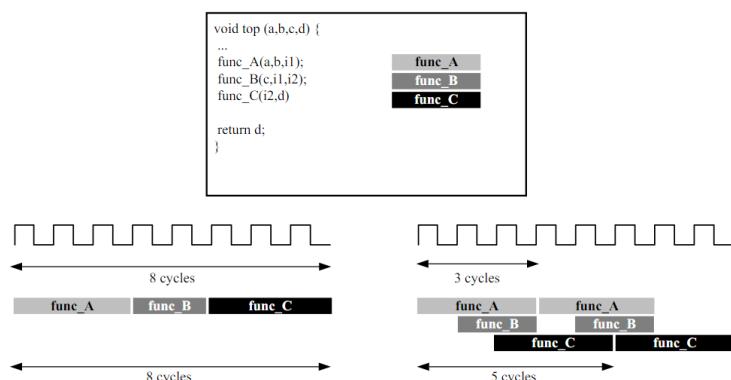
```
1 for(int i = 0; i < X; i++) {
2     #pragma HLS unroll factor=2
3     a[i] = b[i] + c[i];
4 }
```

μετασχηματίζεται στον:

```
1 for(int i = 0; i < X; i += 2) {
2     a[i] = b[i] + c[i];
3     if (i+1 >= X) break;
4     a[i+1] = b[i+1] + c[i+1];
5 }
```

Dataflow

Η βελτιστοποίηση dataflow είναι παρόμοια με το pipeline που αναλύθηκε προηγουμένως αλλά στην πραγματικότητα εφαρμόζεται σε υψηλότερο επίπεδο στη σχεδιαστική ιεραρχία. Επιτρέπει στις συναρτήσεις του προγράμματος να επικαλύπτονται κατά τη λειτουργία, αυξάνοντας το συγχρονισμό της RTL υλοποίησης και της ρυθμοαπόδοσης. Για παράδειγμα, συναρτήσεις που προσπελαύνουν πίνακες πρέπει να ολοκληρώσουν όλες τις εντολές ανάγνωσης και εγγραφής σε αυτούς προτού ολοκληρωθεί η εκτέλεσή τους, αποτρέποντας την εκκίνηση λειτουργίας της επόμενης συνάρτησης. Η dataflow βελτιστοποίηση επιτρέπει επομένως, σε μία συνάρτηση ή βρόχο να ξεκινήσει τη λειτουργία της πριν την ολοκλήρωση της προηγούμενης.



Σχήμα 2.22: dataflow βελτιστοποίηση [14]

Array Optimizations

Στο Vivado HLS, οι πίνακες αντιπροσωπεύουν συνήθως αποθηκευτικό χώρο και για αυτό συντίθενται σε μνήμες. Η μνήμη που ορίζεται κατά το HLS αντιστοιχίζεται σε φυσικούς πόρους του PL - BRAM/κατανεμημένη RAM - και είναι αναγκαίο να είναι γνωστό το μέγεθος της και πως αντιστοιχίζεται σε σχέση με τους διαθέσιμους πόρους του FPGA. Υπάρχει ένα σύνολο βελτιστοποιήσεων που μπορεί να εφαρμοστεί στους πίνακες κατά τη διάρκεια του σχεδιασμού.

- **Resource** Ο σχεδιαστής μπορεί να επιλέξει σε τι μνήμη θα αντιστοιχίσει τον πίνακα.
- **Array Map** Με αυτή τη βελτιστοποίηση πολλοί μικροί πίνακες μπορούν να συνδυαστούν σε έναν μεγαλύτερο μειώνοντας έτσι τον απαιτούμενο αριθμό πόρων μνήμης.
- **Array Partition** Μπορεί να θεωρηθεί ως η αντίθετη τεχνική του Array Mapping καθώς επιτρέπει στο σχεδιαστή να διαιρέσει έναν πίνακα σε μικρότερους υποπίνακες ώστε να αυξήσει το ρυθμό με τον οποίο πραγματοποιούνται μεταφορές δεδομένων από/προς τη μνήμη (πχ 2-port BRAM).
- **Array Reshape** Η ντιρεκτίβα αυτή επιτρέπει έναν πίνακα με μεγάλο αριθμό στοιχείων μικρού μήκους, να μετασχηματιστεί σε έναν πίνακα με λιγότερες αλλά μεγαλύτερες λέξεις μειώνοντας έτσι το συνολικό αριθμό προσπέλασης της μνήμης.
- **Stream** Με την εντολή Stream, ο πίνακας αντιστοιχίζεται σε δομές FIFO αντί RAM.

2.4.3.5 Περιορισμοί HLS

Η πλατφόρμα Vivado HLS ενώ υποστηρίζει ένα μεγάλο εύρος της γλώσσας προγραμματισμού C, ένα σύνολο constructs της δεν είναι συνθέσιμα ή μπορούν να οδηγήσουν σε σφάλματα κατά τη διάρκεια της σχεδιαστικής ροής [15]. Για να είναι ο κώδικας συνθέσιμος πρέπει:

- Να περιέχει το σύνολο της λειτουργικότητας της σχεδίασης
- Να μην βασίζεται σε κλήσεις του συστήματος για την πραγματοποίηση λειτουργιών

- Οι δομές μνήμης να είναι στατικές και όχι δυναμικές

Οι αλήσεις συστήματος δεν είναι συνθέσιμες καθώς είναι ενέργειες που σχετίζονται με την εκτέλεση εργασιών βασισμένων στο λειτουργικό σύστημα στο οποίο τρέχει ο κώδικας. Το Vivado HLS αγνοεί αυτό το είδος εντολών που δεν έχουν καμία επίδραση στην εκτέλεση του προγράμματος. Αν η χρήση τους όμως είναι αναγκαία για την πραγματοποίηση αποσφαλμάτωσης, τότε με τη βοήθεια του **macro __SYNTHESIS__** το Vivado HLS εξαίρει το συγκεκριμένο κομμάτι κώδικα. Για παράδειγμα:

```
1 #ifndef __SYNTHESIS__
2 ofstream myfile;
3 myfile.open ("example.txt");
4 myfile << "Hello World!\n";
5 myfile.close();
6 #endif
```

Οποιαδήποτε εντολή που διαχειρίζεται την κατανομή μνήμης, όπως για παράδειγμα οι **malloc()**, **alloc()** και **free()**, χρησιμοποιεί πόρους που υπάρχουν στη μνήμη του συστήματος που δεσμεύονται ή ελευθερώνονται κατά τη διάρκεια της εκτέλεσης. Τέτοιες εντολές πρέπει να αφαιρούνται από το σχεδισμό πριν τη σύνθεση και οι δυναμικές δομές να μετατραπούν σε στατικές με γνωστά όρια.

Κεφάλαιο 3

Υπολογιστική όραση

3.1 Ορισμός

Με τον όρο Computer Vision αναφερόμαστε στο διεπιστημονικό πεδίο που ασχολείται με τον τρόπο με τον οποίο μπορούν να κατασκευαστούν υπολογιστές για την επίτευξη υψηλής κατανόησης από ψηφιακές εικόνες και βίντεο. Από την άποψη της μηχανικής, είναι οι τρόποι για την επίτευξη της αυτοματοποίησης εργασιών που μπορούν να πραγματοποιηθούν από την ανθρώπινη όραση. [16]

Τα καθήκοντα της υπολογιστικής όρασης περιλαμβάνουν μεθόδους απόκτησης, επεξεργασίας, ανάλυσης και κατανόησης ψηφιακών εικόνων και εξαγωγής δεδομένων από τον πραγματικό κόσμο με σκοπό την παραγωγή αριθμητικών ή συμβολικών πληροφοριών. όπως για παράδειγμα στην μορφή λήψης αποφάσεων.

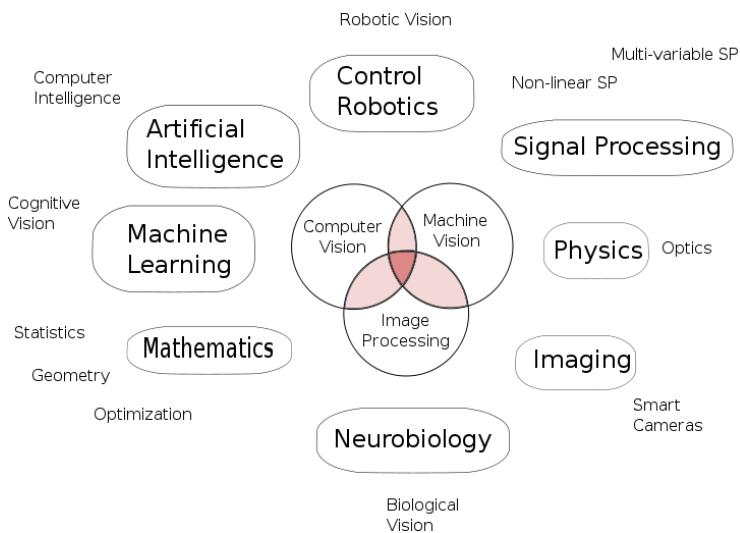
Η κατανόηση σε αυτό το πλαίσιο ταυτίζεται με τη μετατροπή των οπτικών εικόνων σε περιγραφές του κόσμου που μπορούν να αλληλεπιδρούν με άλλες διαδικασίες σκέψης και να διεγείρουν τις κατάλληλες ενέργειες.

Αυτή η κατανόηση των εικόνων μπορεί να θεωρηθεί ως η απευμπλοκή συμβολικών πληροφοριών από δεδομένα εικόνας χρησιμοποιώντας μοντέλα κατασκευασμένα με τη βοήθεια της γεωμετρίας, της φυσικής, των στατιστικών και της θεωρίας της μάθησης. Από τη σκοπιά της επιστήμης, η μηχανική όραση ασχολείται με τη θεωρία πίσω από τα τεχνητά συστήματα που εξάγουν πληροφορίες από εικόνες.

Τα δεδομένα αυτά μπορούν να λάβουν πολλαπλές μορφές όπως ακολουθίες βίντεο, προβολές από πολλές κάμερες ή πολυδιάστατα δεδομένα από ιατρικό σαρωτή. Από την σκοπιά της τεχνολογίας όμως, επιδιώκει να εφαρμόσει τις θεωρίες και τα μοντέλα για την κατασκευή συστημάτων ηλεκτρονικής όρασης.

Οι υποτομείς της μηχανικής όρασης περιλαμβάνουν την αναδημιουργία σκηνών,

την ανίχνευση συμβάντων, την παρακολούθηση βίντεο, την αναγνώριση αντικειμένων, την εκτίμηση θέσης 3D, τη μάθηση, την εκτίμηση κίνησης και την αποκατάσταση της εικόνας.



Σχήμα 3.1: Υπολογιστική όραση & σχετικοί τομείς [17]

3.2 Ιστορική αναδρομή

Στα τέλη της δεκαετίας του 1960, η μηχανική όραση ξεκίνησε σε πανεπιστήμια πρωτοπόρα στην τεχνητή νοημοσύνη. Σκοπός ήταν να μιμείται το ανθρώπινο οπτικό σύστημα, ως ένα ένα ενδιάμεσο βήμα για να προσδώσει έξυπνη συμπεριφορά σε ρομπότ. Αυτό που διαφοροποίησε τη υπολογιστική όραση από τον επιχρατέστερο τομέα της ψηφιακής επεξεργασίας εικόνων ήταν η επιθυμία να εξαχθεί τρισδιάστατη δομή από εικόνες με στόχο την επίτευξη πλήρους κατανόησης της σκηνής. Μελέτες στη δεκαετία του '70 αποτέλεσαν τα θεμέλια για πολλούς από τους αλγορίθμους υπολογιστικής όρασης που χρησιμοποιούνται ακόμη και τώρα, συμπεριλαμβανομένης της εξαγωγής ακμών από εικόνες, σήμανσης γραμμών, μη πολυεδρικής και πολυεδρικής μοντελοποίησης, αναπαράστασης αντικειμένων ως διασυνδέσεων μικρότερων δομών, οπτικής ροής και εκτίμηση κίνησης.

Την επόμενη δεκαετία πραγματοποιήθηκαν μελέτες βασισμένες σε πιο αυστηρές μαθηματικές αναλύσεις και ποσοτικές πτυχές της μηχανικής όρασης. Οι ερευνητές συνειδητοποίησαν επίσης ότι πολλές μαθηματικές έννοιες θα μπορούσαν να αντιμετωπιστούν εντός του ίδιου πλαισίου βελτιστοποίησης όπως για παράδειγμα η κανονι-

κοποίηση και τα τυχαία πεδία Markov. Μέχρι τη δεκαετία του 1990, μερικά από τα προηγούμενα ερευνητικά θέματα έγιναν πιο ενεργά από τα υπόλοιπα. Η έρευνα σε 3D-ανακατασκευές οδήγησε σε καλύτερη κατανόηση της βαθμονόμησης της κάμερας. Με την εμφάνιση των μεθόδων βελτιστοποίησης για τη βαθμονόμηση της κάμερας, διαπιστώθηκε ότι πολλές από τις ιδέες είχαν ήδη εξερευνηθεί στη θεωρία προσαρμογής δέσμης από το πεδίο της φωτογραφικής. Αυτό οδήγησε σε μεθόδους για αραιές 3-D ανακατασκευές σκηνών από πολλές εικόνες. Αυτή η δεκαετία σηματοδότησε επίσης την πρώτη φορά που οι τεχνικές στατιστικής μάθησης χρησιμοποιήθηκαν στην πράξη για την αναγνώριση προσώπων στις εικόνες.

Προς τα τέλη της δεκαετίας του 1990, σημαντική αλλαγή επήλθε με την αυξημένη αλληλεπίδραση μεταξύ των πεδίων των γραφικών και της μηχανικής όρασης. Αυτό περιελάμβανε απεικόνιση με βάση την εικόνα, μορφοποίηση εικόνας, παρεμβολή προβολής, ραφή πανοραμικών εικόνων και απόδοση φωτεινού πεδίου. Πρόσφατα, έχει παρατηρηθεί αναζωπύρωση σε μεθόδους βασισμένες σε χαρακτηριστικά, που χρησιμοποιούνται σε συνδυασμό με τεχνικές machine learning και περίπλοκα frameworks βελτιστοποίησης.

3.3 Εφαρμογές

Οι εφαρμογές της υπολογιστικής όρασης κυμαίνονται από την ανάπτυξη συστημάτων βιομηχανικής μηχανικής όρασης έως τη διερεύνηση τεχνητής νοημοσύνης και ανάπτυξη ρομπότ και ηλεκτρονικών υπολογιστών ικανών να κατανοήσουν τον κόσμο τριγύρω τους. Τα πεδία της υπολογιστικής όρασης και μηχανικής όρασης έχουν σημαντική αλληλεπικάλυψη. Η υπολογιστική όραση καλύπτει τη βασική τεχνολογία της αυτοματοποιημένης ανάλυσης εικόνας που χρησιμοποιείται σε πολλά πεδία.

Η μηχανική όραση αναφέρεται συνήθως σε μια διαδικασία συνδυασμού της αυτοματοποιημένης ανάλυσης εικόνας με άλλες μεθόδους και τεχνολογίες για την παροχή αυτοματοποιημένης επιθεώρησης και καθοδήγησης ρομπότ σε βιομηχανικές εφαρμογές. Σε πολλές εφαρμογές υπολογιστικής όρασης, οι υπολογιστές έχουν προπρογραμματιστεί για την επίλυση μια συγκεκριμένης εργασίας αλλά οι μέθοδοι που βασίζονται στη μάθηση γίνονται ολοένα συχνότερες. Παραδείγματα εφαρμογής της υπολογιστικής όρασης περιλαμβάνουν συστήματα για:

- Αυτοματοποιημένη επιθεώρηση, πχ σε βιομηχανικές εφαρμογές, γραμμές παραγωγής
 - Παροχή βοήθειας στους ανθρώπους σε εργασίες αναγνώρισης
 - Διαδικασίες ελέγχου
 - Ανίχνευση συμβάντων
 - Αλληλεπίδραση, πχ ως είσοδος σε μια συσκευή αλληλεπίδρασης ανθρώπου-μηχανής
 - Μοντελοποίηση αντικειμένων
 - Πλοήγηση, πχ σε αυτόνομα οχήματα
 - Οργάνωση πληροφοριών

Ένα από τα σημαντικότερα πεδία εφαρμογής της υπολογιστικής όρασης είναι αυτό της ιατρικής υπολογιστικής όρασης. Αυτός ο τομέας χαρακτηρίζεται από την εξαγωγή πληροφοριών από δεδομένα εικόνας με σκοπό την πραγματοποίηση ιατρικής διάγνωσης ενός ασθενούς. Γενικότερα, τα δεδομένα έχουν τη μορφή μικροσκοπικών εικόνων, εικόνων ακτίνων X, αγγειογραφικών εικόνων, εικόνων υπερήχου και εικόνων τομογραφίας. Από δεδομένα τέτοιου τύπου μπορεί να εξαχθεί ένα μεγάλο πλήθος δεδομένων όπως γιατί παράδειγμα, η ανίχνευση όγκων, αρτιοσκλήρυνσης κα. Είναι δυνατή επίσης η μέτρηση των διαστάσεων των οργάνων, της ροής αίματος κλπ.

