



Πανεπιστήμιο Δυτικής Μακεδονίας  
Πολυτεχνική Σχολή  
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Υλοποίηση και Επιτάχυνση Συνελικτικού  
Νευρωνικού Δικτύου σε Ετερογενές Σύστημα για  
Αναγνώριση Χειρόγραφων Αριθμών

Διπλωματική εργασία  
του  
Θεόδωρου Ταλουμζή

Επιβλέπων: Δρ. Μηνάς Δασυγένης  
Εργαστήριο Ρομποτικής, Ενσωματωμένων και  
Ολοκληρωμένων Συστημάτων

Κοζάνη/Οκτώμβριος/2025





University of Western Macedonia  
Faculty of Engineering  
Department of Electrical and Computer Engineering

# Implementation of a Convolutional Neural Network on a Heterogeneous System for Accelerated Classification of Numerical Handwritten Characters

Diploma Thesis  
of  
Theodoros Taloumtzis

**Supervisor:** Dr. Minas Dasygenis  
Laboratory of Robotics, Embedded and Integrated Systems

**Kozani/October/2025**



## **ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ**

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο “Υλοποίηση και Επιτάχυνση Συνελικτικού Νευρωνικού Δικτύου σε Ετερογενές Σύστημα για Αναγνώριση Χειρόγραφων Αριθμών” καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Δασυγένη Μηνά, Αναπληρωτή Καθηγητή αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

**Copyright © Ταλουμτζής Θεόδωρος, Δρ. Μηνάς Δασυγένης, 2025, Κοζάνη**

Υπογραφή Φοιτητή: Θεόδωρος Ταλουμτζής



## Περίληψη

Η ραγδαία εξάπλωση της τεχνητής νοημοσύνης και των εφαρμογών μηχανικής μάθησης έχει αυξήσει σημαντικά τη ζήτηση για αποδοτική εκτέλεση αλγορίθμων σε συστήματα *edge computing*. Σε αυτά τα περιβάλλοντα, οι απαιτήσεις για χαμηλή καθυστέρηση, χαμηλή κατανάλωση ενέργειας και λειτουργία σε πραγματικό χρόνο καθιστούν τις παραδοσιακές πλατφόρμες CPU και GPU συχνά μη βιώσιμες. Σε αυτό το πλαίσιο, οι προγραμματιζόμενες συστοιχίες πυλών (Field-Programmable Gate Arrays - FPGAs) αναδεικνύονται ως ιδανική λύση, προσφέροντας παραλληλισμό σε επίπεδο υλικού και δυνατότητα επαναπρογραμματισμού με υψηλή ενεργειακή αποδοτικότητα.

Η παρούσα εργασία εστιάζει στην υλοποίηση συνελικτικών νευρωνικών δικτύων (Convolutional Neural Networks, CNNs) σε FPGA μέσω τεχνικών Υψηλού Επιπέδου Σύνθεσης (High-Level Synthesis, HLS). Αναπτύχθηκε μία ολοκληρωμένη ροή ανάπτυξης που επιτρέπει τη μεταφορά εκπαιδευμένων μοντέλων από περιβάλλοντα λογισμικού σε υλοποιήσιμη περιγραφή υλικού, με στόχο τη βελτιστοποίηση της καθυστέρησης, της χρήσης πόρων και της ενεργειακής αποδοτικότητας.

Η μεθοδολογία βασίστηκε σε σύγχρονα εργαλεία ανοιχτού και εμπορικού λογισμικού, όπως **Keras/QKeras** για την εκπαίδευση και κβάντιση, **hls4ml** για τη μετατροπή σε C/C++ περιγραφή, και **Vitis HLS, Vivado** για τη σύνθεση και την υλοποίηση. Η τελική ανάπτυξη και αξιολόγηση πραγματοποιήθηκε στην πλατφόρμα **AMD Kria KR260**, με χρήση του **PYNQ framework** για τη διαχείριση των επιταχυντών και τη συλλογή μετρήσεων ισχύος και καθυστέρησης.

Τα αποτελέσματα απέδειξαν ότι η προτεινόμενη ροή παράγει λειτουργικούς επιταχυντές CNN με ακρίβεια 98,5%, καθυστέρηση μικρότερη από 1,5 ms και μέση κατανάλωση ισχύος περίπου 3,5 W. Η ενεργειακή αποδοτικότητα φτάνει τα  $2,1 \times 10^4$  1/J, περίπου τριπλάσια σε σχέση με την GPU αναφοράς, ενώ η χρήση DSPs περιορίστηκε στο 10% της διαθέσιμης χωρητικότητας. Η CPU εμφάνισε πολύ χαμηλή καθυστέρηση για μικρά σύνολα δεδομένων (0,019 ms) αλλά χαμηλότερη αποδοτικότητα σε μεγάλα φορτία, λόγω περιορισμένου παραλληλισμού.

Συνολικά, η εργασία αποδεικνύει ότι τα FPGAs μπορούν να υποστηρίξουν εκτέλεση νευρωνικών δικτύων σε πραγματικό χρόνο με υψηλή ενεργειακή αποδοτικότητα, καθιστώντας τα κατάλληλη επιλογή για εφαρμογές τεχνητής νοημοσύνης στα άκρα. Η προτεινόμενη ροή ανάπτυξης συνδυάζει ευελιξία, αυτοματοποίηση και προβλεψιμότητα, αποτελώντας μια σταθερή βάση για τη μελλοντική υλοποίηση πιο σύνθετων δικτύων και ετερογενών επιταχυντών.

---

**Λέξεις κλειδιά:** Τεχνητή Νοημοσύνη, FPGA, CNN, Υψηλού Επιπέδου Σύνθεση, hls4ml, Vitis HLS, Kria KR260, Κβάντιση, Κλάδεμα, Ενεργειακή Αποδοτικότητα.

## Abstract

The rapid expansion of Artificial Intelligence (AI) and Machine Learning (ML) applications has significantly increased the demand for efficient algorithm execution in *edge computing* systems. In such environments, the requirements for low latency, low power consumption, and real-time operation often render traditional CPU and GPU platforms impractical. In this context, **Field-Programmable Gate Arrays (FPGAs)** emerge as an ideal solution, offering hardware-level parallelism and reconfigurability with high energy efficiency.

This work focuses on the implementation of **Convolutional Neural Networks (CNNs)** on FPGA devices using **High-Level Synthesis (HLS)** techniques. An end-to-end development flow was designed, enabling the deployment of trained neural models from software environments to synthesizable hardware descriptions, with the goal of optimizing latency, resource utilization, and energy efficiency.

The proposed methodology leverages modern open-source and commercial tools, including **Keras/QKeras** for model training and quantization, **hls4ml** for conversion into C/C++ descriptions, and **Vitis HLS/Vivado** for synthesis and implementation. Final deployment and evaluation were performed on the **AMD Kria KR260** platform, using the **PYNQ framework** to manage accelerator execution and collect performance and power measurements.

Experimental results demonstrated that the proposed flow produces functional CNN accelerators achieving an accuracy of 98,5%, latency below 1,5 ms, and an average board power consumption of approximately 3,5 W. The achieved energy efficiency reached  $2,1 \times 10^4$  1/J, about three times higher than the GPU reference, while DSP utilization remained at only 10% of the available fabric. The CPU achieved very low latency for small datasets (0,019 ms), but lower overall efficiency for larger workloads due to limited parallelism.

Overall, the work demonstrates that FPGAs can support real-time neural network inference with high energy efficiency, making them a compelling choice for *edge AI* applications. The proposed development flow combines flexibility, automation, and predictability, providing a robust foundation for future implementations of more complex networks and heterogeneous accelerators.

**Keywords:** Artificial Intelligence, FPGA, CNN, High-Level Synthesis, hls4ml, Vitis HLS, Kria KR260, Quantization, Pruning, Energy Efficiency.

## Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του προπτυχιακού προγράμματος σπουδών της Πολυτεχνικής Σχολής του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας. Στο σημείο αυτό θα ήθελα να εκφράσω τις ευχαριστίες μου προς τα πρόσωπα που συνέβαλαν καθοριστικά στην ολοκλήρωση της παρούσας εργασίας.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της διπλωματικής μου, Δρ. Μηνά Δασυγένη, για την εμπιστοσύνη που μου έδειξε στην ανάληψη του θέματος, για τη συνεχή καθοδήγηση, την υπομονή και τη διαρκή του υποστήριξη καθ' όλη τη διάρκεια της εκπόνησης. Η συμβολή του υπήρξε καθοριστική τόσο στην επιστημονική προσέγγιση του έργου όσο και στην ολοκλήρωσή του.