Οι στρατιωτικές εφαρμογές είναι ίσως μία από τις μεγαλύτερες περιοχές της υπολογιστικής όρασης. Τα προφανή παραδείγματα είναι η ανίχνευση εχθρικών στρατιωτών ή οχημάτων και καθοδήγηση πυραύλων. Τα πιο προηγμένα συστήματα για την καθοδήγηση πυραύλων στέλνουν τον πύραυλο σε μια περιοχή αντί για έναν συγκεκριμένο στόχο και η επιλογή στόχου γίνεται όταν ο πύραυλος φτάσει στην περιοχή με βάση τα τοπικά δεδομένα εικόνας. Οι σύγχρονες στρατιωτικές έννοιες, όπως η ‘συνειδητοποίηση των πεδίων μάχης’, υποδηλώνουν ότι διάφοροι αισθητήρες, συμπεριλαμβανομένων των αισθητήρων εικόνας, παρέχουν μια πλούσια συλλογή πληροφοριών σχετικά με μια σκηνή μάχης που μπορεί να χρησιμοποιηθεί για τη στήριξη στρατηγικών αποφάσεων. Στην περίπτωση αυτή, η αυτόματη επεξεργασία των δεδομένων χρησιμοποιείται για να μειώσει την πολυπλοκότητα και να συγχωνεύσει πληροφορίες από πολλούς αισθητήρες για να αυξήσει την αξιοπιστία.

3.4 Μέθοδοι συστήματος υπολογιστικής όρασης

Η οργάνωση ενός συστήματος ηλεκτρονικής όρασης εξαρτάται σε μεγάλο βαθμό από την εφαρμογή. Ορισμένα συστήματα είναι αυτόνομες εφαρμογές που επιλύουν ένα συγκεκριμένο πρόβλημα μέτρησης ή ανίχνευσης, ενώ άλλα αποτελούν ένα υποσύστημα μεγαλύτερου σχεδιασμού το οποίο περιλαμβάνει για παράδειγμα υποσυστήματα για τον έλεγχο μηχανικών ενεργοποιητών, προγραμματισμό, βάσεις δεδομένων, μηχανικές διεπαφές κλπ. Η συγκεκριμένη εφαρμογή ενός συστήματος υπολογιστικής όρασης εξαρτάται επίσης από το εάν η λειτουργικότητά του είναι προκαθορισμένη ή αν κάποιο μέρος του μπορεί να μάθει ή να τροποποιηθεί κατά τη λειτουργία. Πολλές λειτουργίες είναι μοναδικές για την εφαρμογή. Υπάρχουν, ωστόσο, τυπικές λειτουργίες που εντοπίζονται σε πολλά συστήματα ηλεκτρονικής όρασης.

- **Συλλογή εικόνας** - Μια ψηφιακή εικόνα παράγεται από έναν ή περισσότερους αισθητήρες εικόνας. Ανάλογα με τον τύπο του αισθητήρα η εικόνα μπορεί να είναι είτε μία 2D εικόνα, είτε 3D. Μπορεί επίσης να είναι μια ακολουθία εικόνων.
- **Προεπεξεργασία** - Προτού να μπορεί να εφαρμοστεί μια μέθοδος ηλεκτρονικής όρασης σε δεδομένα εικόνας ώστε να εξαχθεί κάποια συγκεκριμένη πληροφορία, είναι συνήθως απαραίτητη η επεξεργασία των δεδομένων προκειμένου να διασφαλιστεί ότι ικανοποιεί ορισμένες υποθέσεις που υποδηλώνει η μέθοδος. Για παράδειγμα:
 - Επαναδειγματοληψία
 - Μείωση θορύβου
 - Βελτίωση αντίθεσης
- **Εξαγωγή χαρακτηριστικών** - Χαρακτηριστικά της εικόνας σε διάφορα επίπεδα πολυπλοκότητας εξάγονται από τα δεδομένα της. Τυπικά παραδείγματα τέτοιων χαρακτηριστικών είναι:
 - Γραμμές, ακμές και κορυφογραμμές
 - Σημεία τοπικού ενδιαφέροντος όπως γωνίες, κηλίδες ή σημεία
- **Ανίχνευση / τιμηματοποίηση** - Σε κάποιο σημείο της επεξεργασίας λαμβάνεται απόφαση σχετικά με το ποια σημεία εικόνας ή περιοχές της εικόνας είναι χρήσιμα για περαιτέρω επεξεργασία
 - Επιλογή συγκεκριμένου συνόλου σημείων ενδιαφέροντος
 - Τιμηματοποίηση μιας ή πολλαπλών περιοχών εικόνας που περιέχουν ένα

συγκεκριμένο αντικείμενο ενδιαφέροντος

- **Επεξεργασία υψηλού επιπέδου** - Σε αυτό το βήμα η είσοδος είναι συνήθως ένα μικρό σύνολο δεδομένων, για παράδειγμα ένα σύνολο σημείων ή μια περιοχή εικόνας που υποτίθεται ότι περιέχει ένα συγκεκριμένο αντικείμενο. Η υπόλοιπη επεξεργασία αφορά, για παράδειγμα:
 - Επαλήθευση ότι τα δεδομένα ικανοποιούν παραδοχές που βασίζονται σε μοντέλα και εφαρμογές
 - Εκτίμηση συγκεκριμένων παραμέτρων της εφαρμογής
 - Αναγνώριση εικόνων
 - Καταχώρηση εικόνας
- **Λήψη αποφάσεων** - Πραγματοποιείται η τελική απόφαση που απαιτεί η εφαρμογή όπως για παράδειγμα:
 - Pass / fail στις εφαρμογές αυτόματης επιθεώρησης
 - Match / no match σε εφαρμογές αναγνώρισης
 - Flag για περαιτέρω επιθεώρηση από άνθρωπο

3.5 Υπολογιστική όραση & FPGAs

Η πολυπλοκότητα των αλγορίθμων υπολογιστικής όρασης σε συνδυασμό με τη συνεχώς αυξανόμενη ζήτηση για εφαρμογές με τεράστια μεγέθη δεδομένων όπως για παράδειγμα εικόνες ή βίντεο υψηλής ανάλυσης, έχει οδηγήσει στην ανάγκη για μεγάλη υπολογιστική ισχύ. Οι επεξεργαστές γενικού σκοπού καθώς και σε αρκετές περιπτώσεις οι GPUs δε μπορούν να ικανοποιήσουν τις απαιτήσεις ισχύος. Συνεπώς, συστήματα όπως ASICs αποκτούν σημαντικό πλεονέκτημα στον τομέα της παρεχόμενης απόδοσης σε realtime εφαρμογές. Η προσέγγιση με ASICs όμως παρουσιάζει αρκετές δυσκολίες: το κόστος είναι υψηλό, ο σχεδιασμός τους είναι χρονοβόρος και μη επαναδιαμορφώσιμος μετά την κατασκευή τους.

Το γεγονός όμως ότι τα FPGA μπορούν να επαναδιαμορφώνονται τα κάνει κατάλληλα για αυτού του είδους εφαρμογές, ξεπερνώντας τους περιορισμούς που επιβάλλονται με τη χρήση των ASICs. Οι πρόσφατες βελτιώσεις στην τεχνολογία των FPGA σημαίνουν ότι μπορούν να επιτύχουν πολύ υψηλή απόδοση, κοντά στα επίπεδα των ASICs. Η ταυτόχρονη εκτέλεση των εντολών των επαναδιαμορφώσιμων συστή-

ματων τα καθιστά πλέον κατάλληλα για την υλοποίηση αλγορίθμων υπολογιστικής όρασης. Για παράδειγμα, μία από τις συνηθέστερες διεργασίες σε αλγορίθμους φηφιακής επεξεργασίας εικόνας ή σήματος είναι η συνέλιξη. Πολλά συστήματα μηχανικής όρασης χρησιμοποιούν δισδιάστατη συνέλιξη με πυρήνες μετασχηματισμών (3×3 , 5×5) για την εφαρμογή φίλτρων σε εικόνες με σκοπό την αναγνώριση ακμών κα. Σε έναν επεξεργαστή γενικού σκοπού, αυτού του είδους οι διεργασίες είναι χρονοβόρες απαιτώντας εκατομμύρια πολλαπλασιασμούς και προσθέσεις αλλά στα FPGA μπορεί να εκτελεστούν ταυτόχρονα.

Κεφάλαιο 4

Αλγόριθμος Αναγνώρισης Ακμών Canny

4.1 Κατάτμηση εικόνων

Στην υπολογιστική όραση, με τον όρο κατάτμηση εικόνων αναφερόμαστε στη διαδικασία της διαίρεσης μίας ψηφιακής εικόνας σε πολλαπλά τμήματα (σύνολα εικονοστοιχείων). Ο στόχος της είναι να απλοποιήσει και / ή να αλλάξει την αναπαράσταση της εικόνας σε κάτι πιο ουσιαστικό και ευκολότερο στην ανάλυση. Χρησιμοποιείται κυρίως για τον εντοπισμό αντικειμένων και ορίων (γραμμών, ακμών, κά). Πιο συγκεκριμένα, η κατάτμηση της εικόνας είναι η διαδικασία εκχώρησης μιας ετικέτας σε κάθε εικονοστοιχείο της, έτσι ώστε εικονοστοιχεία με την ίδια ετικέτα να μοιράζονται ορισμένα χαρακτηριστικά.

Στο παρόν κεφάλαιο θα ασχοληθούμε κυρίως με την ανίχνευση ακμών και πιο συγκεκριμένα με τον ανιχνευτή ακμών Canny.

4.2 Εισαγωγή στην αναγνώριση ακμών

Παρόλο που η αναγνώριση σημείων και γραμμών παίζουν σημαντικό ρόλο στην κατάτμηση εικόνων, η αναγνώριση ακμών είναι η πιο συνηθισμένη προσέγγιση για την αναγνώριση σημαντικών ασυνεχειών σε μια εικόνα. Η ανίχνευση ακμών περιλαμβάνει μια ποικιλία μαθηματικών μεθόδων που στοχεύουν στην αναγνώριση σημείων σε μια ψηφιακή εικόνα στην οποία η φωτεινότητά της αλλάζει απότομα ή εμφανίζει ασυνέχειες. Τα σημεία στα οποία αλλάζει έντονα η φωτεινότητα της εικόνας είναι συνήθως οργανωμένα σε ένα σύνολο τμημάτων καμπύλων γραμμών που ορίζονται ως άκρα. Το ίδιο πρόβλημα εντοπισμού ασυνεχειών σε μονοδιάστατα σήματα είναι

γνωστό ως ανίχνευση βημάτων και το πρόβλημα της ανεύρεσης ασυνεχειών σήματος με την πάροδο του χρόνου είναι γνωστό ως ανίχνευση αλλαγής. Η ανίχνευση ακμών είναι ένα θεμελιώδες εργαλείο στην επεξεργασία εικόνας, τη μηχανική όραση και την υπολογιστική όραση, ιδιαίτερα στους τομείς της ανίχνευσης χαρακτηριστικών και της εξαγωγής χαρακτηριστικών.

Ο σκοπός της ανίχνευσης αιχμηρών αλλαγών στη φωτεινότητα μιας εικόνας είναι η καταγραφή σημαντικών γεγονότων και αλλαγών στις ιδιότητες του κόσμου. Εύκολα διαπιστώνεται ότι κάτω από γενικές υποθέσεις, οι ασυνέχεις της φωτεινότητας σε μια εικόνα αντιστοιχούν σε:

- ασυνέχειες βάθους
- ασυνέχειες στον προσανατολισμό της επιφάνειας
- αλλαγές στις ιδιότητες των υλικών
- μεταβολές στο φωτισμό της σκηνής

Στην ιδανική περίπτωση, η εφαρμογή ενός ανιχνευτή ακμών μπορεί να οδηγήσει σε ένα σύνολο συνδεδεμένων καμπυλών που υποδεικνύουν τα όρια των αντικειμένων μίας και των επιφανειών μίας εικόνας. Η εφαρμογή ενός τέτοιου αλγορίθμου μειώνει σημαντικά την ποσότητα των προς επεξεργασία δεδομένων και συνεπώς μπορεί να φιλτράρει πληροφορίες που μπορούν να θεωρηθούν λιγότερο σημαντικές, διατηρώντας παράλληλα τις σημαντικές δομικές ιδιότητες της εικόνας. Εάν η διαδικασία της ανίχνευσης ακμών είναι επιτυχής, η ερμηνεία των πληροφοριών της εικόνας απλοποιείται ουσιαστικά. Στην πραγματικότητα όμως, είναι αρκετά δύσκολο να αποκτηθούν ιδανικές ακμές σε εικόνες της καθημερινής ζωής με μέτρια πολυπλοκότητα.

Οι ακμές που εξάγονται από εικόνες με μη σημαντικό περιεχόμενο, συχνά παρεμποδίζονται από κατακερματισμό, δηλαδή ακμές που δεν συνδέονται ώστε να σχηματίσουν τμήματα καμπυλών καθώς και ψευδείς ακμές που δεν αντιστοιχούν σε ενδιαφέροντα σημεία.

Η ανίχνευση ακμών είναι ένα από τα θεμελιώδη βήματα στην επεξεργασία, την ανάλυση και την αναγνώριση προτύπων της εικόνας καθώς και σε τεχνικές υπολογιστικής όρασης. Για το λόγο αυτό, θα ασχοληθούμε με τον αλγόριθμο αναγνώρισης ακμών Canny ο οποίος θα αναλυθεί στη συνέχεια.

4.2.1 Προσεγγίσεις

Υπάρχουν πολλές μέθοδοι ανίχνευσης ακμών, αλλά οι περισσότερες από αυτές μπορούν να ομαδοποιηθούν σε δύο κατηγορίες, βασισμένες σε αναζήτηση και zero-crossing. Οι μέθοδοι που ανήκουν στην πρώτη κατηγορία ανιχνεύουν τις ακμές, υπολογίζοντας πρώτα ένα μέτρο της 'δύναμης' της ακμής - συνήθως μια έκφραση της παραγώγου πρώτης τάξης - όπως το μέτρο της κλίσης και στη συνέχεια αναζητώντας τοπικά κατευθυντικά μέγιστα του μέτρου της κλίσης. Οι μέθοδοι της δεύτερης κατηγορίας αναζητούν zero crossings σε παραγώγους δεύτερης τάξης. Συνήθως χρησιμοποιούνται Laplacian zero crossings ή zero crossings μίας μη γραμμικής διαφορικής έκφρασης. Ως στάδιο προκαταρκτικής επεξεργασίας, εφαρμόζεται σχεδόν πάντα ένα στάδιο εξομάλυνσης, συνήθως εξομάλυνση Gaussian για την μείωση του θορύβου.

Οι μέθοδοι ανίχνευσης άκρων που έχουν δημοσιευθεί διαφέρουν κυρίως στους τύπους φίλτρων εξομάλυνσης που εφαρμόζονται και τον τρόπο με τον οποίο υπολογίζονται οι ακμές. Καθώς πολλές μέθοδοι βασίζονται στον υπολογισμό των διαβαθμίσεων της εικόνας, διαφέρουν επίσης και στους τύπους φίλτρων που χρησιμοποιούνται για τον υπολογισμό των εκτιμήσεων κλίσης στις x και y κατευθύνσεις.

4.3 Ανιχνευτής ακμών Canny

Ο ανιχνευτής ακμών Canny είναι ένας αλγόριθμος πολλαπλών σταδίων με στόχο τον υπολογισμό ενός μεγάλου εύρους ακμών σε μία εικόνα. Αναπτύχθηκε από τον John F. Canny¹ το 1986 με στόχο να είναι ένας άριστος ανιχνευτής ακμών. Αποτελεί ακόμα μία από τις πιο συνηθισμένες μεθόδους ανίχνευσης ακμών.

4.3.1 Ανάπτυξη του αλγορίθμου

Ο Canny διαπίστωσε ότι οι απαιτήσεις για την εφαρμογή ανίχνευσης ακμών σε διαφορετικά συστήματα όρασης είναι σχετικά παρόμοιες. Επομένως, μια τέτοια λύση μπορεί να εφαρμοστεί σε ένα ευρύ φάσμα καταστάσεων.

¹https://en.wikipedia.org/wiki/John_Canny

Τα γενικά κριτήρια ανίχνευσης ακμών περιλαμβάνουν:

- Ανίχνευση ακμής με χαμηλό ρυθμό σφάλματος, δηλαδή η ανίχνευση πρέπει να προσδιορίσει με ακρίβεια όσες περισσότερες ακμές μίας εικόνας μπορεί
- Το σημείο της ακμής που προσδιορίστηκε να είναι εντοπισμένο με ακρίβεια στο κέντρο της
- Μια δεδομένη ακμή της εικόνας θα πρέπει να σημειώνεται μόνο μία φορά και όπου είναι δυνατόν, ο θόρυβος της εικόνας δε πρέπει να δημιουργεί ψευδείς ακμές

Μεταξύ των μεθόδων ανίχνευσης ακμών που έχουν αναπτυχθεί μέχρι τώρα, ο αλγόριθμος του Canny είναι ένας από τις πιο αυστηρά καθορισμένες μεθόδους, παρέχοντας καλή και αξιόπιστη ανίχνευση. Λόγω της ικανότητάς του να πληροί τις τρεις προϋποθέσεις που αναφέρθηκαν παραπάνω, έγινε ένας από τους πιο δημοφιλείς αλγορίθμους ανίχνευσης ακμών.

4.3.2 Διαδικασία ανίχνευσης ακμών

Η διαδικασία της ανίχνευσης ακμών αναλύεται στα ακόλουθα βήματα:

1. Μετατροπή RGB εικόνας σε grayscale
2. Εφαρμογή Gaussian φίλτρου για τη μείωση του θορύβου της εικόνας και την εξομάλυνσή της
3. Γιολογισμός του μέτρου της κλίσης και της κατεύθυνσης της
4. Διαγραφή ακμών που δεν είναι τοπικά μέγιστα (Non-maximum suppression)
5. Εφαρμογή διπλής κατωφλίωσης για τον καθορισμό πιθανών ακμών
6. Σύνδεση ακμών με μέθοδο υστέρησης. Ολοκλήρωση της ανίχνευσης των ακμών, αφαιρώντας όλες όσες είναι αδύναμες και δε συνδέονται με ισχυρές ακμές

Στη συνέχεια θα αναλυθούν τα προαναφερθέντα στάδια του αλγορίθμου.

Μετατροπή RGB σε grayscale

Το πρώτο βήμα της διαδικασίας εύρεσης ακμών είναι η μετατροπή της RGB εικόνας σε grayscale. Αυτό γίνεται χυρίως για λόγους πολυπλοκότητας κατά τη διάρκεια της επεξεργασίας και μείωσης του συνολικού όγκου δεδομένων. Η μετατροπή αυτή είναι μία εύκολη διαδικασία και μπορεί να εφαρμοστεί σε εφαρμογές πραγματικού χρόνου.

Ενώ υπάρχουν πολλοί τρόποι για να μετατραπεί η αρχική εικόνα σε grayscale ο πιο συνηθισμένος είναι να ευρεθεί ο μέσος όρος των 3 καναλιών (R,G,B). Η μέθοδος αυτή όμως δεν παράγει ικανοποιητικά αποτελέσματα.

$$Gray = (Red + Green + Blue)/3$$

Η ITU-R² προτείνει τη μέθοδο Luma³, που είναι ο σταθμισμένος όρος των RGB συνιστωσών ως την καταλληλότερη μέθοδο για μετατροπή σε grayscale. Η μέθοδος αυτή δίνεται από τον ακόλουθο τύπο.

$$Gray = 0.2126 \times Red + 0.7152 \times Green + 0.00722 \times Blue$$



Σχήμα 4.1: Μετατροπή σε grayscale, μέσος όρος & μέθοδος Luma

Είναι εμφανές ότι η μέθοδος του μέσου όρου δε παράγει ικανοποιητικά αποτελέσματα καθώς δε λαμβάνει υπόψη της τα επίπεδα της φωτεινότητας του κάθε καναλιού.

²<https://en.wikipedia.org/wiki/ITU-R>

³[https://en.wikipedia.org/wiki/Luma_\(video\)](https://en.wikipedia.org/wiki/Luma_(video))

Gaussian φίλτρο

Δεδομένου ότι όλα τα αποτελέσματα ανίχνευσης ακμών επηρεάζονται εύκολα από το θόρυβο της εικόνας, είναι απαραίτητο να φιλτραριστεί για να αποτραπεί η φευδής ανίχνευση ακμών. Για την εξομάλυνση χρησιμοποιείται ένα Gaussian φίλτρο. Καθώς, ο μετασχηματισμός Fourier της Gaussian συνάρτησης δίνει άλλη μία Gaussian συνάρτηση, η εφαρμογή του θολώματος μειώνει τα υψηλής συχνότητας στοιχεία μίας εικόνας. Συνεπώς, μπορούμε να θεωρήσουμε το Gaussian blur ως ένα χαμηλοπερατό φίλτρο.

To Gaussian blur είναι ένας τύπος φίλτρου θολώματος που χρησιμοποιεί μία συνάρτηση Gaussian (που εκφράζει επίσης την κανονική κατανομή) για να υπολογίσει το μετασχηματισμό του κάθε εικονοστοιχείου της εικόνας. Η εξίσωση για το Gaussian φίλτρο σε δύο διαστάσεις δίνεται από:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (4.1)$$

όπου x είναι η απόσταση από τη αρχή του οριζόντιου άξονα, y η απόσταση από τη αρχή του κάθετου άξονα και σ η τυπική απόκλιση της Gaussian κατανομής. Με την αύξηση της τυπικής απόκλισης επιτυγχάνεται μεγαλύτερο θόλωμα της εικόνας όπως φαίνεται στο σχήμα.

Απαραίτητη προϋπόθεση του φίλτρου είναι ότι το άθροισμα όλων των συντελεστών του πρέπει να ισούται με 1. Επομένως, για την αποφυγή αύξησης της φωτεινότητας της εικόνας πρέπει να κανονικοποιηθεί. Αυτό επιτυγχάνεται μέσω της πρόσθεσης όλων των συντελεστών και της διαιρέσης τους με το άθροισμα αυτό.

Για παράδειγμα, ένα συμμετρικό φίλτρο μεγέθους $3x3$ και $\sigma = 1$ είναι το ακόλουθο:

$$G = \begin{bmatrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{bmatrix} = \frac{1}{0.7799} \times \begin{bmatrix} 0.0586 & 0.0966 & 0.0586 \\ 0.0966 & 0.1592 & 0.0966 \\ 0.0586 & 0.0966 & 0.0586 \end{bmatrix}$$

Μετά τον υπολογισμό του πυρήνα μετασχηματισμού, το θόλωμα εφαρμόζεται στην εικόνα με τη μέθοδο της συνέλιξης. Το κύριο εικονοστοιχείο λαμβάνει τη μεγαλύτερη τιμή καθώς πολλαπλασιάζεται με το μεγαλύτερο συντελεστή του φίλτρου (κεντρική τιμή) ενώ τα γειτονικά λαμβάνουν μικρότερες τιμές όσο η απόστασή τους από το κεντρικό εικονοστοιχείο αυξάνει. Ως αποτέλεσμα, η διαδικασία αυτή οδηγεί σε θόλωμα που διατηρεί τα σύνορα και τις ακμές. Η διαδικασία της συνέλιξης με το φίλτρο παρουσιάζεται συνοπτικά στη συνέχεια:

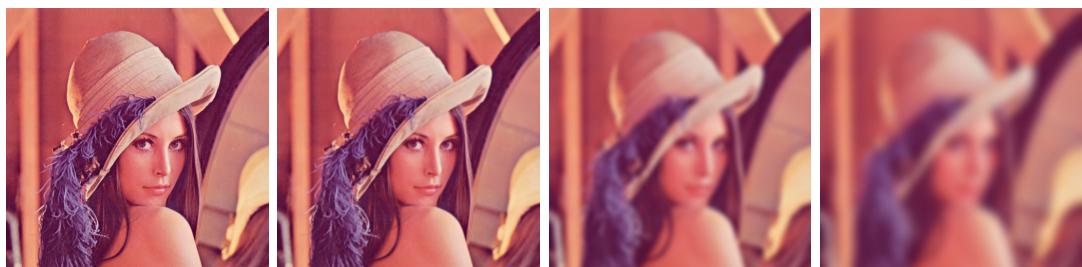
Algorithm 1: Συνέλιξη

```

1 for  $i = 1$  to  $m - 1$  do
2   for  $j = 1$  to  $n - 1$  do
3      $sum = 0$ ; for  $j = -1$  to  $+1$  do
4       for  $j = -1$  to  $+1$  do
5          $sum = sum + k(j, i) \times f(x - j, y - i)$ 
6       end
7     end
8   end
9 end

```

Παραδείγματα εικόνων μετά την εφαρμογή θολώματος με $\sigma = 1, \sigma = 5$ και $\sigma = 10$.



(a) Είσοδος

(b) $\sigma = 1$ (c) $\sigma = 5$ (d) $\sigma = 10$ Σχήμα 4.2: Αποτελέσματα Canny για $thresh = 0.2$

Gradient

Η ακμή μίας εικόνας μπορεί να δείχνει σε διάφορες κατευθύνσεις, οπότε ο αλγόριθμος χρησιμοποιεί φίλτρα για να ανιχνεύσει τις κάθετες και οριζόντιες ακμές στη θολή εικόνα. Ο τελεστής ανίχνευσης ακμών (Roberts, Sobel ή Prewitt) επιστρέφει μια τιμή για την πρώτη παράγωγο στην οριζόντια κατεύθυνση - G_x και στην κάθετη κατεύθυνση - G_y . Από τα δεδομένα αυτά μπορούμε να υπολογίσουμε το μέτρο της ακμής καθώς και την κατεύθυνσή της. Ο τελεστής χρησιμοποιεί δύο 3×3 πυρήνες που συνελίσσονται με την εικόνα για να υπολογίσει προσεγγιστικά τις παραγώγους.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Ένας δεύτερος τρόπος για την εύρεση των ακμών είναι η απευθείας εύρεση των πρώτων παραγώγων χωρίς τη χρήση φίλτρων. Η κλίση μίας εικόνας $f(x, y)$ στην τοποθεσία (x, y) μπορεί να οριστεί ως το διάνυσμα:

$$\nabla(f) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (4.2)$$

Και στις δύο περιπτώσεις το μέτρο και η κατεύθυνση της κλίσης υπολογίζεται με τη βοήθεια των ακόλουθων εξισώσεων:

$$G = \sqrt{G_x^2 + G_y^2} \quad (4.3)$$

$$\theta = \arctan(G_y, G_x) \quad (4.4)$$

Non-maximum suppression

Η μέθοδος Non-maximum suppression είναι μία τεχνική αραιώσης της ακμής. Σκοπός αυτού του βήματος είναι η μετατροπή των 'θολών' ακμών που ανιχνεύθηκαν σε 'αιχμηρές'. Αυτό επιτυγχάνεται με τη διατήρηση όλων των τοπικών μεγίστων και τη διαγραφή όλων των άλλων.

- Στρογγυλοποίηση της κατεύθυνσης της κλίσης στις πλησιέστερες 45 μοίρες, που αντιστοιχεί στη χρήση μια συνδεδεμένης γειτονιάς 8 εικονοστοιχείων

- Σύγκριση του μέτρου της κλίσης του τρέχοντος εικονοστοιχείου με το μέτρο του εικονοστοιχείου στην κατεύθυνση θετικής και αρνητικής κλίσης. Για παράδειγμα, εάν η κατεύθυνση της κλίσης είναι βόρεια (90°) τότε συγκρίνεται με τα εικονοστοιχεία προς το βορρά και το νότο
- Εάν το μέτρο του τρέχοντος εικονοστοιχείου είναι το μεγαλύτερο, διατηρείται η τιμή του, αλλιώς αφαιρείται.

Κατωφλίωση

Μετά την εφαρμογή της NMS, τα εναπομείναντα εικονοστοιχεία ακμών παρέχουν μία πιο ακριβή αναπαράσταση των πραγματικών ακμών της εικόνας. Ωστόσο, παραμένουν κάποιες ακμές που προκλήθηκαν από το θόρυβο και τις μεταβολές στο χρώμα της εικόνας. Προκειμένου να ληφθούν υπόψη οι φευδείς αυτές απεικονίσεις, είναι απαραίτητο να φιλτραριστούν τα εικονοστοιχεία με αδύνατο μέτρο κλίσης και να διατηρηθούν αυτά που παρουσιάζουν υψηλή τιμή. Αυτό κατορθώνεται επιλέγοντας τιμές υψηλού και χαμηλού κατωφλίου.

Εάν το μέτρο κλίσης μιας ακμής είναι υψηλότερο από το υψηλό κατώφλι, σημειώνεται ως ακμή, αλλιώς εάν το μέτρο του είναι μικρότερο από το υψηλό κατώφλι και μεγαλύτερο από το χαμηλό κατώφλι σημειώνεται ως εικονοστοιχείο ασθενούς ακμής. Τέλος, αν η τιμή του μέτρου είναι μικρότερη από την τιμή του χαμηλού κατωφλίου, το εικονοστοιχείο καταστέλλεται. Οι δύο τιμές κατωφλίων καθορίζονται εμπειρικά και ο ορισμός τους θα εξαρτηθεί από το περιεχόμενο μιας δεδομένης εικόνας εισόδου.

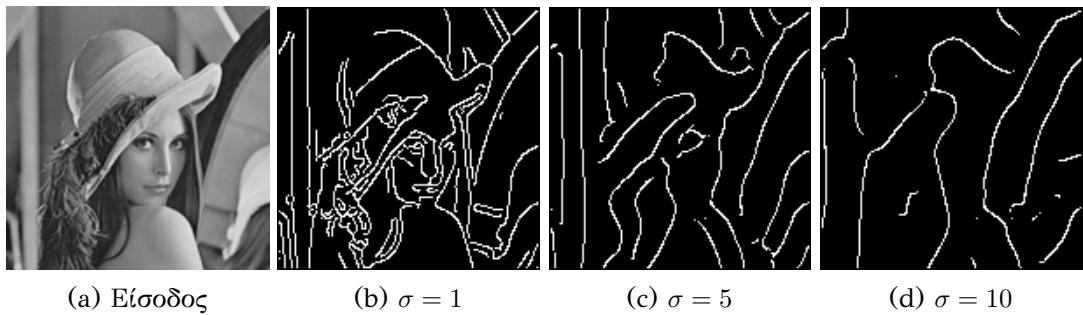
Ένωση ακμών με υστέρηση

Μέχρι στιγμής, τα εικονοστοιχεία που έχουν κατηγοριοποιηθεί ως ισχυρές ακμές θα πρέπει σίγουρα να συμμετέχουν στην τελική εικόνα, καθώς έχουν προκύψει από τις πραγματικές ακμές της αρχικής εικόνας. Ωστόσο, θα πρέπει να παρθεί απόφαση σχετικά με τις ασθενείς ακμές που είναι πιθανό να έχουν προκύψει από θόρυβο. Για να επιτευχθεί, επομένως, ένα ακριβές αποτέλεσμα οι αδύναμες ακμές πρέπει να αφαιρεθούν.

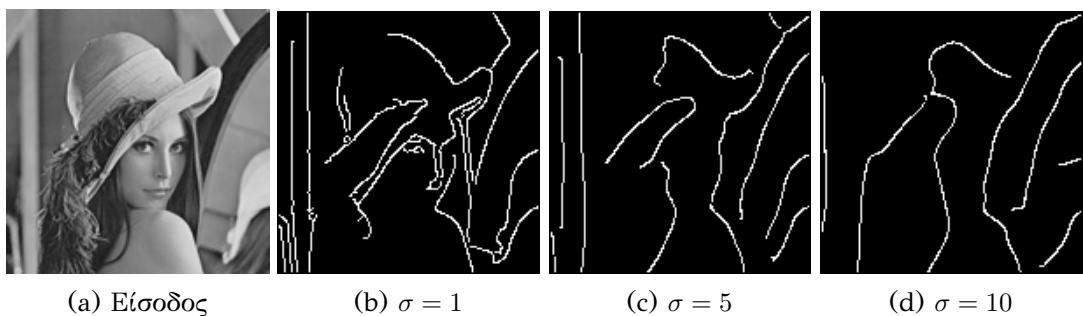
Συνήθως ένα εικονοστοιχείο ασθενούς ακμής που προκαλείται από τις πραγματικές άκρες θα συνδέεται με μία ισχυρή ακμή. Για να συνδεθούν οι ακμές, μπορεί

να εφαρμοστεί ανάλυση blob⁴ παρατηρώντας τα γειτονικά εικονοστοιχεία. Όσο υπάρχει μια ισχυρή ακμή που εμπλέκεται στην κηλίδα, αυτό το ασθενές άκρο μπορεί να αναγνωριστεί ως ένα που πρέπει να διατηρηθεί.