Επίσης, θα ήθελα να ευχαριστήσω το Εργαστήριο Ρομποτικής και Ενσωματωμένων Συστημάτων για τη διάθεση του απαραίτητου εξοπλισμού και ιδιαίτερα της πλακέτας FPGA Kria KR260, που χρησιμοποιήθηκε για την πειραματική υλοποίηση και τις μετρήσεις της εργασίας.

Ιδιαίτερες ευχαριστίες οφείλω στους καθηγητές του τμήματος, των οποίων η διδασκαλία και οι γνώσεις συνέβαλαν ουσιαστικά στη διαμόρφωση του επιστημονικού μου υπόβαθρου και στην ανάπτυξη δεξιοτήτων που αποδείχθηκαν κρίσιμες για την υλοποίηση της παρούσας εργασίας.

Τέλος, θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη προς την οικογένειά μου για την αμέριστη στήριξη, την κατανόηση και την πίστη τους σε εμένα, καθώς και στους φίλους και συμφοιτητές μου για τη συντροφικότητα, τη βοήθεια και την ηθική υποστήριξη καθ' όλη τη διάρκεια των σπουδών και της συγγραφής της παρούσας διπλωματικής εργασίας.

# Περιεχόμενα

|          |                                                       |           |
|----------|-------------------------------------------------------|-----------|
| <b>1</b> | <b>Εισαγωγή</b>                                       | <b>9</b>  |
| 1.1      | Αντικείμενο της διπλωματικής                          | 9         |
| 1.2      | Παρόμοια έρευνα                                       | 11        |
| 1.3      | Στόχος της διπλωματικής                               | 12        |
| 1.4      | Οργάνωση της εργασίας                                 | 13        |
| <b>2</b> | <b>Θεωρητικό Υπόβαθρο και Έννοιες</b>                 | <b>15</b> |
| 2.1      | Συνελικτικά Νευρωνικά Δίκτυα                          | 15        |
| 2.1.1    | Εισαγωγή στα Νευρωνικά Δίκτυα                         | 15        |
| 2.1.2    | Εισαγωγή στα CNNs                                     | 17        |
| 2.1.3    | Πλαίσια Μηχανικής Μάθησης                             | 19        |
| 2.1.4    | Σημαντικές τοπολογίες CNN για ταξινόμηση              | 20        |
| 2.2      | Αρχιτεκτονική και Λειτουργία των FPGA                 | 21        |
| 2.2.1    | Εισαγωγή στα FPGA                                     | 21        |
| 2.2.2    | Εσωτερική αρχιτεκτονική                               | 22        |
| 2.2.3    | FPGA έναντι ASIC                                      | 23        |
| 2.2.4    | FPGA έναντι CPU/GPU για επιτάχυνση νευρωνικών δικτύων | 23        |
| 2.2.5    | Ροές εργαλείων για FPGA                               | 24        |
| 2.3      | Λειτουργικά Συστήματα και Εργαλεία                    | 25        |
| 2.3.1    | Εργαλεία που χρησιμοποιήθηκαν στην εργασία            | 25        |
| 2.3.2    | Γλώσσες Περιγραφής Υλικού                             | 30        |
| 2.4      | Μετρήσεις και Ορισμοί                                 | 31        |
| 2.4.1    | Μετρικές Ταξινόμησης                                  | 31        |
| 2.4.2    | Μετρικές απόδοσης                                     | 32        |
| 2.4.3    | Ενεργειακές μετρήσεις                                 | 33        |
| 2.4.4    | Μετρικές χρήσης και επαναχρησιμοποίησης πόρων         | 34        |
| 2.5      | Αρχιτεκτονική SoC FPGA και Πλατφόρμα AMD Kria KR260   | 35        |
| 2.5.1    | Πλατφόρμα AMD Kria KR260                              | 35        |
| 2.5.2    | Processing System (PS) και Programmable Logic (PL)    | 35        |
| 2.5.3    | Πρωτόκολλα Διασύνδεσης: AMBA AXI4 και AXI4-Stream     | 37        |

|          |                                                      |           |
|----------|------------------------------------------------------|-----------|
| 2.5.4    | Ιεραρχία Μνήμης και Μετακίνηση Δεδομένων (DMA)       | 37        |
| 2.5.5    | Βασικά Χαρακτηριστικά του K26 SoM                    | 38        |
| 2.5.6    | Καταλληλότητα της KR260 για CNN Accelerators         | 38        |
| <b>3</b> | <b>Μεθοδολογία</b>                                   | <b>40</b> |
| 3.1      | Περίληψη της Προσέγγισης                             | 40        |
| 3.1.1    | Στρατηγική                                           | 41        |
| 3.1.2    | Γενική Ροή Εργασίας                                  | 41        |
| 3.2      | Σχεδιασμός του Μοντέλου                              | 42        |
| 3.2.1    | Σύνολο Δεδομένων                                     | 42        |
| 3.2.2    | Αρχιτεκτονική                                        | 43        |
| 3.2.3    | Εκπαίδευση                                           | 44        |
| 3.2.4    | Βελτιστοποιήσεις                                     | 45        |
| 3.2.5    | Κβαντοποίηση                                         | 45        |
| 3.2.6    | Συνολική Εκτίμηση                                    | 46        |
| 3.2.7    | Κατασκευή Βασικού Μοντέλου (Baseline)                | 46        |
| 3.2.8    | Κατασκευή Κβαντισμένου Μοντέλου (QKeras)             | 48        |
| 3.3      | Μετατροπή Μοντέλου και Ροή HLS                       | 49        |
| 3.3.1    | Βασική παραμετροποίηση <code>hls4ml</code>           | 49        |
| 3.3.2    | Ανάλυση Κώδικα Υπολογισμού <i>Reuse Factor</i>       | 50        |
| 3.3.3    | Ροή μετάφρασης σε C/C++ και HLS                      | 53        |
| 3.3.4    | Validation και ανάλυση testbench                     | 54        |
| 3.3.5    | Σύνθεση με Vivado                                    | 55        |
| 3.4      | Ανάλυση Κώδικα                                       | 55        |
| 3.4.1    | Ανάλυση του AXI-Stream Wrapper                       | 56        |
| 3.4.2    | AXI-Stream Wrapper: Δομή και Αιτιολόγηση             | 57        |
| 3.4.3    | Ανάλυση Testbench                                    | 60        |
| <b>4</b> | <b>Πλατφόρμα, Υλοποίηση και Ανάπτυξη</b>             | <b>61</b> |
| 4.1      | Εισαγωγή                                             | 61        |
| 4.2      | Υλοποίηση σε FPGA                                    | 62        |
| 4.2.1    | Εισαγωγή και Συσκευασία IP                           | 62        |
| 4.2.2    | Σχεδίαση Block Design                                | 63        |
| 4.2.3    | Ρυθμίσεις Παραμέτρων                                 | 63        |
| 4.2.4    | Wrappers και Templates                               | 63        |
| 4.2.5    | Σενάρια Αυτοματοποίησης και Ανάλυσης                 | 64        |
| 4.2.6    | Σύνοψη                                               | 64        |
| 4.3      | Ενσωμάτωση και Παραμετροποίηση Συστήματος στο Vivado | 64        |