4.3.3 Παραδείγματα εφαρμογής αλγορίθμου



Σχήμα 4.3: Αποτελέσματα Canny για $thresh = 0.2$



Σχήμα 4.4: Αποτελέσματα Canny για $thresh = 0.5$

Παρατηρούμε ότι αυξάνοντας την τυπική απόκλιση του γκαουσιανού θολώματος, αρκετές σημαντικές ακμές της αρχικής εικόνας δεν αναγνωρίζονται. Αντίστοιχα, το ίδιο ισχύει και για το κατώφλι που έχουμε ορίσει. Στην πρώτη περίπτωση, το κατώφλι είναι κανονικοποιημένο ως προς το 255, δηλαδή ή κανονική τιμή του είναι $thresh = 0.2 \times 255 = 51$. Οι ακμές που έχουν τιμή μικρότερη του κατωφλίου απορρίπτονται. Στη δεύτερη περίπτωση, το κατώφλι είναι ίσο με $thresh = 0.5 \times 255 = 122.5$ και παρατηρούμε ότι αναγνωρίζονται πολύ λιγότερες ακμές.

Καταλήγοντας, πρέπει να οριστεί μία τιμή για την τυπική απόκλιση καθώς και μία για το κατώφλι, οι οποίες θα δίνουν σε σταθερή βάση ικανοποιητικά αποτελέσματα. Οι τιμές αυτές βρίσκονται μετά από δοκιμές και εξαρτώνται συνήθως από τη

⁴https://en.wikipedia.org/wiki/Connected-component_labeling

φωτογραφία. Συνήθως, επιλέγουμε $\sigma = 1.5$ και $thresh = 45$.

Κεφάλαιο 5

Υλοποίηση

5.1 Περιγραφή & Ανάλυση Προβλήματος

Στην παρούσα διπλωματική εργασία, ο τελικός στόχος είναι η υλοποίηση ενός ενσωματωμένου συστήματος για την αναγνώριση ακμών με τη μέθοδο του Canny. Αρχικά, θα αξιολογηθεί η σύνθεση της βασισμένης σε C++ υλοποίησης και στη συνέχεια θα υλοποιηθεί ένα πλήρες σύστημα για τη μεταφορά δεδομένων ανάμεσα στην προγραμματιζόμενη λογική και τον ενσωματωμένο επεξεργαστή. Για την επίτευξη του στόχου μας χρησιμοποιήθηκε η πλατφόρμα της Xilinx, Vivado 2016.4 που περιλαμβάνει τα απαραίτητα εργαλεία. Αυτά είναι το HLS, Vivado και SDK. Στις επόμενες υποενότητες θα γίνει εκτενέστερη αναφορά στα εργαλεία αυτά. Επιπλέον στόχοι της εργασίας είναι η βελτιστοποίηση της απόδοσης του αλγορίθμου και η αξιολόγηση της πλατφόρμας Vivado. Στη συνέχεια θα γίνει αναφορά σε όλα τα βήματα της εργασίας με τη σειρά που πραγματοποιήθηκαν καθώς και στα αποτελέσματα της υλοποίησης. Η ροή της σχεδίασης που πραγματοποιήθηκε είναι η εξής:

- Συγγραφή, προσημοίωση και εξαγωγή IP μέσω του Vivado HLS - Ενότητα 5.2
- Δημιουργία πλήρους συστήματος και παραγωγή bitstream μέσω του Vivado - Ενότητα 5.3
- Συγγραφή κώδικα για τον ARM και προγραμματισμός FPGA μέσω του Xilinx SDK - Ενότητα 5.4

Οι προδιαγραφές που έχουμε ορίσει για την εφαρμογή μας είναι η είσοδος δεδομένων (εικόνα) στο FPGA τα οποία είναι αποθηκευμένα σε BRAM, η επιλογή μίας τιμής κατωφλίου, η επεξεργασία τους και στο τέλος η έξοδος της επεξεργασμένης εικόνας πίσω στον επεξεργαστή, όπου θα γίνει και η τελική επαλήθευση. Για τη μεταφορά των δεδομένων αποφασίσαμε να χρησιμοποιήσουμε την AXI4-Stream διεπαφή

με τη βοήθεια ενός AXI DMA Controller και το AXI4-Lite για τη μεταφορά της τιμής κατωφλίου. Αρκετά συχνά γίνεται συνδυασμός του AXI4-Stream και των Memory Mapped Protocols. Μια διαφορετική προσέγγιση είναι να αναπτυχθούν συστήματα που συνδυάζουν AXI4-Stream και AXI memory mapped IPs. Συνήθως επιλέγεται ένας ελεγκτής DMA για να μεταφέρει δεδομένα από και προς τη μνήμη. Θα γίνει εκτενέστερη αναφορά στην επόμενη ενότητα. Στο τέλος του κεφαλαίου θα αναλυθούν τα αποτελέσματα της υλοποίησης.

5.2 HLS

Πρώτο βήμα της εργασίας ήταν η ανάπτυξη του αλγορίθμου σε C++ ώστε να ληφθούν τα πρώτα αποτελέσματα και να επιπλυθούν τυχόν προβλήματα. Για την ανάπτυξη χρησιμοποιήθηκε η πλατφόρμα της Microsoft, Visual Studio Community 2015 η οποία παρέχει μεγάλο πλήθος εργαλείων (debugging, profiling κα). Μελετήθηκαν διάφοροι τρόποι για την υλοποίηση και επιλέχθηκαν αυτοί που θεωρήθηκαν καταλληλότεροι. Αναπτύχθηκε παράλληλα ένα πρόγραμμα για την φόρτωση και αποθήκευση εικόνων ώστε να διευκολυνθεί η αποσφαλμάτωση του κώδικα μας.

Η πρώτη πρόκληση που έπρεπε να αντιμετωπιστεί ήταν η μεταφορά του πρωτότυπου προγράμματος από το Visual Studio στο HLS. Για το λόγο αυτό, αφιερώθηκε αρκετός χρόνος για τη μελέτη της τεχνολογίας των FPGA, της αναπτυξιακής πλακέτας ZC702 και της πλατφόρμας Vivado. Έχει γίνει αναφορά στα δύο πρώτα στο δεύτερο κεφάλαιο της εργασίας. Στην παρούσα ενότητα θα γίνει αναφορά στο HLS και στη διαδικασία που ακολουθήθηκε για την ανάπτυξη του αλγορίθμου. Τα στάδια της ροής σχεδιασμού ενός αλγορίθμου είναι τα άκολουθα:

1. Μεταφορά αλγορίθμου στο HLS & συγγραφή της top συνάρτησης
2. Ανάπτυξη testbench
3. Προσομοίωση αλγορίθμου σε C
4. Σύνθεση
5. Προσομοίωση σε RTL επίπεδο
6. Εξαγωγή κώδικα HDL ή IP

Παρόλο που το Vivado HLS υποστηρίζει ένα μεγάλο μέρος της C, μερικές εντολές και δομές μνήμης δεν είναι συνθέσιμες ή μπορούν να προκαλέσουν σφάλματα κατά τη ροή του σχεδιασμού. Για να μπορεί μία συνάρτηση να είναι συνθέσιμη - όπως αναφέρθηκε στο δεύτερο κεφάλαιο - πρέπει:

- Να περιέχει το σύνολο της λειτουργικότητας της σχεδίασης
- Να μην βασίζεται σε κλήσεις του συστήματος για την πραγματοποίηση λειτουργιών
- Οι δομές μνήμης να είναι δεδομένου μεγέθους και όχι δυναμικές

5.2.1 Δομή κώδικα & Top Function

Το συνολικό πρόγραμμα έχει διασπαστεί σε επιμέρους συναρτήσεις ώστε να διευκολυνθεί η αποσφαλμάτωσή του προκειμένου να διασφαλιστεί η αποτελεσματικότερη βελτιστοποίηση του κώδικα. Η ροή του αλγορίθμου φαίνεται στη συνέχεια.



Σχήμα 5.1: Ροή αλγορίθμου Canny Edge Detection

Εκτός της κύριας συνάρτησης έχουν υλοποιηθεί οι ακόλουθες συναρτήσεις: `gaussian()`, `grad()`, `edgeID()`. Στην κύρια συνάρτηση καλούνται οι προαναφερθείσες συναρτήσεις για την παραγωγή του δεδομένων και τέλος πραγματοποείται η έξοδος της επεξεργασμένης εικόνας προς το AXI4-Stream ώστε να ληφθεί από το υπολογιστικό σύστημα.

Στο header αρχείο του προγράμματος έχουν οριστεί οι δομές δεδομένων που θα χρησιμοποιηθούν από το πρόγραμμά μας και έχουν αποθηκευθεί στη BRAM του FPGA.

Κύρια συνάρτηση - canny()

Η κύρια συνάρτηση δέχεται σαν ορίσματα τα instances του `hls::stream` για την είσοδο και έξοδο των δεδομένων και την τιμή του κατωφλίου. Με τη βοήθεια κατάλληλων pragmas ορίζονται οι απαραίτητες διεπαφές του AXI4-Stream και AXI4-Lite. Στη συνέχεια η κλήση των βοηθητικών συναρτήσεων. Τέλος, πραγματοποιείται η έξοδος των δεδομένων προς τον DMA controller.

gaussian()

Η συνάρτηση αυτή αποτελεί την πιο απαιτητική ρουτίνα του αλγορίθμου μας. Πραγματοποείται η συνέλιξη της εικόνας με τον πυρήνα μετασχηματισμού Gauss όπως αυτή αναλύθηκε στην προηγούμενη ενότητα. Υπάρχουν 4 εμφωλευμένοι βρόχοι που προσπελαύνουν την εικόνα και τον πυρήνα στις δύο διαστάσεις. Όπως θα δούμε στη συνέχεια, η συγκεκριμένη συνάρτηση έχει το μεγαλύτερο αντίκτυπο στην απόδοση του IP μας.

grad()

Σκοπός της συνάρτησης είναι η εύρεση του διανύσματος κλίσης της εικόνας με μία από τις μεθόδους που αναλύθηκαν στο προηγούμενο κεφάλαιο. Επιλέχθηκε η μέθοδος της εύρεσης των πρώτων παραγώγων ώστε να μειωθούν οι πόροι που απαιτούνται. Υπολογίζει στη συνέχεια το μέτρο της κλίσης από τα διανύσματα κλίσης και το αποθηκεύει. Το αποτέλεσμα που θα προκύψει είναι μία πρώιμη μορφή της τελικής εικόνας.

edgeID()

Η τελική συνάρτηση που καλείται αποσκοπεί στην εύρεση των ακμών της εικόνας. Αρχικά, υπολογίζει την κατεύθυνση τους - πάνω, κάτω, αριστερά και δεξιά. Στη συνέχεια εφαρμόζει η τενχική non-maximum suppression και κατωφλίωση ώστε να προκύψει η τελική εικόνα.

Για την πλήρη υλοποίηση χρησιμοποιήθηκαν τέσσερις μονοδιάστατοι πίνακες μεγέθους 125×125 τύπου `uint8_t` για την είσοδο, έξοδο και μέτρο της κλίσης και `int8_t` για τα διανύσματα των κλίσεων τα οποία μπορούν να λάβουν και αρνητικές τιμές. Η κάθε συναρτήση είχε σαν όρισμα τον αντίστοιχο δείκτη του κάθε πίνακα. Λόγω των περιορισμών του Vivado HLS, υπάρχουν δύο τρόποι για να κληθούν οι συναρτήσεις. Ο πρώτος τρόπος είναι με το πλήρες όνομα του πίνακα μαζί με το μέγεθος του και ο δεύτερος με έναν δείκτη. Οι πίνακες ορίστηκαν σαν static με αποτέλεσμα να αποθηκεύονται από τον 1ο κύκλο του ρολογιού στη BRAM του FPGA, εξοικονομώντας κατ' αυτόν τον τρόπο πολύτιμο latency.

Για την καλύτερη αξιοποίηση του FPGA, πραγματοποιήθηκαν βελτιστοποιήσεις και μετατροπές συγκεκριμένων αλγορίθμων - θα παρουσιαστούν στο επόμενο κεφάλαιο - ώστε να επιτευχθεί ένα ικανοποιητικό ποσοστό επιτάχυνσης.

5.2.2 Testbench

Ο έλεγχος της σωστής λειτουργίας του υλικού είναι αναγκαίος και αποτελεί αναπόσπαστο κομμάτι του HLS. Για να ελέγξουμε επομένως το υλικό που παράγεται, αναπτύσσουμε ένα αρχείο testbench. Το αρχείο αυτό περιλαμβάνει την `main()` στην οποία πραγματοποιείται ο έλεγχος.

Δημιουργούμε ένα αρχείο και το ονομάζουμε `test.cpp`. Το αρχείο αυτό θα περιέχει κώδικα υπεύθυνο για τον έλεγχο της ορθής λειτουργίας της συνάρτησής μας. Για την αποθήκευση των επεξεργασμένων εικόνων και τη φόρτωση της εικόνας που περιέχει τα ‘golden data’ χρησιμοποιήθηκε η open-source βιβλιοθήκη EasyBMP [18] η οποία περιέχει τις απαραίτητες συναρτήσεις και δομές δεδομένων για τη φόρτωση και την αποθήκευση εικόνων. Οι λειτουργίες του testbench είναι οι εξής:

- Κλήση κύριας συνάρτησης
- Λήψη δεδομένων μετά την επεξεργασία τους
- Σύγκριση με τα αποτελέσματα της συνάρτησης που εκτελείται στον ΗΥ
- Αποθήκευση εικόνας στον ΗΥ για οπτική αξιολόγηση

5.2.3 Δημιουργία Project & Solutions

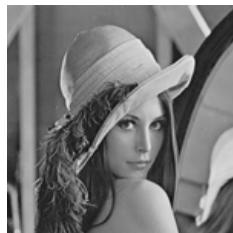
Μετά την δημιουργία όλων των απαραίτητων αρχείων, δημιουργούμε ένα νέο project στο Vivado HLS, ορίζοντας τα αρχεία πηγαίου κώδικα - `canny.cpp` και τα testbench αρχεία - `test.cpp` & `EasyBMP.cpp`. Ορίζουμε επίσης, την κύρια συνάρτηση, στη συγκεκριμένη περίπτωση είναι η `canny()`. Στη συνέχεια ορίζουμε το FPGA ZC702 και περίοδο ρολογιού ίση με 10 ns.

Μπορούμε επίσης να δημιουργήσουμε πολλαπλά solutions, όπου θα δοκιμάσουμε διάφορες βελτιστοποιήσεις του κώδικα μας ώστε να αποφασίσουμε ποια υλοποίηση θα χρησιμοποιήσουμε στη συνέχεια.

5.2.4 Προσομοίωση με κώδικα C

Η εικόνα grayscale που θα επεξεργαστούμε είναι ανάλυσης 125×125 με μέγεθος 67.584 bytes - 66.2 KB. Τα δεδομένα της εικόνας είναι αποθηκευμένα στον πίνακα **input** που έχει οριστεί σε ένα header αρχείο.