|          |                                                                  |           |
|----------|------------------------------------------------------------------|-----------|
| 4.3.1    | Εισαγωγή IP . . . . .                                            | 65        |
| 4.3.2    | Σχεδίαση Block Design . . . . .                                  | 65        |
| 4.3.3    | Ρύθμιση IP . . . . .                                             | 66        |
| 4.3.4    | Τελική Σύνθεση και Παραγωγή Bitstream . . . . .                  | 70        |
| 4.4      | Ρύθμιση και Ανάπτυξη στην Πλατφόρμα Kria KR260 . . . . .         | 71        |
| 4.4.1    | Ρύθμιση Περιβάλλοντος Ubuntu και Εγκατάσταση Kria PYNQ . . . . . | 71        |
| 4.4.2    | Overlay και Driver . . . . .                                     | 74        |
| 4.4.3    | Εκτέλεση Επί της Συσκευής . . . . .                              | 74        |
| 4.4.4    | Επικύρωση Αποτελεσμάτων . . . . .                                | 75        |
| 4.4.5    | Σύνοψη Ροής Ανάπτυξης . . . . .                                  | 75        |
| 4.5      | Υλοποιήσεις σε CPU και GPU . . . . .                             | 75        |
| 4.5.1    | Εκπαίδευση και Εξαγωγή Μοντέλου . . . . .                        | 76        |
| 4.5.2    | Υλοποίηση σε CPU . . . . .                                       | 76        |
| 4.5.3    | Υλοποίηση σε GPU . . . . .                                       | 76        |
| 4.5.4    | Συζήτηση . . . . .                                               | 77        |
| 4.6      | Διαδικασία Επικύρωσης . . . . .                                  | 77        |
| 4.6.1    | Πλάνο Ελέγχου Ορθής Λειτουργίας . . . . .                        | 77        |
| 4.6.2    | Λειτουργική Επικύρωση . . . . .                                  | 78        |
| 4.6.3    | Επικύρωση Απόδοσης . . . . .                                     | 78        |
| 4.6.4    | Επικύρωση Ενέργειας . . . . .                                    | 79        |
| 4.6.5    | Αυτοματοποίηση και Αναπαραγωγιμότητα . . . . .                   | 79        |
| 4.6.6    | Σύνοψη Αποτελεσμάτων Επικύρωσης . . . . .                        | 80        |
| 4.7      | Ρύθμιση Μέτρησης Ισχύος . . . . .                                | 80        |
| 4.7.1    | Υποδομή Παρακολούθησης . . . . .                                 | 80        |
| 4.7.2    | Υπολογισμός Ισχύος και Ενέργειας . . . . .                       | 80        |
| 4.7.3    | Ενσωμάτωση με τη Ροή Inference . . . . .                         | 81        |
| 4.7.4    | Διαπλατφορμική Παρακολούθηση Ισχύος . . . . .                    | 81        |
| <b>5</b> | <b>Αποτελέσματα</b>                                              | <b>83</b> |
| 5.1      | Αποτελέσματα Εκπαίδευσης . . . . .                               | 83        |
| 5.1.1    | Αποτελέσματα Φάσης Εκπαίδευσης . . . . .                         | 83        |
| 5.1.2    | Ανάλυση ROC Καμπυλών και Σύγκριση Μοντέλων . . . . .             | 86        |
| 5.1.3    | Ανάλυση Κατανομής Βαρών . . . . .                                | 89        |
| 5.2      | Εκτιμήσεις Επίδοσης HLS . . . . .                                | 91        |
| 5.2.1    | Αναφορές Επίδοσης HLS . . . . .                                  | 91        |
| 5.2.2    | Χρήση Πόρων HLS . . . . .                                        | 92        |
| 5.3      | Αναφορές Vivado . . . . .                                        | 93        |
| 5.3.1    | Τελική Χρήση Πόρων . . . . .                                     | 94        |

|           |                                                          |            |
|-----------|----------------------------------------------------------|------------|
| 5.3.2     | Χρονισμός και Ισχύς από Vivado . . . . .                 | 95         |
| 5.3.3     | Εκτίμηση Ισχύος από Vivado . . . . .                     | 95         |
| 5.3.4     | Αποτύπωμα Τοποθέτησης και Δρομολόγησης . . . . .         | 97         |
| 5.4       | Μετρήσεις Επί Πλακέτας . . . . .                         | 100        |
| 5.4.1     | Τάσεις Καθυστέρησης και Ρυθμού Διεκπεραίωσης . . . . .   | 100        |
| 5.4.2     | Μετρήσεις Ισχύος . . . . .                               | 100        |
| 5.4.3     | Ανάλυση Αποδοτικότητας . . . . .                         | 101        |
| 5.4.4     | Ανάλυση Ισχύος και Χρήσης GPU και FPGA . . . . .         | 102        |
| 5.4.5     | Σύνοψη Αποτελεσμάτων . . . . .                           | 104        |
| <b>6</b>  | <b>Συζήτηση, Συμπεράσματα και Μελλοντικές Βελτιώσεις</b> | <b>109</b> |
| 6.1       | Συζήτηση Αποτελεσμάτων . . . . .                         | 109        |
| 6.2       | Συμπεράσματα . . . . .                                   | 111        |
| 6.3       | Μελλοντικές Βελτιώσεις . . . . .                         | 112        |
| <b>A'</b> | <b>Ακρωνύμια</b>                                         | <b>116</b> |

# Κατάλογος σχημάτων

|                                                                                                                |    |
|----------------------------------------------------------------------------------------------------------------|----|
| 2.1.1 Διάγραμμα ενός τεχνητού νευρώνα. . . . .                                                                 | 16 |
| 2.1.2 Παράδειγμα πολυεπίπεδου perceptron (MLP) με στρώμα εισόδου, κρυφά<br>στρώματα και στρώμα εξόδου. . . . . | 17 |
| 2.2.1 Βασική δομή ενός FPGA με επισήμανση των θεμελιωδών δομικών στοιχείων.                                    | 23 |
| 2.2.2 Σύγκριση χρόνου σχεδίασης και απόδοσης μεταξύ x86, DSP, GPU και FPGA                                     | 24 |
| 2.3.1 Απεικόνιση της διαδικασίας σύνθεσης υψηλού επιπέδου . . . . .                                            | 26 |
| 2.3.2 Διάγραμμα ροής του hls4ml . . . . .                                                                      | 26 |
| 2.3.3 Οπτικοποίηση της αρχιτεκτονικής του WSL 2 . . . . .                                                      | 27 |
| 2.5.1 Ενδεικτική εικόνα πλατφόρμας KR260 (SoM) . . . . .                                                       | 36 |
| 2.5.2 Επισκόπηση θυρών PS-PL σε Zynq UltraScale+ . . . . .                                                     | 37 |
| 3.2.1 Κατανομή πλήθους δειγμάτων ανά κλάση στο σύνολο δεδομένων MNIST                                          | 43 |
| 3.2.2 Παράδειγμα εικόνας του MNIST . . . . .                                                                   | 43 |
| 3.2.3 Απεικόνιση της αρχιτεκτονικής του μοντέλου . . . . .                                                     | 44 |
| 3.3.1 Οπτικοποίηση του <i>reuse factor</i> στο hls4ml . . . . .                                                | 50 |
| 3.3.2 Επαλήθευση υλοποίησης στο Vitis HLS . . . . .                                                            | 54 |
| 3.3.3 Εξαγωγή IP στο Vitis HLS . . . . .                                                                       | 55 |
| 4.3.1 Ρυθμίσεις των IP AXI SmartConnect μέρος 1 . . . . .                                                      | 66 |
| 4.3.2 Ρυθμίσεις των IP AXI SmartConnect μέρος 2 . . . . .                                                      | 67 |
| 4.3.3 Ενεργοποίηση της θύρας AXI HP Slave στο PS . . . . .                                                     | 68 |
| 4.3.4 Ρύθμιση συχνότητας ρολογιού PL0 (βήμα 1) . . . . .                                                       | 68 |
| 4.3.5 Ρύθμιση συχνότητας ρολογιού PL0 (βήμα 2) . . . . .                                                       | 69 |
| 4.3.6 Ρύθμιση AXI DMA για 16-bit data path . . . . .                                                           | 69 |
| 4.3.7 Ρύθμιση AXI GPIO . . . . .                                                                               | 70 |
| 4.3.8 Τελικό Block Διάγραμμα . . . . .                                                                         | 70 |
| 5.1.1 Απόδοση εκπαίδευσης και επικύρωσης του βασικού CNN . . . . .                                             | 84 |
| 5.1.2 Απόδοση εκπαίδευσης και επικύρωσης του κλαδεμένου CNN . . . . .                                          | 84 |
| 5.1.3 Απόδοση εκπαίδευσης και επικύρωσης του κβαντισμένου CNN (QKeras,<br>Q0.6) . . . . .                      | 85 |