Εκτελώντας το C Simulation, λαμβάνουμε το αποτέλεσμα του αλγορίθμου μας, το οποίο συγχρίνεται με τα "golden data" που παράγονται από την εκτέλεση του testbench. Επιπλέον, η επεξεργασμένη εικόνα αποθηκεύεται στον ΗΥ ώστε να μπορεί να αξιολογηθεί η ποιότητα της αναγνώρισης ακμών που πραγματοποιήθηκε. Η εικόνα εισόδου και η επεξεργασμένη εικόνα φαίνονται στη συνέχεια:



Σχήμα 5.2: 125×125 Grayscale Lena

5.2.5 Σύνθεση

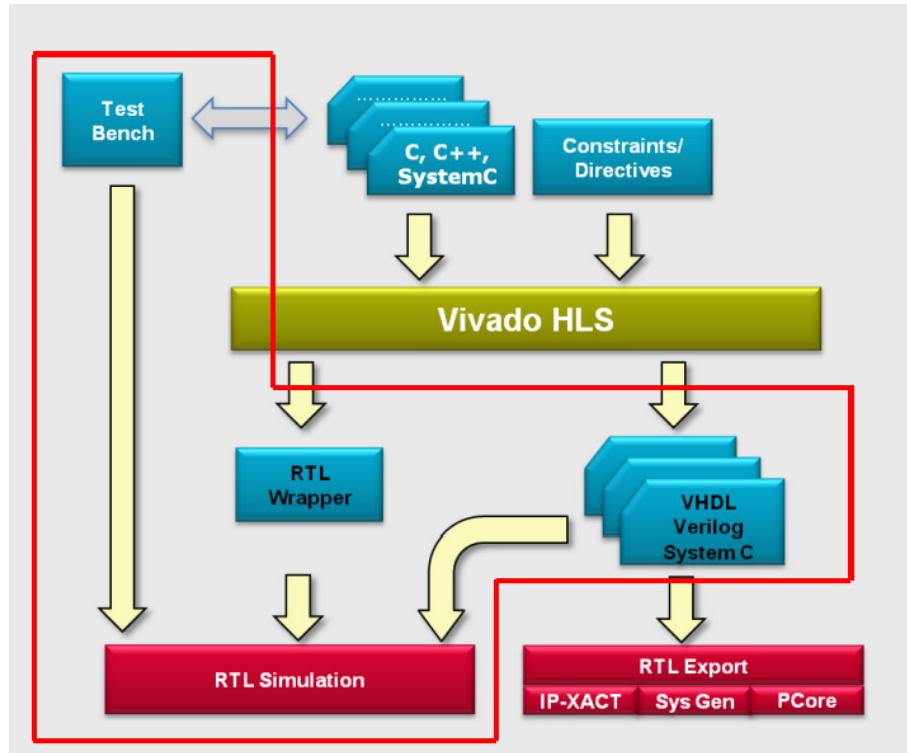
Το επόμενο βήμα μετά την επιτυχή προσομοίωση της συνάρτησής μας είναι η σύνθεσή της. Αν όλα είναι σωστά, παράγεται κώδικας VHDL ή Verilog καθώς επίσης μία αναφορά όπου φαίνεται η αξιοποίηση των πόρων και οι χρονισμοί όλων των instances του προγράμματος μας. Σε αντίθετη περίπτωση, το πρόγραμμα εμφανίζει τα σφάλματα και τις παρατηρήσεις ώστε αυτές να διορθωθούν και να επαναληφθεί η διαδικασία της σύνθεσης.

Σε αυτό το σημείο μπορούν να δημιουργηθούν νέα solutions (δεν είναι αναγκαίο) και να εφαρμοστούν βελτιστοποιήσεις που θα βελτιώσουν την απόδοση του υλικού που σχεδιάστηκε.

5.2.6 RTL Co-Simulation

Μετά την επιτυχή βελτίωση του προγράμματος μας και την επίτευξη των προδιαγραφών που έχουμε ορίσει πραγματοποιούμε την προσομοίωση σε επίπεδο RTL.

Σε αυτό το στάδιο προσομοιώνεται ο κώδικας με τη βοήθεια του testbench με τη διαφορά ότι σε αυτό το στάδιο ελέγχεται το hardware (VHDL, Verilog, SystemC) που παράχθηκε στο προηγούμενο βήμα. Υπάρχει υποστήριξη για 3rd party προσομοιωτές όπως τα XSim, ISim.



Σχήμα 5.3: Ροή RTL CO-SIM [15]

Τα αποτελέσματα της προσομοίωσης εμφανίζονται στην κονσόλα και παράγεται επίσης μία αναφορά που μας ενημερώνει για την επιτυχία/αποτυχία της προσομοίωσης καθώς και για τους κύκλους ρολογιού που απαιτεί το πρόγραμμα μας.

Πίνακας 5.1: Αναφορά Co-Sim

RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Verilog	93304	93304	93304	N/A	N/A	N/A	N/A

5.2.7 IP Export

Τελικό βήμα στη ροή σχεδιασμού HLS είναι η εξαγωγή του υλικού για την χρήση του. Οι διαθέσιμοι τύποι είναι οι ακόλουθοι:

- IP-XACT για τη χρήση στο Vivado - Σειρά 7 & Zynq μόνο
- System Generator IP block - Σειρά 7 & Zynq μόνο
- Pcore IP block για χρήση σε EDK - Σειρά 7, Zynq, Spartan-3, Spartan-6, Virtex-4/5/6

Επιλέχθηκε η πρώτη περίπτωση - IP-XACT καθώς διευκολύνει κατά μεγάλο βαθμό τη διαδικασία δημιουργίας του πλήρους συστήματος στο Vivado που θα αναλυθεί στη συνέχεια.

5.2.8 TCL

Η γλώσσα TCL - Tool Command Language είναι μία γλώσσα προγραμματισμού ενσωματωμένη στη πλατφόρμα Vivado. Αποτελεί μία από τις πρότυπες γλώσσες στη βιομηχανία των ημιαγωγών σε εφαρμογές προγραμματισμού διεπαφών.

Η TCL επιτρέπει την εκτέλεση αυτοματοποιημένων scripts σε όλα τα στάδια της σχεδιαστικής ροής στην πλατφόρμα Vivado, γεγονός που την καθιστά υπερβολικά χρήσιμη. Η μεταφορά του προγράμματος σε κάποιο άλλο υπολογιστικό σύστημα, η εκτέλεση της σύνθεσης, η προσομοίωση και η εξαγωγή μπορούν να πραγματοποιηθούν χωρίς την εκτέλεση των προγραμμάτων σε γραφικών περιβάλλον αλλά από το terminal/cmd, εξοικονομώντας έτσι πολύτιμο χρόνο.

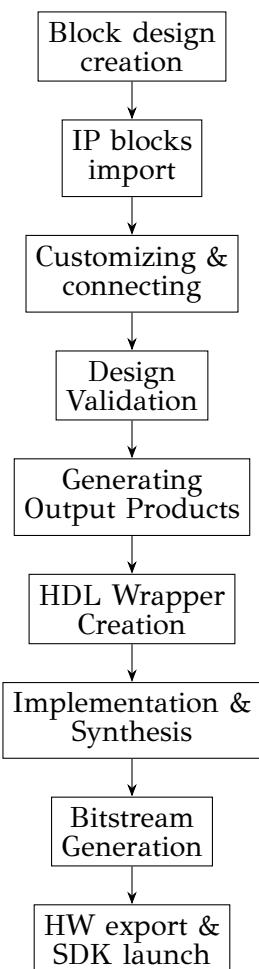
Το HLS παράγει από μόνο του δύο αρχεία TCL: `script.tcl` και `directives.tcl`. Το πρώτο αρχείο είναι υπεύθυνο για τη προσομοίωση, τη σύνθεση και την εξαγωγή του υλικού ενώ στο δεύτερο είναι αποθηκευμένα τα διάφορα `#pragmas` για τη βελτιστοποίηση της σύνθεσης.

Με τη βοήθεια του command prompt του Vivado και την εντολή `vivado_hls -f script.tcl` μπορούμε να εκτελέσουμε όλες τις παραπάνω λειτουργίες χωρίς τη χρήση κάποιου GUI.

5.3 Vivado IP Integrator

To Vivado IP Integrator επιτρέπει στο σχεδιαστή να δημιουργεί πολύπλοκα σχέδια συστημάτων διασυνδέοντας IP μέσω του IP καταλόγου της Xilinx και του ίδιου του σχεδιαστή. Τα σχέδια μπορούν να σχεδιαστούν διαδραστικά μέσω του γραφικού περιβάλλοντος αλλά και μέσω TCL αρχείων. Το Vivado IPI δίνει τη δυνατότητα κατασκευής του σχεδιασμού στο επίπεδο διεπαφής καθώς και στο επίπεδο πυλών όπου χρειάζεται περισσότερη ακρίβεια.

Μια διεπαφή, όπως για παράδειγμα το AXI4-Stream, περιέχει έναν μεγάλο αριθμό σημάτων και διασυνδέσεων. Αν κάθε σήμα ή διασύνδεση ήταν ορατά σε ένα IP block, τότε το συνολικό block design θα ήταν υπερβολικά περίπλοκο. To Vivado ομαδοποιεί αυτά τα σήματα και διασυνδέσεις ώστε να αντιμετωπιστεί αυτό το πρόβλημα. Στο ακόλουθο σχήμα φαίνεται μία απλοποιημένη ροή σχεδιασμού στο Vivado.



Σχήμα 5.4: Βασική ροή σχεδιασμού στο Vivado IPI

5.3.1 Δημιουργία Project

Στην αρχική οθόνη του Vivado, επιλέγουμε τη δημιουργία νέου project. Στο παράθυρο που εμφανίζεται:

- (a) Διαλέγουμε ένα όνομα και στη συνέχεια πατάμε στο **Next**
- (b) Σαν τοποθεσία αποθήκευσης του project επιλέγουμε το φάκελο που περιέχει το HLS project
- (c) Επιλέγουμε το **Do not specify sources at this time**
- (d) Πατάμε **Next**

Στη σελίδα Default Part:

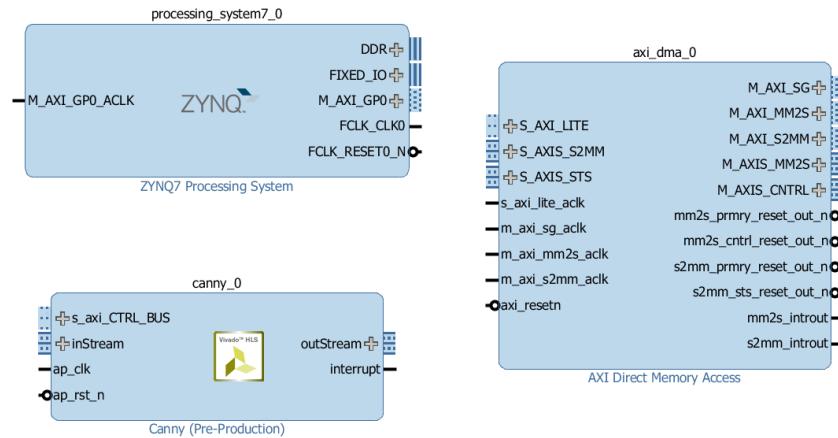
- (a) Πατάμε στο **Boards**
- (b) Επιλέγουμε το **ZYNQ-7 ZC702 Evaluation Board**
- (c) Πατάμε **Next** και στη συνέχεια **Finish**

5.3.2 Δημιουργία Σχεδίου & Εισαγωγή IP

Σκοπός αυτού του βήματος είναι η εισαγωγή του IP block που σχεδιάσαμε με τη βοήθεια του Vivado HLS και η δημιουργία ενός block design όπου θα πραγματοποιηθεί η διασύνδεση όλων των απαραίτητων block του συνολικού συστήματος. Στις ρυθμίσεις του project επιλέγουμε το φάκελο που περιέχει το IP block - στη περίπτωση μας ονομάζεται **canny**.

Στη συνέχεια δημιουργούμε ένα νέο block design και του δίνουμε το επιθυμητό όνομα. Σε αυτό το σημείο πρέπει να καθοριστεί ποια είναι τα περιφερειακά που χρειάζονται για το σύστημά μας με βάση τις προδιαγραφές που έχουμε ορίσει. Τα βασικά περιφερειακά που απαιτούνται είναι τα εξής:

- ZYNQ7 Processing System
- Canny IP
- AXI Direct Memory Access



Σχήμα 5.5: Βασικά IP Block

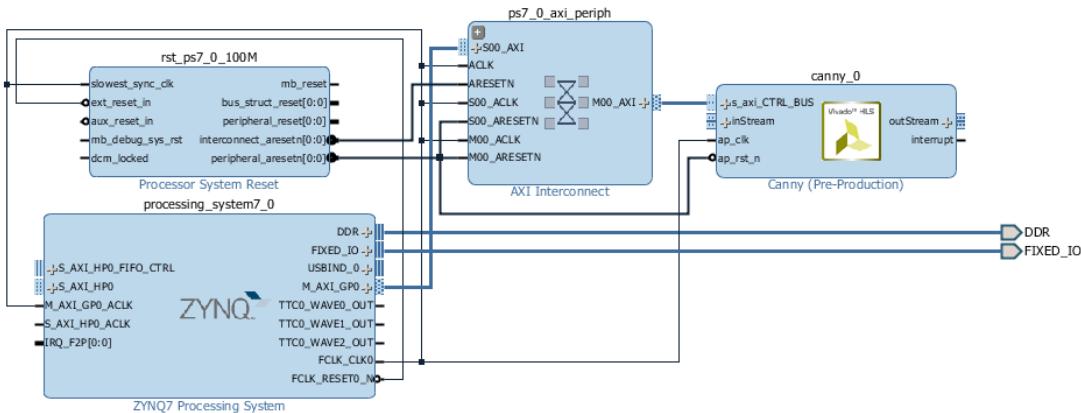
5.3.3 Παραμετροποίηση, Διασύνδεση & Επαλήθευση

Το Vivado έχει τη δυνατότητα να πραγματοποιήσει αυτόματα τις διασυνδέσεις των περιφερειακών και των διάφορων διεπαφών του συστήματος, απλοποιώντας σημαντικά τη διαδικασία. Αρχικά, εισάγουμε το **ZYNQ7 Processing System** και πραγματοποιούμε τις ακόλουθες ρυθμίσεις:

- Εφαρμογή ZC702 Preset
- Ενεργοποίηση των PL-PS Interrupts - IRQ_F2P[15:0]
- Ρύθμιση του ρολογιού του FPGA ίσο με 100 MHz
- Ενεργοποίηση των διεπαφών S AXI HP0 και M AXI GP0

Επιλέγουμε την εντολή **Run Block Automation** η οποία ρυθμίζει το ZYNQ7 PS. Στη συνέχεια εισάγουμε το IP μας και επιλέγουμε την εντολή **Run Connection Automation** η οποία το διασυνδέει με το περιφερειακό μας.

Παρατηρούμε ότι το Vivado εισήγαγε τα Processor System Reset και AXI Interconnect IPs, τα οποία είναι υπεύθυνα για τα reset σήματα και τις διασυνδέσεις AXI αντίστοιχα. Έκανε επίσης όλες τις απαραίτητες συνδέσεις και ρυθμίσεις.



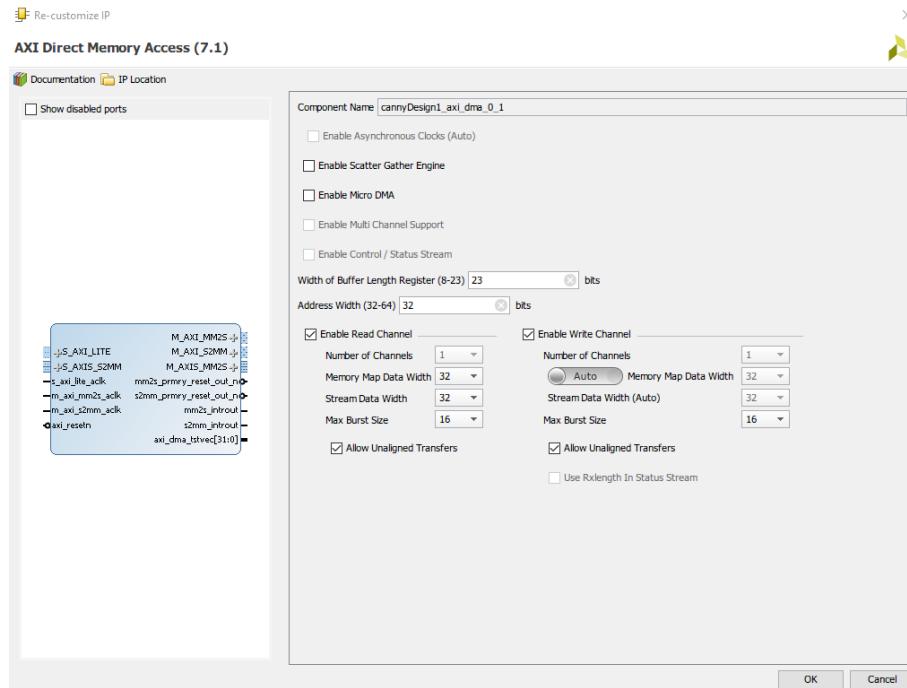
Σχήμα 5.6: Διασύνδεση βασικών IP Blocks

Σειρά έχει η εισαγωγή του AXI DMA περιφερειακού. Με αντίστοιχο τρόπο το εισάγουμε στο design μας και κάνουμε τις απαραίτητες αλλαγές. Ρυθμίζουμε το DMA όπως φαίνεται στην εικόνα, ενεργοποιώντας τον απλό τρόπο λειτουργίας και όχι τον τρόπο Scatter-Gather.