|                                                                                                                                                                                            |     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 5.1.4Απόδοση εκπαίδευσης και επικύρωσης του κβαντισμένου και κλαδεμένου CNN . . . . .                                                                                                      | 85  |
| 5.1.5Σύγκριση ROC μεταξύ βασικού Keras και κβαντισμένου QKeras . . . . .                                                                                                                   | 87  |
| 5.1.6ROC σύγκριση μεταξύ βασικού Keras μοντέλου και μοντέλου μετά τη μετατροπή σε hls4ml . . . . .                                                                                         | 88  |
| 5.1.7ROC σύγκριση μεταξύ κβαντισμένου QKeras και μεταφρασμένου μοντέλου hls4ml . . . . .                                                                                                   | 88  |
| 5.1.8Κατανομή βαρών ανά στρώμα για το βασικό μοντέλο Keras μετά το κλάδεμα                                                                                                                 | 89  |
| 5.1.9Κατανομή βαρών ανά στρώμα για το κβαντισμένο QKeras μοντέλο μετά το κλάδεμα . . . . .                                                                                                 | 90  |
| 5.3.1Εκτίμηση on-chip ισχύος (Vivado) για το βασικό CNN. Η δυναμική ισχύς κυριαρχείται από το PS (85%), ενώ τα στοιχεία του PL (DSP, λογική, BRAM) συνεισφέρουν λιγότερο από 15% . . . . . | 96  |
| 5.3.2Ιστογράμμο της συνολικής on-chip ισχύος στις παραλλαγές CNN . . . . .                                                                                                                 | 97  |
| 5.3.3Αποτύπωμα τοποθέτησης για το <b>baseline</b> CNN . . . . .                                                                                                                            | 98  |
| 5.3.4Αποτύπωμα τοποθέτησης για το <b>quantized</b> CNN . . . . .                                                                                                                           | 98  |
| 5.3.5Αποτύπωμα τοποθέτησης για το <b>optimized</b> CNN. . . . .                                                                                                                            | 99  |
| 5.4.1Καθυστέρηση έναντι χρήσης DSP στις υλοποιήσεις FPGA . . . . .                                                                                                                         | 101 |
| 5.4.2Throughput έναντι χρήσης DSP . . . . .                                                                                                                                                | 101 |
| 5.4.3Μετρημένη κατανάλωση ισχύος στις υλοποιήσεις FPGA . . . . .                                                                                                                           | 102 |
| 5.4.4Αποδοτικότητα (1/J) έναντι χρήσης DSP . . . . .                                                                                                                                       | 102 |
| 5.4.5Ποσοστό χρήσης GPU (AMD Radeon RX 7800 XT) κατά την εκτέλεση inference . . . . .                                                                                                      | 103 |
| 5.4.6Κατανάλωση ισχύος GPU (Board Power) σε Watt κατά την εκτέλεση inference . . . . .                                                                                                     | 103 |
| 5.4.7Χρονοσειρά ισχύος FPGA (Kria KR260) όπως μετρήθηκε από τον αισθητήρα INA260 . . . . .                                                                                                 | 104 |

# Κατάλογος πινάκων

|                                                                                                                |     |
|----------------------------------------------------------------------------------------------------------------|-----|
| 2.5.1 Κατανομή ρόλων μεταξύ PS και PL στο Zynq UltraScale+ MPSoC. . . .                                        | 36  |
| 2.5.2 Ενδεικτικοί πόροι της PL στο K26 SoM. . . . .                                                            | 38  |
| 4.6.1 Βασικά χαρακτηριστικά του αισθητήρα INA260 . . . . .                                                     | 79  |
| 5.1.1 Τελική απόδοση εκπαίδευσης και επικύρωσης των παραλλαγών του μοντέλου . . . . .                          | 86  |
| 5.2.1 Εκτιμήσεις επίδοσης HLS ανά παραλλαγή σχεδίου (απευθείας από τις αναφορές Vitis HLS) . . . . .           | 91  |
| 5.2.2 Χρήση πόρων HLS στις παραλλαγές του CNN (Vitis HLS, πριν την υλοποίηση)                                  | 92  |
| 5.3.1 Χρήση πόρων μετά την υλοποίηση (Vivado) στις παραλλαγές CNN . . .                                        | 94  |
| 5.3.2 Σύνοψη χρονισμού Vivado ανά παραλλαγή . . . . .                                                          | 95  |
| 5.3.3 Εκτίμηση ισχύος (Vivado) στις παραλλαγές CNN. . . . .                                                    | 96  |
| 5.4.1 Σύνοψη επίδοσης: ισχύς, καθυστέρηση, throughput και αποδοτικότητα για όλες τις υλοποιήσεις FPGA. . . . . | 105 |
| 5.4.2 Σύγκριση μετρικών του ίδιου baseline μοντέλου σε επιταχυντές FPGA, GPU και CPU. . . . .                  | 106 |
| 5.4.3 Σύγκριση μετρικών ταξινόμησης μεταξύ κβαντισμένου και βασικού μοντέλου. . . . .                          | 106 |
| 5.4.4 Μέσες τιμές on-chip κατανάλωσης ισχύος (PL Power) ανά παραλλαγή, σε Watt. . . . .                        | 107 |
| 5.4.5 Μετρικές του κανονικού και του πλήρως βελτιστοποιημένου μοντέλου (χρονικές μετρικές). . . . .            | 107 |

# Κεφάλαιο 1

## Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται το αντικείμενο και οι στόχοι της εργασίας, μια σύντομη επισκόπηση συναφούς έρευνας και η οργάνωση του τόμου, ώστε ο αναγνώστης να έχει καθαρή εικόνα του πλαισίου και της συνεισφοράς.

### 1.1 Αντικείμενο της διπλωματικής

Η μηχανική μάθηση (Machine Learning, ML) τα τελευταία χρόνια έχει αναπτυχθεί κυρίως για εκτέλεση σε περιβάλλοντα νεφοϋπολογιστικής (Cloud Computing) ή σε υπολογιστές υψηλών επιδόσεων (High-Performance Computing, HPC), τα οποία όμως χαρακτηρίζονται από αυξημένη κατανάλωση ισχύος. Η νεφοϋπολογιστική (Cloud Computing) βασίζεται σε απομακρυσμένα κέντρα δεδομένων που παρέχουν εικονικοποιημένους υπολογιστικούς πόρους μέσω Διαδικτύου, ενώ η εκτέλεση στο άκρο (Edge Computing) μεταφέρει τον υπολογιστικό φόρτο κοντά στην πηγή των δεδομένων, όπως αισθητήρες, ρομποτικά συστήματα ή αυτόνομα οχήματα. Παρότι η νεφοϋπολογιστική (Cloud Computing) προσφέρει εύκολη κλιμάκωση, συνεπάγεται λανθάνοντα χρόνο (latency) λόγω μεταφοράς δεδομένων, υψηλές απαιτήσεις σε εύρος ζώνης και πιθανά ζητήματα ιδιωτικότητας. Αντίθετα, η υπολογιστική στο άκρο (Edge Computing) μειώνει την εξάρτηση από το δίκτυο και επιτρέπει ταχύτερη και πιο προβλέψιμη λειτουργία, καθιστώντας την κατάλληλη για εφαρμογές που απαιτούν άμεση απόκριση.

Το κεντρικό ερώτημα που εξετάζει η παρούσα εργασία είναι κατά πόσο είναι εφικτή η εκτέλεση μοντέλων βαθιάς μάθησης (Deep Learning, DL) απευθείας κοντά στο σημείο λήψης των δεδομένων. Η απαίτηση αυτή συνδέεται με δύο βασικούς στόχους: την ελαχιστοποίηση της κατανάλωσης ισχύος και τη διατήρηση του λανθάνοντος χρόνου (latency) σε επίπεδα που ικανοποιούν περιορισμούς πραγματικού χρόνου (real-time constraints). Η πρόκληση εντοπίζεται στην ισορροπία ανάμεσα στην ακρίβεια πρόβλεψης και στη χρήση των διαθέσιμων υλικών πόρων, ώστε να καταστεί δυνατή η ενσωμάτωση

τέτοιων μοντέλων σε φορητές ή ενσωματωμένες συσκευές.

Για την επίτευξη των στόχων αυτών διερευνάται η χρήση πλατφόρμων συστήματος-σε-τσιπ (System-on-Chip, SoC) με προγραμματιζόμενες συστοιχίες πυλών (Field Programmable Gate Arrays, FPGA). Οι πλατφόρμες αυτές επιτρέπουν εξειδίκευση της μικροαρχιτεκτονικής στο εκάστοτε μοντέλο, προσφέροντας υψηλό βαθμό παραλληλισμού και ακριβή έλεγχο του χρονισμού. Ένα βασικό πλεονέκτημα των FPGA είναι η δυνατότητα παραμετροποίησης, που τα καθιστά ευέλικτα για ετερογενείς εφαρμογές, σε αντίθεση με τους επεξεργαστές γενικού σκοπού (Central Processing Unit, CPU) και τις μονάδες γραφικών (Graphics Processing Unit, GPU) που ακολουθούν προκαθορισμένη αρχιτεκτονική. Ωστόσο, η ανάπτυξη σε FPGA ενέχει μεγαλύτερη πολυπλοκότητα σχεδίασης και απαιτεί κατάλληλες μεθοδολογίες και εργαλειακή υποστήριξη.

Η μετάβαση από το λογισμικό στο υλικό πραγματοποιείται μέσω σύνθεσης υψηλού επιπέδου (High-Level Synthesis, HLS), η οποία μετατρέπει περιγραφές σε γλώσσα υψηλού επιπέδου σε περιγραφή επιπέδου μεταφοράς καταχωρητών (Register-Transfer Level, RTL). Επιπλέον, εφαρμόζεται αριθμητική σταθερής υποδιαστολής (fixed-point arithmetic), ώστε να μειώνεται η κατανάλωση υπολογιστικών πόρων και ενέργειας χωρίς σημαντική απώλεια στην ακρίβεια. Με τον τρόπο αυτό καθίσταται δυνατή η ανάπτυξη υλοποιήσεων που ανταποκρίνονται στις απαιτήσεις πραγματικού χρόνου και αξιοποιούν τα πλεονεκτήματα των FPGA.

Πέρα από την ακαδημαϊκή σημασία της μελέτης, η ανάγκη για επιτάχυνση μοντέλων βαθιάς μάθησης σε πλατφόρμες τύπου FPGA προκύπτει και από πληθώρα πρακτικών εφαρμογών *edge τεχνητής νοημοσύνης* (Edge AI). Τέτοιες εφαρμογές περιλαμβάνουν συστήματα βιομηχανικής όρασης για ανίχνευση ελαττωματικών προϊόντων σε γραμμές παραγωγής, ρομποτικά οχήματα που απαιτούν λήψη αποφάσεων με ελάχιστη καθυστέρηση, έξυπνες κάμερες που εκτελούν τοπική επεξεργασία εικόνας χωρίς μεταφορά δεδομένων σε cloud, καθώς και φορητές ή ιατρικές συσκευές που πραγματοποιούν ανάλυση βιολογικών σημάτων με χαμηλή κατανάλωση ενέργειας. Οι περιπτώσεις αυτές αναδεικνύουν την αναγκαιότητα ανάπτυξης αποδοτικών CNN επιταχυντών σε επαναπρογραμματιζόμενο υλικό, ικανών να υποστηρίξουν λειτουργία πραγματικού χρόνου και να ανταποκριθούν στους περιορισμούς ενέργειας και χώρου.

Στο πλαίσιο αυτό, η παρούσα διπλωματική εργασία επικεντρώνεται στη μελέτη και υλοποίηση ενός συνελκτικού νευρωνικού δικτύου (Convolutional Neural Network, CNN) για την ταξινόμηση του συνόλου δεδομένων MNIST. Το CNN θα υλοποιηθεί και θα αξιολογηθεί σε CPU, GPU και FPGA, ώστε να αναδειχθούν συγκριτικά τα πλεονεκτήματα και οι περιορισμοί κάθε πλατφόρμας.

## 1.2 Παρόμοια έρευνα

Η επιτάχυνση συνελκτικών νευρωνικών δικτύων (Convolutional Neural Networks, CNN) σε προγραμματιζόμενες συστοιχίες πυλών πεδίου (Field-Programmable Gate Arrays, FPGA) έχει μελετηθεί εκτενώς, με στόχο τη μείωση του λανθάνοντος χρόνου (latency), τη βέλτιστη αξιοποίηση πόρων και τη βελτίωση της ενεργειακής αποδοτικότητας. Οι πιο ώριμες προσεγγίσεις οργανώνονται γύρω από: (α) αρχιτεκτονικές ροής δεδομένων (dataflow architectures) με διασωλήνωση (pipelining) ανά επίπεδο, (β) διερεύνηση χώρου σχεδίασης (Design Space Exploration, DSE) για ισορροπία μεταξύ υπολογισμού και μνήμης, και (γ) συμπληρωματικές τεχνικές συμπίεσης και κβαντοποίησης (quantization). Συστηματικές επισκοπήσεις και μελέτες (*surveys*) τεκμηριώνουν τα παραπάνω και προτείνουν ενιαίες μεθοδολογίες αξιολόγησης, ιδίως για ροές μετασχηματισμού CNN σε FPGA (CNN-to-FPGA toolflows), αναδεικνύοντας κοινά σημεία σύγκρισης όπως ο λανθάνων χρόνος (latency), η διεκπεραιωτική ικανότητα (throughput), η κατανάλωση ενέργειας και η χρήση υλικών πόρων. [3], [43], [44]

Η εργασία *FINN* εισάγει πλήρως ροϊκή (streaming) αρχιτεκτονική για δυαδικοποιημένα ή χαμηλής ακρίβειας νευρωνικά δίκτυα, αξιοποιώντας εξαιρετικά χαμηλά πλάτη λέξης (1–2 bits για βάρη και ενεργοποιήσεις) ώστε να αυξάνει θεαματικά τον παραλληλισμό και να μειώνει δραστικά το κόστος μνήμης και υπολογισμού στα FPGA. [42] Η προσέγγιση αυτή αποτελεί σημείο αναφοράς για εξαιρετικά χαμηλό λανθάνοντα χρόνο (ultra-low-latency inference) και έχει επεκταθεί σε νεότερες παραλλαγές και εργαλεία. [39]

Τα εργαλεία *DNNWeaver* και οι νεότερες ροές CNN-to-FPGA επιδιώκουν, από την πλευρά του λογισμικού, να παραγάγουν παραμετροποιήσιμους επιταχυντές που γεφυρώνουν τα πλαίσια εκπαίδευσης (π.χ. *Caffe*, *Keras*) με υλοποιήσιμες περιγραφές υλικού, υποστηρίζοντας ταυτόχρονα διερεύνηση χώρου σχεδίασης (DSE) για αποδοτικές χαρτογραφήσεις σε διαφορετικές πλατφόρμες. [36], [43]

Ιδιαίτερο ενδιαφέρον παρουσιάζει η εργασία *hls4ml* των Aarrestad et al, η οποία προτείνει ένα ολοκληρωμένο πλαίσιο μεταφοράς νευρωνικών δικτύων σε επίπεδο υλικού μέσω σύνθεσης υψηλού επιπέδου (High-Level Synthesis, HLS). [1] Το *hls4ml* συνδυάζει εργαλεία από το οικοσύστημα του *TensorFlow/Keras* με περιγραφές σε C/C++, δημιουργώντας αυτόματα για το *Vivado HLS* ή το *Vitis HLS*. Η ροή του περιλαμβάνει στάδια παραμετροποίησης, ποσοτικοποίησης (quantization) και επιλογής παραμέτρων ροής δεδομένων (reuse factor, unrolling, pipeline depth), επιτρέποντας την ταχεία παραγωγή και αξιολόγηση επιταχυντών για CNN και πλήρως συνδεδεμένα δίκτυα. Η μελέτη τους απέδειξε ότι, με κατάλληλη ρύθμιση παραμέτρων, μπορούν να επιτευχθούν λανθάνοντες χρόνοι (latency) της τάξης των μικροδευτερολέπτων σε συνελκτικές αρχιτεκτονικές,

αναδεικνύοντας την καταλληλότητα των SoC FPGA για συμπεράσματα πραγματικού χρόνου (real-time inference) στο άκρο (Edge). [13]