Στον απλό τρόπο λειτουργίας οι μεταφορές δεδομένων θα πρέπει να ελεγχθούν μέσω της εφαρμογής που θα αναπτυχθεί στο SDK. Περιλαμβάνει δύο κανάλια: ένα από το DMA προς τη συσκευή και ένα από τη συσκευή προς το DMA. Η εφαρμογή που θα αναπτυχθεί πρέπει να ορίσει τη διεύθυνση του buffer καθώς και το μέγεθος του ώστε να ξεκινήσει η μεταφορά δεδομένων στο αντίστοιχο κανάλι. Αντίθετα, στο Scatter-Gather τρόπο λειτουργίας, η εφαρμογή ορίζει μία λίστα ανταλλαγών δεδομένων τις οποίες επεξεργάζεται το υλικό χωρίς περαιτέρω παρεμβολή του επεξεργαστή. Κατά τη διάρκεια της μεταφοράς, η εφαρμογή μπορεί να προσθέτει επιπλέον εργασίες στο υλικό.

Στη συνέχεια πραγματοποιούνται όλες οι συνδέσεις μεταξύ όλων των περιφερειακών και του ZYNQ7. Το Vivado όμως παρέλειψε τις συνδέσεις των interrupts, και των εισόδων και εξόδων δεδομένων του AXI4-Stream από και προς το DMA, οι οποίες πρέπει να γίνουν χειροκίνητα. Ενώνουμε επομένως το inStream του Canny με το M_AXIS_MM2S και το outStream με το S_AXIS_S2MM. Καθώς ο αριθμός των απαιτούμενων σημάτων διακοπών είναι τρείς, θα προσθέσουμε ένα concat block το οποίο θα συνενώσει τα interrupts του DMA και του Canny και θα συνδεθεί με το interrupt port του ZYNQ7. Σε περίπτωση που αποφασίσουμε να χρησιμοποιήσουμε τον ελεγκτή

DMA με polling, το παραπάνω βήμα μπορεί να παραλειφθεί και το interrupt του Canny συνδέεται απευθείας στο ZYNQ7. Τέλος, ελέγχουμε αν έχουν οριστεί διευθύνσεις για τα περιφερειακά.



Σχήμα 5.7: Ρύθμιση AXI Direct Memory Access

5.3.4 Σύνθεση, bitstream & εξαγωγή στο SDK

Output products generation

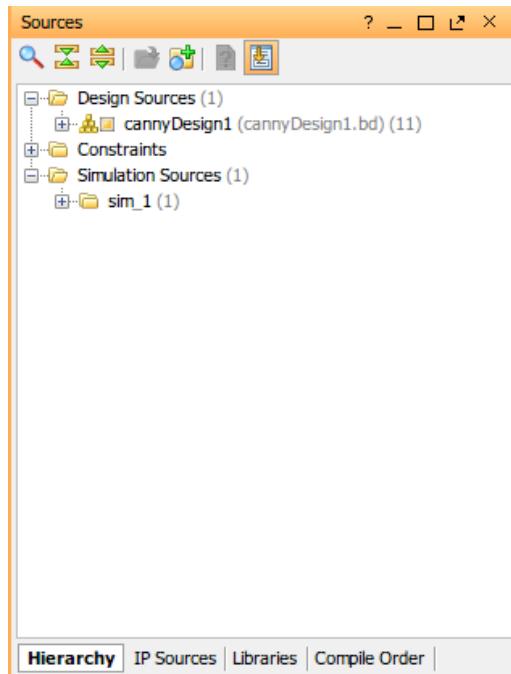
Η εντολή αυτή παραθέτει όλα τους IP πυρήνες του σχεδίου μας και παράγει τα απαραίτητα αποτελέσματα, όπως για παράδειγμα, πρότυπα των instances, HDL, περιορισμούς, αρχεία προσομοίωσης κάτιοντας.

HDL Wrapper Creation

Με την εντολή αυτή ορίζεται ένα HDL wrapper για τον τρέχοντα σχεδιασμό που το ορίζει ως το top-level σχεδιασμό, επιτρέποντας τη σύνθεση, υλοποίηση και την παραγωγή του bitstream.

Αφού βεβαιώσουμε ότι έχουν πραγματοποιηθεί όλες οι απαραίτητες συνδέσεις επιλέγουμε **Validate Design** ώστε να ελεγχθεί το σύστημα μας. Στη συνέχεια από το παράθυρο **Sources** επιλέγουμε το **Generate output products** και στη συνέχεια

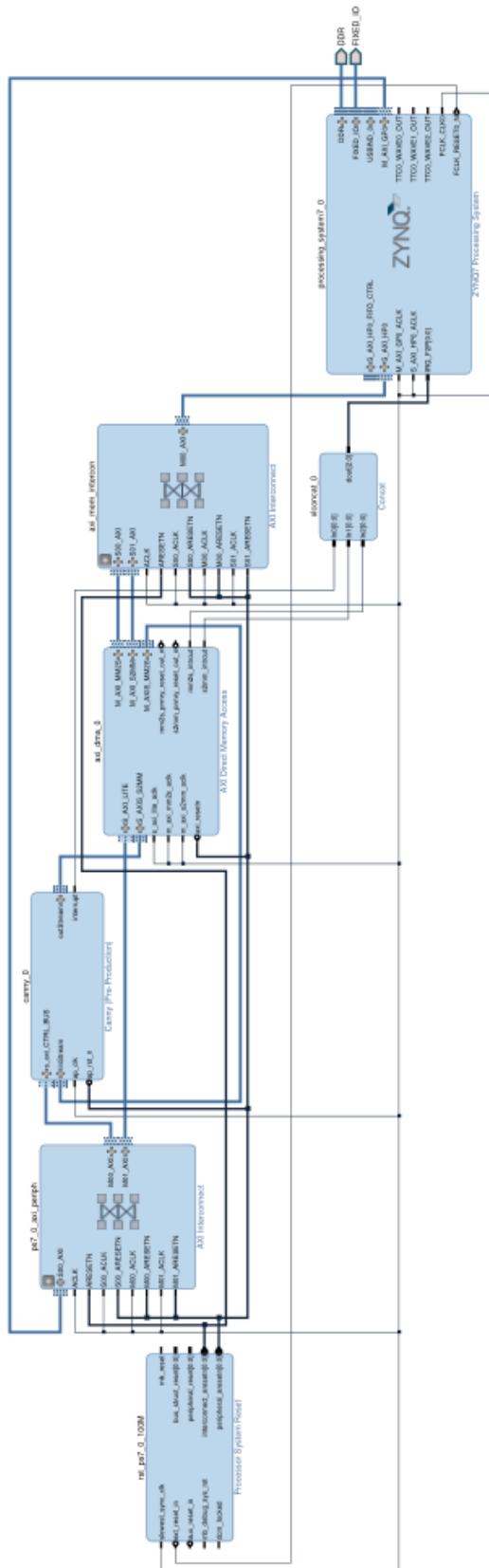
Generate HDL Wrapper. Επόμενο βήμα είναι η εκτέλεση της υλοποίησης.



Σχήμα 5.8: Sources window

Τελευταίο βήμα είναι η σύνθεση, η υλοποίηση και η παραγωγή του αρχείο προγραμματισμού του FPGA. Με την εντολή **Generate Bitstream** πραγματοποιούνται αυτές οι εργασίες. Το Vivado δημιουργεί μία αναφορά για τους απαραίτητους πόρους, τις εκτιμήσεις κατανάλωσης κά. Δημιουργείται επίσης και μία άποψη του FPGA Fabric με όλα τα κατασκευαστικά blocks που χρειάστηκαν για την υλοποίηση του πλήρους συστήματος.

Επιλέγοντας **Export Hardware**, εξάγουμε το υλικό ώστε να το χρησιμοποιήσουμε για τον προγραμματισμό του μέσω του SDK, το οποίο εκτελείται μέσω την εντολής **Launch SDK**.



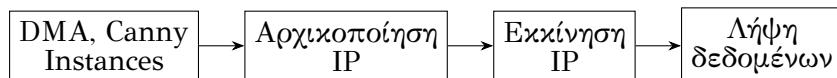
Σχήμα 5.9: Πλήρες σύστημα

5.4 SDK

Σκοπός του Software Development Kit (SDK) της Xilinx είναι ο έλεγχος του επεξεργαστικού συστήματος που αποτελείται από 2 ARM πυρήνες. Το Vivado κατά την εξαγωγή του υλικού δημιουργεί μία πλατφόρμα που περιέχει το σύστημα που σχεδιάστηκε και στην οποία βασιζόμαστε για να αναπτύξουμε τον κώδικα μας.

Το πρώτο βήμα είναι η δημιουργία ενός νέου application project όπου επιλέγουμε το όνομα του, την πλατφόρμα στην οποία θα βασιστεί το project μας. Στο επόμενο βήμα, είναι σκόπιμο να επιλεχθεί η έτοιμη εφαρμογή ‘hello world’ καθώς περιέχει header αρχεία τα οποία είναι απαραίτητα για τη σωστή λειτουργία του προγράμματος μας και δεν περιέχονται έαν είχαμε επιλέξει μία κενή εφαρμογή.

Σκοπός μας είναι η ανάπτυξη μίας εφαρμογής η οποία θα αρχικοποιεί τον DMA ελεγκτή και τις διακοπές ώστε να πραγματοποιείται μεταφορά δεδομένων ανάμεσα στο FPGA και στον επεξεργαστή και θα καλεί τον αλγορίθμο που αναπτύχθηκε στο HLS. Τέλος, θα πραγματοποιείται η έξοδος των δεδομένων μέσω της διεπαφής UART προς τον ΗΥ όπου θα αποθηκεύεται η εικόνα ώστε να έχουμε και μία οπτική απεικόνιση των αποτελεσμάτων. Η βασική ροή της εφαρμογής είναι η εξής:



Σχήμα 5.10: Ροή εφαρμογής ARM

5.4.1 Απαραίτητα αρχεία

Τα βασικά αρχεία που θα χρησιμοποιηθούν και πρέπει να συμπεριληφθούν στον κώδικα μας είναι τα **xcanpy.c** και **xparameters.h**. Το πρώτο αρχείο, αποτελεί τους drivers του IP Block μας και δημιουργήθηκε από το Vivado HLS κατά τη διαδικασία εξαγωγής σε IP. Περιλαμβάνει τις διάφορες εντολές για την εκκίνηση της λειτουργίας, τον έλεγχο της κατάστασής του (idle, ready, done) καθώς και εντολές για το χειρισμό των αιτήσεων διακοπών. Το δεύτερο αρχείο δημιουργήθηκε από το Vivado και περιέχει τις απαιτούμενες τιμές των περιφερειακών όπως για παράδειγμα τις διευθύνσεις των διεπαφών του AXI4, τις διευθύνσεις της DDR μνήμης κάτιοντας.

Δύο εξίσου σημαντικά αρχεία είναι τα **xaxidma.h** και **xscugic.h**. Το πρώτο πε-

ριλαμβάνει τις απαραίτητες συναρτήσεις για την αρχικοποίηση και τον έλεγχο του DMA ελεγκτή ενώ το δεύτερο είναι ο ελεγκτής των αιτήσεων διακοπών. Περιλαμβάνει όλες τις δομές και συναρτήσεις για τον πλήρη έλεγχο των διακοπών του υπολογιστικού συστήματος. Τέλος, τα δεδομένα εισόδου της εικόνας σε grayscale μορφή είναι αποθηκευμένα σε ένα header αρχείο.

```

1 #include <stdio.h>
2 #include "stdlib.h"
3 #include "platform.h"
4 #include "xcanny.h"
5 #include "xparameters.h"
6 #include "xaxidma.h"
7 #include "xscugic.h"
```

5.4.2 Δομή κώδικα

Αρχικά, σε αντίθεση με τη ροή σχεδιασμού στο HLS, η κύρια συνάρτηση είναι η **main()**. Έχει οριστεί επίσης μία συνάρτηση, η **initPeripherals()**.

Δημιουργείται ένα instance για τους drivers του ελεγκτή DMA που θα χρησιμοποιηθεί και στη συνέχεια ορίζεται η δομή των ρυθμίσεων διαμόρφωσης για τους drivers που δημιουργήθηκαν προηγουμένως. Αντίστοιχα, πραγματοποιούνται οι ίδιες λειτουργίες για το canny IP Block. Η λειτουργία των δομών αυτών είναι η μεταφορά πληροφοριών του υλικού από την εφαρμογή στους drivers.

```

1 //DMA Instance
2 XAxiDma      AxiDma;
3 XAxiDma_Config *AxiDma_cfg;
4 //Canny IP Instance
5 XCanny        xCanny;
6 XCanny_Config *xCanny_cfg;
```

main()

Ο σκοπός της κύριας συνάρτησης είναι η εκκίνηση του Canny IP με σκοπό τη λήψη δεδομένων αφότου πραγματοποιηθεί η επεξεργασία τους στο FPGA fabric.

Γίνονται συνεχείς έλεγχοι για τη σωστή λειτουργία της μεταφοράς δεδομένων. Ζητείται επίσης, από το χρήστη να εισάγει μία τιμή κατωφλίωσης. Η εισαγωγή της τιμής σηματοδοτεί επίσης την έναρξη της επεξεργασίας των δεδομένων της εικόνας. Αφότου ολοκληρωθεί η λήψη των επεξεργασμένων δεδομένων, αυτά αποστέλλονται μέσω της διεπαφής UART στον ΗΥ όπου τα λαμβάνουμε με τη βοήθεια του PuTTY και τα αποθηκεύουμε σε αρχείο εικόνας ώστε να αξιολογηθούν.

Για να πραγματοποιηθεί μία απλή μεταφορά δεδομένων τόσο από τον επεξεργαστή προς το περιφερειακό όσο και από το περιφερειακό προς τον επεξεργαστή χρησιμοποιείται η εντολή `XAxiDma_SimpleTransfer(XAxiDma *InstancePtr, UINTPTR BuffAddr, u32 Length, int Direction)`. Το τμήμα κώδικα που εκτελεί τις συγκεκριμένες λειτουργίες είναι το εξής:

```

1 // Receiving output data from Canny IP
2 status = XAxiDma_SimpleTransfer(&AxiDma, (u32)outBuffer, ARRAY_SIZE
3 * sizeof(int), XAXIDMA_DEVICE_TO_DMA);
4 if (status != XST_SUCCESS) {
5     printf("Error: DMA transfer from Vivado HLS block failed\n");
6     return XST_FAILURE;
7 }
8 while (XAxiDma_Busy(&AxiDma, XAXIDMA_DEVICE_TO_DMA)) ;
9 printf("Success: Received results!\r\n");

```

Για τον έλεγχο της κατάστασης του περιφερειακού μας μπορούμε να χρησιμοποιήσουμε τις εντολές που μας παρέχονται από τους drivers που παρήγθησαν μέσω της διαδικασίας εξαγωγής του IP μας. Αυτές είναι οι `isDone`, `isIdle`, `isReady`.

```

1 int isDone, isIdle, isReady;
2
3 isDone = XCanny_IsDone(&xCanny);
4 isIdle = XCanny_IsIdle(&xCanny);
5 isReady = XCanny_IsReady(&xCanny);
6 printf("Canny edge detector: isDone %d, isIdle %d, isReady%d\r\n",
7       isDone, isIdle, isReady);

```

Αφότου πραγματοποιηθεί η λήψη των δεδομένων και πριν τη χρήση τους, πρέπει να πραγματοποιήσουμε το "invalidation" τους ώστε η προσπέλασή τους να

είναι επιτυχής. Οι διαδικασίες αυτές πραγματοποιούνται ως εξής:

```

1 Xil_DCacheFlushRange( (int)output, ARRAY_SIZE*sizeof(int));
2
3 // Perform receive function
4
5 Xil_DCacheInvalidateRange( (u32)outBuffer, ARRAY_SIZE*sizeof(int));
6
7 // Use data

```

initPeripherals()

Βασικός σκοπός της συγκεκριμένης συνάρτησης είναι η αρχικοποίηση των API κλήσεων των drivers του DMA και του Canny και εκκίνηση των περιφερειακών του συστήματος. Παράλληλα, πραγματοποιούνται έλεγχοι ώστε να επιβεβαιωθεί ότι τα περιφερειακά έχουν αρχικοποιηθεί σωστά και είναι λειτουργικά. Στην περίπτωση που υπάρχει κάποιο πρόβλημα, η εκτέλεση του προγράμματος τερματίζεται αυτόματα.

```

1 int status ;
2 // Init Canny Core
3 printf("\nInitialization of Canny IP Block\n");
4 xCanny_cfg = XCanny_LookupConfig(XPAR_CANNY_0_DEVICE_ID);
5 if (xCanny_cfg) {
6     int status = XCanny_CfgInitialize(&xCanny, xCanny_cfg);
7     if (status != XST_SUCCESS)
8         printf("Error initializing Canny IP\n");
9     else
10        printf("\r\nSuccessfully initialized Canny IP\n");
11 }
12
13 // Init axiDMA Core
14 printf("\nInitialization of AXI DMA Core\n");
15 AxiDma_cfg = XAxiDma_LookupConfig(XPAR_AXIDMA_0_DEVICE_ID);
16 if (AxiDma_cfg) {
17     int status = XAxiDma_CfgInitialize(&AxiDma, AxiDma_cfg);
18     if (status != XST_SUCCESS)
19         printf("Error initializing AXI DMA Core\n");
20     else

```

```
21     printf("Success AXIDMA\n");  
22 }
```

Κεφάλαιο 6

Βελτιστοποίηση & Αποτελέσματα

Στο δεύτερο κεφάλαιο έχει γίνει μία αναλυτική παρουσίαση των εργαλείων βελτιστοποίησης που παρέχει το Vivado HLS. Στο παρόν κεφάλαιο, θα αναφερθούμε στην προσπάθεια εφαρμογής τους στο πρόγραμμα που αναπτύχθηκε με στόχο τη μείωση της καθυστέρησης. Θα παρουσιαστούν, επίσης, και τα αποτέλεσματα του συστήματος που αναπτύχθηκε.