Πρόσφατες ερευνητικές εργασίες συγκλίνουν στο ότι ο συνδυασμός αραίωσης (pruning), χαμηλής ακρίβειας (π.χ. 8 bit και κάτω) και κατάλληλου dataflow είναι κρίσιμος για την επίτευξη ταυτόχρονα χαμηλού λανθάνοντος χρόνου και υψηλής ενεργειακής αποδοτικότητας, ιδίως όταν ο περιοριστικός παράγοντας είναι το εύρος ζώνης της μνήμης (memory bandwidth). [44] Ερευνητικά αποτελέσματα με έμφαση στο latency για συστήματα FPGA στο άκρο (edge FPGAs) δείχνουν ότι η αποφυγή πρόσβασης σε εξωτερική μνήμη DRAM και η χρήση αποκλειστικά της ενσωματωμένης μνήμης (on-chip memory) μπορεί να βελτιώσει σημαντικά τόσο τον χρόνο εκτέλεσης όσο και την κατανάλωση ενέργειας. [32]

Με βάση τα παραπάνω αποτελέσματα, η παρούσα εργασία: (i) υιοθετεί χαρτογράφηση επιπέδων ροής και διασωλήνωσης (pipelined), (ii) εφαρμόζει αραίωση (pruning) και κβαντοποίηση σταθερής υποδιαστολής (fixed-point quantization) ώστε τα βάρη και οι ενεργοποιήσεις να παραμένουν εντός της ενσωματωμένης μνήμης, και (iii) αξιοποιεί τη σύνθεση υψηλού επιπέδου (High-Level Synthesis, HLS) μέσω της *hls4ml* για ταχεία μετάβαση από το εκπαιδευμένο μοντέλο σε περιγραφή επιπέδου μεταφοράς καταχωρητών (Register-Transfer Level, RTL) μέσω C/C++, στοχεύοντας χαμηλό λανθάνοντα χρόνο και υψηλή ενεργειακή αποδοτικότητα σε πλατφόρμες τύπου SoC FPGA. [1], [13]

### 1.3 Στόχος της διπλωματικής

Στόχος της παρούσας εργασίας είναι να τεκμηριωθεί, μέσω συστηματικής αξιολόγησης, ότι ένα συνελικτικό νευρωνικό δίκτυο (Convolutional Neural Network, CNN) μικρής έως μέσης πολυπλοκότητας, μετά από κατάλληλη συμπίεση και κβαντοποίηση (quantization), μπορεί να υλοποιηθεί αποδοτικά σε προγραμματιζόμενη συστοιχία πυλών πεδίου (Field-Programmable Gate Array, FPGA) με χρήση σύνθεσης υψηλού επιπέδου (High-Level Synthesis, HLS). Η εργασία στοχεύει στην επίτευξη χαμηλού λανθάνοντος χρόνου (latency) και αποτελεσματικής αξιοποίησης υλικών πόρων, σε σύγκριση με ισοδύναμες υλοποιήσεις σε επεξεργαστές γενικού σκοπού (Central Processing Unit, CPU) και σε μονάδες γραφικών (Graphics Processing Unit, GPU). Παράλληλα, αξιολογείται η κατανάλωση ισχύος και η ενεργειακή αποδοτικότητα, με έμφαση στη σύγκριση μεταξύ FPGA και GPU, όπου είναι δυνατή η αξιόπιστη μέτρηση ενεργειακών χαρακτηριστικών. Η επιδίωξη δεν περιορίζεται στη μείωση του χρόνου εκτέλεσης, αλλά επεκτείνεται και στη διατήρηση της ακρίβειας ταξινόμησης σε αποδεκτά επίπεδα, ώστε η προτεινόμενη λύση να είναι πρακτικά εφαρμόσιμη σε περιβάλλοντα πραγματικού χρόνου.

Η πειραματική ροή περιλαμβάνει την εκπαίδευση του μοντέλου στο σύνολο δεδομένων *MNIST*, τη συμπίεση μέσω αραίωσης (pruning) και την εκπαίδευση με επίγνωση κβαντοποίησης (quantization-aware training), προκειμένου να επιτευχθεί μετάβαση σε αριθμητική σταθερής υποδιαστολής (fixed-point arithmetic). Η μεταφορά του εκπαιδευμένου μοντέλου στο υλικό πραγματοποιείται αυτόματα μέσω του πλαισίου *hls4ml*, ενός ανοικτού κώδικα εργαλείου που μετατρέπει μοντέλα μηχανικής μάθησης (Machine Learning, ML) σε κώδικα C/C++ κατάλληλο για σύνθεση υψηλού επιπέδου (HLS). Στη συνέχεια, πραγματοποιείται σύνθεση σε επίπεδο μεταφοράς καταχωρητών (Register-Transfer Level, RTL) με το εργαλείο *Xilinx Vitis HLS*, το οποίο μεταφράζει τον παραγόμενο κώδικα C/C++ σε υλοποιήσιμη περιγραφή υλικού. Ακολουθεί η δημιουργία έργου στο *Xilinx Vivado*, όπου ενσωματώνονται τα απαραίτητα περιφερειακά υποσυστήματα (peripheral components) και διαμορφώνεται η συνολική αρχιτεκτονική του συστήματος, οδηγώντας στη δημιουργία του αρχείου *bitstream* που προγραμματίζει την πλατφόρμα FPGA.

Η αξιολόγηση πραγματοποιείται στο ίδιο μοντέλο, εκτελεσμένο σε CPU, GPU και FPGA, με καταγραφή και σύγκριση των μετρικών λανθάνοντος χρόνου, διεκπεραιωτικής ικανότητας (throughput), χρήσης υλικών πόρων και ακρίβειας ταξινόμησης. Η ανάλυση ενεργειακής αποδοτικότητας βασίζεται σε μετρήσεις κατανάλωσης ισχύος (power measurements) που πραγματοποιούνται για τις πλατφόρμες FPGA και GPU. Μέσα από αυτή τη συγκριτική μελέτη αναδεικνύονται τα πλεονεκτήματα και οι περιορισμοί κάθε πλατφόρμας, επιβεβαιώνοντας την καταλληλότητα των FPGA για επιταχυνόμενες εφαρμογές τεχνητής νοημοσύνης (Artificial Intelligence, AI) σε περιβάλλοντα πραγματικού χρόνου.

## 1.4 Οργάνωση της εργασίας

Η εργασία αποτελείται από έξι κύρια κεφάλαια, τα οποία ακολουθούν μια προοδευτική δομή από τη θεωρία προς την υλοποίηση και την αξιολόγηση αποτελεσμάτων. Στο **Κεφάλαιο 2** παρουσιάζεται το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση της εργασίας. Αναλύονται οι αρχές λειτουργίας των Συνελικτικών Νευρωνικών Δικτύων, η αρχιτεκτονική και η λειτουργία των FPGA, καθώς και οι τεχνικές Σύνθεσης Υψηλού Επιπέδου. Παρουσιάζονται επίσης τα εργαλεία που αξιοποιούνται και περιγράφεται η πλατφόρμα *AMD Kria KR260*, μαζί με τις μετρικές αξιολόγησης απόδοσης και ενέργειας που χρησιμοποιούνται στα επόμενα κεφάλαια. Το **Κεφάλαιο 3** περιγράφει τη μεθοδολογία που ακολουθήθηκε για τη σχεδίαση, εκπαίδευση και μεταφορά του CNN σε υλοποιήσιμη μορφή. Αναλύεται η διαδικασία εκπαίδευσης στο περιβάλλον Keras/QKeras, η εφαρμογή τεχνικών συμπίεσης και κβαντοποίησης, και η αυτόματη μετατροπή του

μοντέλου σε C/C++ μέσω του εργαλείου `hls4ml`. Παρουσιάζεται η σύνθεση σε RTL με το Vitis HLS, οι βελτιστοποιήσεις και η ροή επαλήθευσης πριν από την ενσωμάτωση στο Vivado. Στο **Κεφάλαιο 4** αναλύεται η πλήρης διαδικασία υλοποίησης και ανάπτυξης του επιταχυντή CNN στην πλατφόρμα *AMD Kria KR260*. Περιγράφεται η ενσωμάτωση του HLS πυρήνα στο Vivado, η δημιουργία του bitstream, η ανάπτυξη μέσω του *PYNQ framework* και η ρύθμιση του συστήματος PS-PL. Παρουσιάζονται επίσης οι υλοποιήσεις αναφοράς σε CPU και GPU για συγκριτικούς σκοπούς και αναλύεται η διαδικασία επικύρωσης της λειτουργίας και της κατανάλωσης ισχύος. Το **Κεφάλαιο 5** επικεντρώνεται στην παρουσίαση και ανάλυση των αποτελεσμάτων. Παρουσιάζονται τα αποτελέσματα εκπαίδευσης και επαλήθευσης των μοντέλων, οι αναφορές HLS και Vivado σχετικά με τη χρήση πόρων και τον χρονισμό, καθώς και οι *on-board* μετρήσεις καθυστέρησης, *throughput* και κατανάλωσης ενέργειας. Τα δεδομένα συγκρίνονται μεταξύ των υλοποιήσεων FPGA, GPU και CPU, αναδεικνύοντας τα πλεονεκτήματα και τους περιορισμούς κάθε πλατφόρμας. Τέλος, το **Κεφάλαιο 6** συνοψίζει τα κύρια συμπεράσματα της μελέτης και συζητά τη συμβολή της στην αξιοποίηση των FPGA για επιταχυνόμενες εφαρμογές Τεχνητής Νοημοσύνης. Παρουσιάζονται οι γενικές παρατηρήσεις σχετικά με την επίδοση, την ενεργειακή αποδοτικότητα και τη μεταφορά του μοντέλου, καθώς και προτάσεις για μελλοντική έρευνα και επέκταση της εργασίας σε πιο σύνθετες αρχιτεκτονικές ή σύνολα δεδομένων μεγαλύτερης κλίμακας.

## Κεφάλαιο 2

# Θεωρητικό Υπόβαθρο και Έννοιες

### 2.1 Συνελικτικά Νευρωνικά Δίκτυα

Στις επόμενες ενότητες παρουσιάζεται μια σύντομη επισκόπηση των νευρωνικών δικτύων (Neural Networks) γενικότερα και των συνελικτικών νευρωνικών δικτύων (Convolutional Neural Networks, CNN) ειδικότερα. Αρχικά δίνεται μια διαισθητική περιγραφή της λειτουργίας των νευρωνικών δικτύων και της διαδικασίας εκπαίδευσής τους. Στη συνέχεια εξετάζονται τα CNN, μια αρχιτεκτονική ιδιαίτερα κατάλληλη για επεξεργασία εικόνων, και παρουσιάζεται ο τρόπος κατασκευής τους. Τέλος, περιγράφονται συνοπτικά οι σημαντικότερες τοπολογίες CNN που χρησιμοποιούνται για ταξινόμηση εικόνων.

#### 2.1.1 Εισαγωγή στα Νευρωνικά Δίκτυα

##### Βιολογική έμπνευση

Τα νευρωνικά δίκτυα αποτελούν υπολογιστικές αρχιτεκτονικές που εμπνεύστηκαν από τα βιολογικά νευρικά συστήματα. Οι βιολογικοί εγκέφαλοι περιλαμβάνουν μεγάλους πληθυσμούς νευρώνων που συνδέονται μέσω συνάψεων. Κάθε νευρώνας ολοκληρώνει τα εισερχόμενα συναπτικά σήματα και παράγει σήμα εξόδου όταν το δυναμικό της μεμβράνης του υπερβεί ένα κατώφλι [17], [22]. Η παρατήρηση αυτή οδήγησε στη διαμόρφωση αφηρημένων μοντέλων τεχνητών νευρώνων (artificial neurons), τα οποία υπολογίζουν σταθμισμένο άθροισμα εισόδων και εφαρμόζουν μη γραμμική συνάρτηση ενεργοποίησης, σχηματίζοντας κατευθυνόμενα δίκτυα απλών μονάδων επεξεργασίας [17], [35]. Η μάθηση σε τέτοια δίκτυα συνίσταται στην προσαρμογή των συναπτικών βαρών ώστε να βελτιωθεί η απόδοση σε μια εργασία, με ιδέες συγγενείς προς τη θεωρία του Hebb [17].

Τα πρῶμα τεχνητά μοντέλα, ὅπως ο *perceptron*, ἔδειξαν ὅτι μποροῦν νὰ μάθουν γραμμικά διαχωρίσιμα προβλήματα. Ἡ μετέπειτα ἀνάπτυξη τῆς *οπισθοδρομικῆς διάδοσης σφάλματος* (backpropagation) επέτρεψε σὲ πολυεπίπεδα δίκτυα νὰ μάθουν ιεραρχικὲς καὶ μὴ γραμμικὲς ἀναπαραστάσεις μεγάλης κλίμακας [17]. Στὴν υπολογιστικὴ ὄραση, οὐ βιολογικὲς ἰδέες γιὰ *τοπικά πεδία υποδοχῆς* (receptive fields) καὶ ιεραρχικὴ ἐξαγωγή χαρακτηριστικῶν οδήγησαν στὴν ἀνάπτυξη συνελκτικῶν ἀρχιτεκτονικῶν, ποὺ ἐκμεταλλεύονται *χωρικὴ τοπικότητα* καὶ *μοίρασμα παραμέτρων* [28].

### Τεχνητός νευρώνας

Ένας τεχνητός νευρώνας μοντελοποιεῖται ὡς ἀπλὴ υπολογιστικὴ μονάδα. Γιὰ διά- νυσμα εἰσόδου  $\mathbf{x} \in \mathbb{R}^d$ , βάρη  $\mathbf{w} \in \mathbb{R}^d$  καὶ μεροληψία  $b \in \mathbb{R}$ :

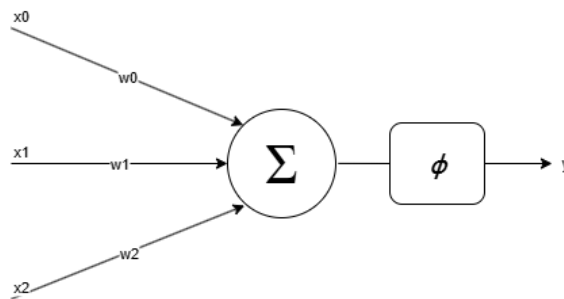
$$z = \mathbf{w}^\top \mathbf{x} + b, \quad y = \phi(z), \quad (2.1)$$

ὅπου  $\phi(\cdot)$  εἶναι μὴ γραμμικὴ συνάρτηση ενεργοποίησης. Συνήθεις ἐπιλογές εἶναι ἡ *διορθωμένη γραμμικὴ μονάδα* (Rectified Linear Unit, ReLU)  $\text{ReLU}(z) = \max(0, z)$ , ἡ *λογιστικὴ* (sigmoid)  $\sigma(z) = \frac{1}{1+e^{-z}}$  καὶ ἡ *υπερβολικὴ εφαπτομένη* (tanh) [17]. Ἡ στοίβαξη πολλῶν τέτοιων μονάδων σὲ στρώσεις σχηματίζει *πολυεπίπεδο perceptron* (Multi-Layer Perceptron, MLP), ὅπου κάθε στρώση εφαρμόζει γραμμικὸ μετασχηματισμὸ καὶ ἐν συνεχείᾳ μὴ γραμμικότητα, ἐπιτρέποντας προσέγγιση πολὺπλοκων συναρτήσεων [17], [35].

Σε μορφή διανυσματικῆς–μητρικῆς ἀναπαράστασης, μὴ στρώση με  $m$  νευρώνες λαμβάνει  $\mathbf{x} \in \mathbb{R}^d$  καὶ παράγει:

$$\mathbf{y} = \phi(W\mathbf{x} + \mathbf{b}), \quad W \in \mathbb{R}^{m \times d}, \quad \mathbf{b} \in \mathbb{R}^m, \quad (2.2)$$

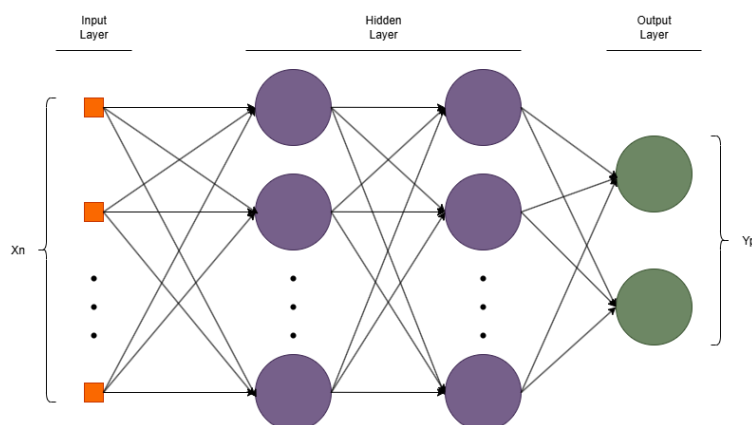
με  $\phi$  νὰ εφαρμόζεται κατὰ στοιχείο. Κατὰ τὴν ἐκπαίδευση, οὐ παράμετροι  $(W, \mathbf{b})$  προσαρμόζονται με μεθόδους βελτιστοποίησης βασισμένες σὲ κλίσεις.