6.1 Βελτιστοποίηση

Μην έχοντας θέσει συγκεκριμένες προδιαγραφές για το σύστημα, θα προσπαθήσουμε να βρούμε μία μέση λύση ανάμεσα στη μείωση της συνολικής καθυστέρησης και την αξιοποίηση των πόρων. Στόχος μας είναι, επομένως να εντοπίσουμε πιθανές μεθόδους βελτιστοποίησης που θα μας δώσουν τα καλύτερα δυνατά αποτελέσματα.

Πρώτο βήμα κατά τη διαδικασία βελτιστοποίησης είναι η συγγραφή του κώδικα χωρίς κανένα είδος βελτιστοποίησης ώστε να ληφθεί μία baseline μέτρηση με την οποία θα συγκριθούν οι διάφορες βελτιστοποιημένες υλοποιήσεις.

Στη συνέχεια θα εφαρμοστούν διάφοροι συνδυασμοί βελτιστοποιήσεων και θα επιλεχθεί εκείνος που παρέχει την καλύτερη επιτάχυνση και αξιοποίηση πόρων. Στην περίπτωση που κάποιος συνδυασμός οδηγήσει σε κατανάλωση περισσότερων φυσικών πόρων από τους διαθέσιμους της συσκευής, τότε αυτός απορρίπτεται απευθείας. Προσοχή πρέπει να δοθεί στο ότι οι πόροι του συνολικού συστήματος θα αυξηθούν κατά κάποιο ποσοστό όταν θα σχεδιάσουμε το πλήρες σύστημα στο Vivado λόγω της εισαγωγής των περιφερειακών που θα χρησιμοποιηθούν.

Γιάρχουν δύο τρόποι για την εφαρμογή βελτιστοποιήσεων στο Vivado HLS: απευθείας στον πηγαίο κώδικα η σε ξεχωριστό tcl αρχείο. Με το ξεχωριστό αρχείο tcl, διευκολύνεται σε μεγάλο βαθμό η διαδικασία εύρεσης του καλύτερου συνδυα-

σμού καθώς μπορούμε να δημιουργήσουμε πολλαπλά solutions και να εφαρμόσουμε τις βελτιώσεις που περιέχονται στο αρχείο **directives.tcl** στο καθένα. Και στις δύο περιπτώσεις οι βελτιστοποιήσεις πραγματοποιούνται με τη βοήθεια της ντιρεκτίβας **#pragma HLS**.

Ο συνδυασμός βελτιώσεων που παρουσίασε τα καλύτερα αποτελέσματα περιγράφεται στη συνέχεια. Προέκυψε μετά από ένα πλήθος διαφορετικών συνδυασμών / προσεγγίσεων και προσπαθειών. Στην επόμενη ενότητα θα παρατεθούν τα τελικά αποτελέσματα καθώς και μερικές ενδιάμεσες προσπάθειες - τόσο επιτυχημένες όσο και αποτυχημένες.

6.1.1 Top συνάρτηση

Στην κύρια συνάρτηση πραγματοποιούνται 2 βρόχοι για την είσοδο και έξοδο δεδομένων από και προς τον ελεγκτή DMA αντίστοιχα καθώς και οι κλήσεις των συναρτήσεων.

Λόγω της φύσης του αλγορίθμου μας και του τρόπου προσέγγισής μας τα δεδομένα εισόδου έχουν αποθηκευτεί σε BRAM. Οι βρόχοι επομένως που επεξεργάζονται τα δεδομένα απαιτούν την ανάγνωση από την RAM με αποτέλεσμα να εισάγεται καθυστέρηση λόγω του μικρού αριθμού θυρών της. Για το λόγο αυτό επιλέχθηκε να διαρεθούν οι buffers σε μικρότερους ώστε να επιτευχθεί ταχύτερη ανάγνωση και επεξεργασία δεδομένων καθώς και αυξημένη ρυθμοαπόδοσης. Μετά από πολλούς διαφορετικούς συνδυασμούς καταλήξαμε στην καλύτερη δύνατη διαίρεση. Αυτή επιτυγχάνεται με τη ντιρεκτίβα **array_partition**.

Πιο συγκεκριμένα, ο μονοδιάστατος πίνακας **input** διαιρέθηκε κυκλικά, δημιουργώντας έτσι μικρότερους πίνακες μέσω της τεχνικής interleaving¹. Πρακτικά, κάθε στοιχείο του πίνακα τοποθετείται σε έναν νέο πίνακα μέχρι να πραγματοποιηθεί η πλήρης διαίρεση. Για παράδειγμα, αν επιλεχθεί ένας βαθμός διαίρεσης, *factor* = 3

¹https://en.wikipedia.org/wiki/Interleaved_memory

η διαίρεση θα γίνει ως εξής:

$$\begin{aligned}input[0] &\longrightarrow input1[0] \\input[1] &\longrightarrow input2[0] \\input[2] &\longrightarrow input3[0] \\input[3] &\longrightarrow input4[0]\end{aligned}$$

Ο καλύτερος συμβιβασμός μεταξύ απόδοσης και κατανάλωσης πόρων επιτεύχθηκε με την κυκλική διαίρεση μόνο στον πίνακα `input` βαθμού διαίρεσης ίσου με 2 λόγω των πολλαπλών προσπελάσεων του στη συνάρτηση όπου εκτελείται η πράξη της συνέλιξης.

```
1 static uint8_t input[IMAGE_WIDTH*IMAGE_HEIGHT];  
2 #pragma HLS array_partition variable=input cyclic factor=2
```

6.1.2 Γκαουσιανό φίλτρο

Η συγκεκριμένη συνάρτηση αποτελεί τη πιο "χρονοβόρα" συνάρτηση του αλγορίθμου μας, καθώς περιέχει μαθηματικές πράξεις με floating point μεταβλητές. Αρχικά, με τη χρήση της προηγούμενης βελτίωση που πραγματοποιήθηκε (array cyclic partitioning) μειώθηκε δραστικά ο συνολικός αριθμός κύκλων που απαιτείται για την επεξεργασία των δεδομένων. Εφαρμόστηκε επίσης παραλληλοποίηση στο βρόχο που προσπελαύνει τις στήλες της εικόνας καθώς και στον εσωτερικό βρόχο που διατρέχει τον πυρήνα μετασχηματισμού.

Η βελτιστοποίηση αυτή επέφερε μία δραματική μείωση στους κύκλους ρολογιού καθώς και στο latency της. Πριν τη βελτιστοποίηση, το υλικό που δημιουργήθηκε από το HLS δε θα ήταν εφικτό να επιτύχει στην προσομοίωση λόγω της μεγάλης συχνότητας ρολογιού.

6.1.3 grad

Στη συνάρτηση αυτή εφαρμόσθηκαν εν μέρει αντίστοιχες βελτιώσεις με τις προηγούμενες. Πραγματοποιήθηκε παραλληλοποίηση στους δύο εξωτερικούς βρόχους

της συνάρτησης καθώς και στον εσωτερικό βρόχο της δεύτερης λειτουργίας της που είναι η εύρεση του μέτρου της κλίσης.

```

1 for (i = 0; i < IMAGE_HEIGHT; ++i)
2     #pragma HLS pipeline
3     for (j = 1; j < IMAGE_WIDTH - 1; ++j)
4         find gradX;
5
6 for (j = 0; j < IMAGE_WIDTH; ++j)
7     #pragma HLS pipeline
8     for (i = 1; i < IMAGE_HEIGHT - 1; ++i)
9         find gradY;
10
11 for (i = 0; i < IMAGE_HEIGHT; ++i)
12     for (j = 0; j < IMAGE_WIDTH; ++j, ++t)
13         #pragma HLS pipeline
14         find gradMag;
```

6.1.4 edgeID

Η τεχνική που έδωσε τα καλύτερα αποτελέσματα και σε αυτή την συνάρτηση είναι η παραλληλοποίηση του εσωτερικού βρόχου της συνάρτησης.

6.2 Αποτελέσματα

Για την αξιολόγηση της σωστής λειτουργίας της υλοποίησης μας, συγκρίνονται οπτικά οι εικόνες εξόδου. Επίσης, γίνεται αξιολόγηση των αναγνωρισμένων ακμών για διάφορες τιμές του κατωφλίου. Στην παρούσα ενότητα αναλύονται και τα αποτελέσματα χρονισμού και κατανάλωσης πόρων τόσο στο Vivado HLS όσο και στο Vivado IPI.

6.2.1 Αποτελέσματα υλοποίησης

Αρχικά, παρατίθενται οι μετρήσεις της baseline υλοποίησης. Με βάση αυτές συγχρίνονται τα αποτελέσματα μετα την εφαρμογή των διάφορων τεχνικών βελτιστοποίησης. Περιληπτικά, οι βελτιστοποιήσεις που εφαρμόστηκαν φαίνονται στον ακόλουθο πίνακα.

Συνάρτηση	Τεχνικές
canny	Array Partition, Pipeline
gaussian	Pipeline
grad	Pipeline Rewind
edgeID	Pipeline

Πίνακας 6.1: Τεχνικές βελτιστοποίησης

Στην αρχική υλοποίηση υπάρχει πιθανότητα να μην εκτελεστεί ολόκληρος ο βρόχος λόγω των διαφόρων συνθηκών που υπάρχουν στο σώμα του. Στον πίνακα εμφανίζεται η χειρότερη δυνατή περίπτωση. Στην τελική υλοποίηση όμως, η συνάρτηση θα εκτελέσει όλες τις επαναλήψεις του βρόχου και θα καταναλώσει 45×10^3 κύκλους ρολογιού. Παρατηρούμε μία σημαντική επιτάχυνση της συνάρτησης με σχετικά μικρό αντίκτυπο στους πόρους του FPGA.

Στον ακόλουθο πίνακα εμφανίζονται τα αποτελέσματα της αναφοράς του HLS για την αρχική υλοποίηση και την τελική βελτιστοποιημένη έκδοση. Θα παρατεθούν όμως και τα αποτελέσματα μερικών "ενδιάμεσων" υλοποιήσεων. Η υλοποίηση **impl_1** περιλαμβάνει μόνο **pipeline** σε όλες τις συναρτήσεις εκτός της **grad()**, η **impl_2** περιλαμβάνει **pipeline** και **array partition** και η **impl_3** περιλαμβάνει **pipeline** σε όλους τους βρόχους του αλγορίθμου και **array partition**. Παρατηρούμε ότι η τρίτη υλοποίηση παρουσιάζει τα καλύτερα αποτελέσματα επιτυγχάνοντας τη μεγαλύτερη επιτάχυνση.

Υλοποίηση	Αξιοποίηση Πόρων				Κύκλοι Ρολογιού	Ρολόι	Επιτάχυνση	Αντίκτυπο
	BRAM	DSP	FF	LUT				
Baseline	12%	4%	1%	5%	3.1×10^6	8.10 ns	-	-
impl_1	23%	23%	14%	41%	1.4×10^5	9.09 ns	x21	x7.2
impl_2	23%	23%	14%	41%	1.41×10^5	9.09 ns	x21	x7.2
impl_3	23%	24%	17%	52%	9.4×10^4	9.09 ns	x32	x9.4

Πίνακας 6.2: Σύγκριση υλοποιήσεων

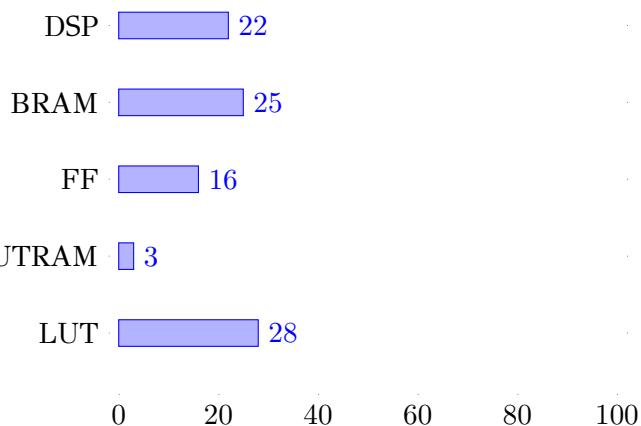
Με βάση τα δεδομένα του προηγούμενου πίνακα γίνεται σαφές ότι η καλύτερη υλοποίηση είναι η τελευταία. Η τελική υλοποίηση προσφέρει ικανοποιητική συνολική επιτάχυνση και χαμηλή συχνότητα ρολογιού με μία ποσοστιαία αύξηση 900% στην κατανάλωση των LUT. Έχει επιτευχθεί, επομένως ένας πολύ καλός συμβιβασμός μεταξύ απόδοσης και κατανάλωσης πόρων. Η συνολική κατανάλωση θα αυξηθεί βέβαια κατά τη διάρκεια του σχεδιασμού του πλήρους συστήματος στο Vivado IPI.

Μετά την επιτυχή προσομοίωση και εξαγωγή του υλικού στο HLS, σχεδιάζεται το πλήρες σύστημα στο Vivado όπως έχει περιγραφεί στο προηγούμενο κεφάλαιο. Το Vivado παράγει αναλυτική αναφορά σχετικά με τη τελική αξιοποίηση των διαθέσιμων πόρων.

Έχοντας ορίσει τη συχνότητα λειτουργίας του FPGA ίση με 100 MHz παρατηρούμε ότι για να ολοκληρωθεί η επεξεργασία της εικόνας εισόδου απαιτούνται 93278 κύκλοι ρολογιού. Επομένως, ο συνολικός χρόνος που απαιτείται για την επεξεργασία ενός frame είναι: $93278 \times 10^{-8} = 0,00093278 s = 0.93 ms$.

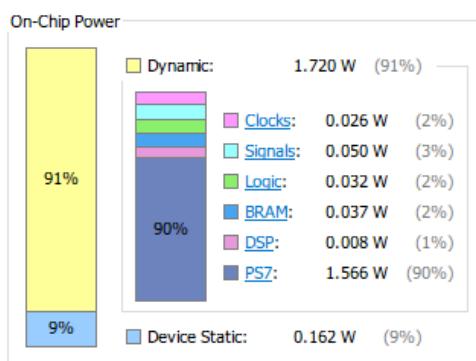
Παρατηρείται επίσης, μία μείωση στο συνολικό αριθμό LUT που χρησιμοποιήθηκαν στο τελικό σχέδιο. Αυτή η διαφορά οφείλεται στο γεγονός ότι το HLS κάνει μία εκτίμηση της αξιοποίησης των πόρων καθώς και σε διάφορες βελτιστοποιήσεις που πραγματοποιούει το Vivado κατά τη διαδικασία της σύνθεσης.

Πόροι	Αξιοποίηση	Διαθέσιμοι
DSP	48	220
BRAM	35	140
FF	16988	106400
LUTRAM	511	17400
LUT	14986	53200



Σχήμα 6.1: Ποσοστό αξιοποίησης συνολικών πόρων

Στο ακόλουθο σχήμα φαίνεται η εκτίμηση της κατανάλωσης του συστήματος.

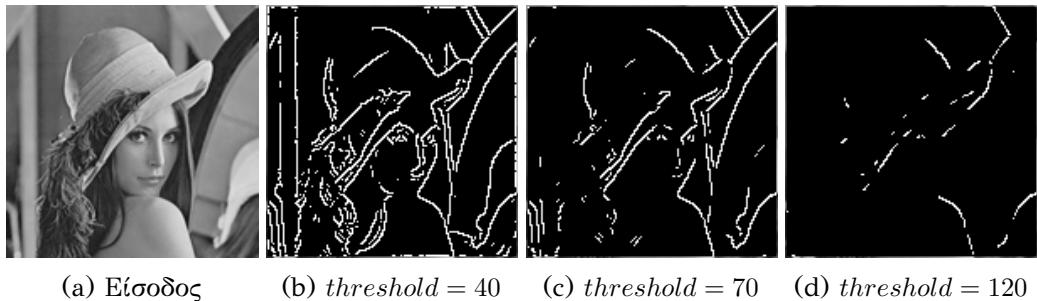


Σχήμα 6.2: Εκτίμηση κατανάλωσης

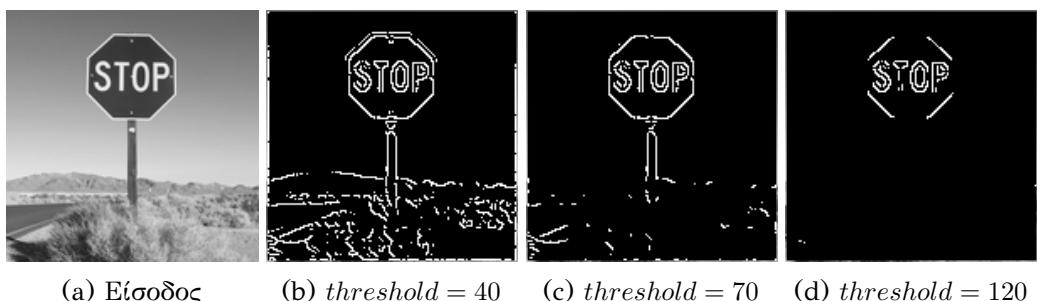
6.2.2 Αποτελέσματα επεξεργασίας

Τελευταίο βήμα της εργασίας είναι η αξιολόγηση της σωστής λειτουργίας του αλγορίθμου επεξεργασίας ακμών που υλοποιήθηκε. Για το λόγο αυτό, επιλέχθηκαν

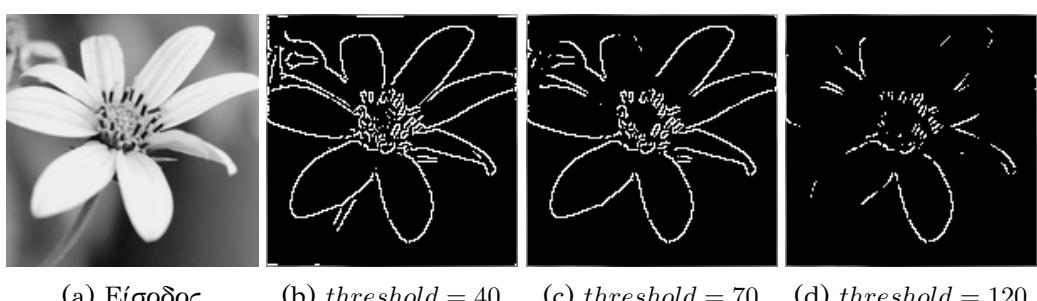
τρεις διαφορετικές εικόνες που θα υποστούν επεξεργασία. Παράλληλα, θα επιλεχθούν τρεις διαφορετικές τιμές κατωφλίωσης, ώστε να επιλεχθεί εκείνη που παράγει τα βέλτιστα αποτελέσματα.



Σχήμα 6.3: Εικόνα #1



Σχήμα 6.4: Εικόνα #2



Σχήμα 6.5: Εικόνα #3

Παρατηρούμε ότι ο αλγόριθμος μας παράγει σωστά αποτελέσματα με καλά αναγνωρισμένες ακμές. Επίσης, είναι εμφανές ότι η τιμή του κατωφλίου που δίνει τα καλύτερα αποτελέσματα διαφέρει από εικόνα σε εικόνα. Επομένως, η σωστή τιμή επιλέγεται με τη μέθοδο "trial & error". Η τιμή $threshold = 40$ δίνει όμως τα καλύτερα αποτελέσματα σε σταθερή βάση.

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα

Η όλη διαδικασία για την ανάπτυξη και υλοποίηση ενός συστήματος υπολογιστικής όρασης σε FPGA με τη μέθοδο HLS είναι αρκετά πολύπλοκη αλλά αποδοτική για πολλές περιπτώσεις εφαρμογών. Εμφανίζει σημαντικά πλεονεκτήματα αλλά και μειονεκτήματα που πρέπει να συνεκτιμήθουν πριν από την έναρξη οποιουδήποτε project.

Η ανάπτυξη του αλγορίθμου για την εφαρμογή μας γίνεται αρχικά στον VHDL σε κάποια γλώσσα προγραμματισμού υψηλού επιπέδου και έτσι δεν απαιτείται προσπάθεια εκμάθησης μία νέας γλώσσας (VHDL, Verilog). Γνώσεις κάποιας γλώσσας περιγραφής υλικού μπορούν να διευκολύνουν σε ορισμένες περιπτώσεις την υλοποίηση της εφαρμογής μας. Η μεταφορά, επίσης, του αλγορίθμου μας στο Vivado HLS είναι απλή διαδικασία. Επίσης, η αποσφαλμάτωση, η αλλαγή κάποιων βημάτων και η βελτιστοποίησή του είναι σχετικά απλά βήματα και μπορούν να πραγματοποιηθούν σε μικρό χρονικό διάστημα. Ως παράδειγμα, αναφέρουμε ότι μετά την ανάπτυξη του κώδικα μας, επετεύχθη χωρίς μεγάλο κόπο η μείωση των απαιτούμενων κύκλων από 3.1×10^6 σε 9.4×10^4 .

Πρέπει όμως σε αυτό το σημείο να τονιστεί ότι δεν είναι όλοι οι κώδικες συνθέσιμοι και πρέπει να ληφθεί ειδική μέριμνα κατά τη συγγραφή ώστε να τηρηθούν οι περιορισμοί του HLS. Επίσης, σημαντικό ρόλο στην απόδοση του αλγορίθμου παίζει και ο τρόπος αντιμετώπισης του προβλήματος. Αρκετές φορές, ο συμβατικός τρόπος γραφής κώδικα δεν ”ταιριάζει” στο FGPA με αποτέλεσμα τη μειωμένη απόδοση του συστήματος.

Σχετικά, με τη διαδικασία ελέγχου της ορθής λειτουργίας του αλγορίθμου, αυτή πραγματοποιείται αρκετά γρήγορα και εύκολα καθώς το testbench αναπτύσσεται σε

γλώσσα προγραμματισμού υψηλού επιπέδου. Από τους ελέγχους που πραγματοποιήθηκαν διαπιστώθηκε η πλήρης ταύτιση των εικόνων που επεξεργάστηκαν στο FPGA με αυτές που παράχθηκαν στον ΗΥ.

Είναι γεγονός ότι η αρχική ανάπτυξη του πλήρους συστήματος απαιτεί άριστη γνώση των επιμέρους τμημάτων του. Συνεπώς, είναι αναγκαία η μελέτη μεγάλου όγκου πληροφοριών από τα εγχειρίδια τόσο του Vivado HLS όσο και του χρησιμοποιούμενου εξοπλισμού της Xilinx. Στη συνέχεια βέβαια τα πράγματα είναι ευκολότερα όταν έχει αποκτηθεί η σχετική εμπειρία.

Παρά το γεγονός όμως ότι η διαδικασία ανάπτυξης του αλγορίθμου και η μεταφορά του στο HLS είναι σημαντικά συντομότερη από την ανάπτυξη του σε HDL, πολλές φορές εμφανίζει σφάλματα. Όσο περισσότερα βήματα γίνονται τόσο μεγαλύτερη είναι και η πιθανότητα εμφάνισης σφαλμάτων.

Καταληκτικά, συμπεραίνουμε ότι είναι δυνατή η υλοποίηση ενός συστήματος CV σε ένα εύλογο χρονικό διάστημα, επιτυγχάνοντας ικανοποιητικά επίπεδα απόδοσης. Το σύστημα που αναπτύχθηκε είναι εύκολο να τροποποιηθεί μέσω απλών διαδικασιών σε σύντομο χρόνο. Αυτή η μέθοδος ανάπτυξης υλικού παρά το γεγονός ότι ειναι σχετικά καινούρια μπορεί να χρησιμοποιηθεί στη βιομηχανία για ανάπτυξη πρωτότυπων και συστημάτων που δεν απαιτούν ακρίβεια σε επίπεδο bit. Απαιτείται όμως σημαντική εμπειρία στη συγγραφή κώδικα με κατάλληλο τρόπο ώστε να αναπτυχθεί η επιθυμητή αρχιτεκτονική υλικού. Παρ' όλα αυτά, είναι φανερό ότι το HLS είναι ένα συνεχώς βελτιούμενο εργαλείο το οποίο αξίζει να βρίσκεται στο πεδίο ενδιαφέροντος των σχεδιαστών υλικού.

7.2 Μελλοντικές εργασίες & βελτιώσεις

Κλείνοντας, θα πρέπει να αναφέρουμε ότι τα αποτελέσματα της παρούσας εργασίας μπορούν να συμπληρωθούν μελλοντικά με περαιτέρω βελτιώσεις ή προσθήκη νέων δυνατοτήτων.

- Τροποποίηση του αλγορίθμου για την υποστήριξη εικόνων μεγαλύτερης ανάλυσης καθώς και περαιτέρω αύξηση της απόδοσης του. Αυτό μπορεί να επιτευχθεί με την υλοποίηση παραθύρων για την επεξεργασία των νέων δεδομένων ώστε

να επιτευχθεί μείωση στο συνολικό χρόνο καθυστέρησης παραγωγής νέου δείγματος και εξοικονόμηση μνήμης στο FPGA. Μία ακόμη τεχνική είναι η διαίρεση της εικόνας σε N μικρότερες και η διαδοχική επεξεργασία τους.

2. Αύξηση της λειτουργικότητάς του αλγορίθμου μας με τη σύνδεση περιφερειακών για τη λήψη και αποστολή δεδομένων, όπως για παράδειγμα μέσω κάμερας ή διεπαφής HDMI.
3. Αξιοποίηση των βιβλιοθηκών που παρέχονται από τη Xilinx όπως για παράδειγμα, η βιβλιοθήκη openCV¹ η οποία παρέχει ολόκληρες λειτουργίας υπολογιστικής όρασης με τη μορφή απλών συναρτήσεων.

¹<https://opencv.org/>

Βιβλιογραφία

- [1] **Understanding FPGA Architecture.** https://www.xilinx.com/html_docs/xilinx2017_1/sdaccel_doc/topics/devices/con-fpga-architecture.html. (Accessed on 10/17/2017).
- [2] **7 Series FPGAs Configurable Logic Block User Guide (UG474).** https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. (Accessed on 10/17/2017).
- [3] **LUT.** https://www.xilinx.com/html_docs/xilinx2017_1/sdaccel_doc/topics/devices/con-fpga-arch-lut.html. (Accessed on 10/17/2017).
- [4] **Flip Flop.** https://www.xilinx.com/html_docs/xilinx2017_1/sdaccel_doc/topics/devices/con-fpga-arch-flipflop.html. (Accessed on 10/17/2017).
- [5] **DSP48 Block.** https://www.xilinx.com/html_docs/xilinx2017_1/sdaccel_doc/topics/devices/con-fpga-arch-dsp48.html. (Accessed on 10/17/2017).
- [6] **BRAM and Other Memories.** https://www.xilinx.com/html_docs/xilinx2017_1/sdaccel_doc/topics/devices/con-fpga-arch-bram.html. (Accessed on 10/17/2017).
- [7] **Zynq-7000 All Programmable SoC Data Sheet:Overview.** DS190. Rev. 1.11. Xilinx. Ιούν. 2017.
- [8] Louise H. Crockett κ.ά. **The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.** UK: Strathclyde Academic Media, 2014.
- [9] **Zynq-7000 AP SoC Family Product Tables and Product Selection Guide.** <https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf>. (Accessed on 10/17/2017).
- [10] **Zynq-7000 All Programmable SoC ZC702 Evaluation Kit -Getting Started Guide.** UG926. Rev. 1.21. Xilinx. Σεπτ. 2012.
- [11] **ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide.** UG850. Rev. 1.5. Xilinx. Σεπτ. 2015.

- [12] **Introduction to FPGA Design with Vivado High-Level Synthesis.** UG998. Rev. 1.0. Xilinx. Ιούλ. 2013.
- [13] Wim Meeus κ.ά. «An overview of today's high-level synthesis tools». Στο: *Design Automation for Embedded Systems* 16.3 (Σεπτ. 2012), σσ. 31–51. ISSN: 1572-8080. doi: 10.1007/s10617-012-9096-8. URL: <https://doi.org/10.1007/s10617-012-9096-8>.
- [14] **HLS Pragmas.** https://www.xilinx.com/html_docs/xilinx2017_2/sdaccel_doc/topics/pragmas/concept-Intro_to_HLS_pragmas.html. (Accessed on 10/13/2017).
- [15] **Vivado Design Suite User Guide - High Level Synthesis.** UG902. Rev. 2014.1. Xilinx. Μάι. 2014.
- [16] Wikipedia. **Computer vision — Wikipedia, The Free Encyclopedia.** <http://en.wikipedia.org/w/index.php?title=Computer%20vision&oldid=805574166>. [Online; accessed 17-October-2017]. 2017.
- [17] Wikipedia. **Glossary of machine vision — Wikipedia, The Free Encyclopedia.** <http://en.wikipedia.org/w/index.php?title=Glossary%20of%20machine%20vision&oldid=787508851>. [Online; accessed 17-October-2017]. 2017.
- [18] **EasyBMP Cross-Platform Windows BMP Library: Home.** <http://easybmp.sourceforge.net/>. (Accessed on 10/18/2017).
- [19] **AXI Reference Guide.** UG761. Rev. 13.1. Xilinx. Μαρ. 2011.
- [20] **Vivado Design Suite User Guide Designing IP Subsystems Using IP Integrator.** UG994. Rev. 2015.1. Xilinx. Ιαν. 2015.
- [21] **Vivado Design Suite Tutorial - High-Level Synthesis.** UG871. Rev. 2015.4. Xilinx. Νοέ. 2015.
- [22] **Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries.** XAPP1167. Rev. 3. Xilinx. Ιούν. 2015.
- [23] **A Zynq Accelerator for Floating Point Matrix Multiplication Designed with Vivado HLS.** XAPP1170. Rev. 2. Xilinx. Ιαν. 2016.
- [24] **Vivado Design Suite User Guide - Design Flows Overview.** UG892. Rev. 2016.1. Xilinx. Απρ. 2016.
- [25] **Zynq-7000 All Programmable SoC Technical Reference Manual.** UG585. Rev. 1.11. Xilinx. Σεπτ. 2016.

- [26] **Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool.** XAPP890. Rev. 1.0. Xilinx. Σεπτ. 2012.
- [27] <http://www.songho.ca/dsp/index.html>. (Accessed on 10/13/2017).
- [28] Rafael C. Gonzalez και Richard E. Woods. **Digital Image Processing (3rd Edition)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN: 013168728X.
- [29] John Canny. «A Computational Approach to Edge Detection». Στο: *IEEE Transactions on pattern analysis and machine intelligence* (Νοέ. 1986).
- [30] Philippe Coussy κ.ά. «An Introduction to High-Level Synthesis». Στο: *IEEE Design & Test of Computers* (Αύγ. 2009).
- [31] Prem Kalra. **Canny Edge Detector**. Μαρ. 2009.
- [32] **High-level synthesis — Wikipedia, The Free Encyclopedia**. <http://en.wikipedia.org/w/index.php?title=High-level%20synthesis&oldid=800613213>. [Online; accessed 13-October-2017]. 2017.
- [33] **Gaussian function — Wikipedia, The Free Encyclopedia**. <http://en.wikipedia.org/w/index.php?title=Gaussian%20function&oldid=803511980>. [Online; accessed 13-October-2017]. 2017.
- [34] **Canny edge detector — Wikipedia, The Free Encyclopedia**. <http://en.wikipedia.org/w/index.php?title=Canny%20edge%20detector&oldid=781894186>. [Online; accessed 13-October-2017]. 2017.

Πανεπιστήμιο Πατρών, Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών
Χρήστος-Αλέξανδρος Νικολακάκης
© Οκτώβριος 2017 – Με την επιφύλαξη παντός δικαιώματος.