Σχῆμα 2.1.1: Διάγραμμα ἐνὸς τεχνητοῦ νευρώνα.

## Εκπαίδευση δικτύου

Οι παράμετροι ενός νευρωνικού δικτύου μαθαίνονται από δεδομένα. Στο εποπτευόμενο πλαίσιο (supervised learning), η εκπαίδευση γίνεται σε *εποχές* (epochs) πάνω σε επισημασμένα παραδείγματα. Κάθε επανάληψη εκτελεί *εμπρόσθια διέλευση* (forward pass), υπολογίζει *συνάρτηση απώλειας* (loss) έναντι των ετικετών αλήθειας και ενημερώνει τα βάρη ώστε να ελαχιστοποιηθεί η απώλεια [17]. Η βελτιστοποίηση γίνεται συνήθως με *στοχαστική κατάβαση κλίσης* (Stochastic Gradient Descent, SGD), με μέγεθος βήματος ελεγχόμενο από τον *ρυθμό μάθησης* (learning rate). Οι βαθμίδες υπολογίζονται αποδοτικά μέσω *οπισθοδιάδοσης* (backpropagation), αυτόματη διαφοροποίηση αντίστροφης μορφής που εφαρμόζει τον κανόνα αλυσίδας στρώση-προς-στρώση [17]. Στην πράξη χρησιμοποιούνται *μίνι-παρτίδες* (mini-batches), που σταθεροποιούν τις εκτιμήσεις κλίσεων και επιτρέπουν διανυσματικοποιημένους υπολογισμούς σε υλικό με παραλληλία γραμμικής άλγεβρας (GPUs, DSPs, σύγχρονοι CPUs) [35].



Σχήμα 2.1.2: Παράδειγμα πολυεπίπεδου perceptron (MLP) με στρώμα εισόδου, κρυφά στρώματα και στρώμα εξόδου.

### 2.1.2 Εισαγωγή στα CNNs

Τα συνελκτικά νευρωνικά δίκτυα είναι αρχιτεκτονικές *εμπρόσθιας τροφοδότησης* (feedforward) που αξιοποιούν τρεις επαγόμενες προκαταλήψεις των φυσικών εικόπων: *τοπικότητα*, *στασιμότητα* και *συνθετικότητα*. Τα CNN αντικαθιστούν τις πυκνές μητρικές πολλαπλασιάσεις με *συνελίξεις* (convolutions) με παραμέτρους που μαθαίνονται, επιτυγχάνοντας μοίρασμα παραμέτρων στις χωρικές θέσεις και επιτρέποντας κλιμακούμενη μάθηση σε υψηλοδιάστατες εισόδους.

Για να κατανοηθεί η εσωτερική λειτουργία των CNN, είναι χρήσιμη η παρουσίαση των βασικών τύπων στρώσεων. Κάθε στρώση επιτελεί διακριτό ρόλο, από την εξαγωγή τοπικών χαρακτηριστικών έως τη συμπίεση πληροφορίας και την τελική ταξινόμηση.

## Τύποι στρώσεων

### Συνελικτικές στρώσεις (convolutional layers)

Για 2D συνελικτική στρώση με  $C_{\text{in}}$  κανάλια εισόδου και  $C_{\text{out}}$  εξόδου, οι πυρήνες  $K \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w}$  εφαρμόζονται στην είσοδο  $X \in \mathbb{R}^{C_{\text{in}} \times h_{\text{in}} \times w_{\text{in}}}$  (αμεροληψίες παραλείπονται). Με πλήρωση (padding)  $p_h, p_w$ , βήμα (stride)  $s_h, s_w$  και διάταση (dilation)  $d_h, d_w$ , το χωρικό μέγεθος εξόδου είναι:

$$h_{\text{out}} = \left\lfloor \frac{h_{\text{in}} + 2p_h - d_h(k_h - 1) - 1}{s_h} \right\rfloor + 1, \quad w_{\text{out}} = \left\lfloor \frac{w_{\text{in}} + 2p_w - d_w(k_w - 1) - 1}{s_w} \right\rfloor + 1. \quad (2.3)$$

Η ρύθμιση “same” χρησιμοποιεί  $p = \lfloor k/2 \rfloor$  (με  $d = 1$ ) ώστε να διατηρείται το χωρικό μέγεθος όταν  $s = 1$ . Συνελίξεις  $1 \times 1$  χρησιμοποιούνται για αλλαγή διαστασιμότητας καναλιών, μείωση υπολογιστικού κόστους και εισαγωγή διακαναλικών αλληλεπιδράσεων.

### Στρώσεις ενεργοποίησης (nonlinearities)

Μη γραμμικότητες κατά στοιχείο ακολουθούν τις συνελίξεις και αυξάνουν την αναπαραστατική ισχύ. Η προεπιλογή στα σύγχρονα CNN είναι η ReLU [16]. Παραλλαγές: Παραμετρική ReLU (PReLU)  $\text{PReLU}(x) = \max(\alpha x, x)$  [20], Maxout [18], Εκθετικές Γραμμικές Μονάδες (Exponential Linear Units, ELU) [10]. Οι sigmoid/tanh χρησιμοποιούνται λιγότερο σε βαθιά CNN λόγω κορεσμού.

### Στρώσεις συγκέντρωσης (pooling)

Το pooling μειώνει τη χωρική ανάλυση διατηρώντας κρίσιμες πληροφορίες. Συνήθεις επιλογές: μέγιστη συγκέντρωση (max pooling) ή μέση συγκέντρωση (average pooling) σε παράθυρα  $2 \times 2$  ή  $3 \times 3$  με βήμα 2. Η ολική μέση συγκέντρωση (global average pooling) συνοψίζει ανά κανάλι σε όλο τον χάρτη χαρακτηριστικών, δίνοντας μέγεθος  $1 \times 1$  και συχνά αντικαθιστώντας μεγάλους πλήρως συνδεδεμένους ταξινομητές [30]. Εναλλακτικά, μείωση μπορεί να επιτευχθεί και μέσω συνελίξεων με μεγαλύτερο stride.

### Στρώσεις κανονικοποίησης (normalization)

Η Κανονικοποίηση Δέσμης (Batch Normalization, BN) κανονικοποιεί προ-ενεργοποιήσεις ώστε να έχουν περίπου μηδενική μέση τιμή και μοναδιαία διασπορά ανά μίνι-παρτίδα, με αφινικές παραμέτρους  $(\gamma, \beta)$ , βελτιώνοντας τη βελτιστοποίηση και επιτρέποντας υψηλότερους ρυθμούς μάθησης [24]. Η Τοπική Κανονικοποίηση Απόκρισης (Local Response Normalization, LRN) εισήχθη στο AlexNet [26] αλλά σπανίως χρησιμοποιείται πλέον, όπου η BN (και άλλες παραλλαγές κανονικοποίησης) προτιμάται.

**Πλήρως συνδεδεμένες στρώσεις (fully connected)**

Οι *dense* στρώσεις χαρτογραφούν τα χαρακτηριστικά σε *λογάριθμους τάξεων* (logits). Ιστορικά συγκέντρωναν τις περισσότερες παραμέτρους, αλλά πολλά σύγχρονα CNN τις αντικαθιστούν με *global average pooling* και μικρούς ταξινομητές, περιορίζοντας παραμέτρους και υπερεκπαίδευση.

**Τεχνικές τακτικοποίησης (regularization)**

Το *Dropout* απενεργοποιεί τυχαία μονάδες κατά την εκπαίδευση, μειώνοντας την εξάρτηση από συγκεκριμένα μονοπάτια και βελτιώνοντας τη γενίκευση [38]. Στα CNN εφαρμόζεται κυρίως στους τελικούς ταξινομητές και σε συνδυασμό με *αύξηση δεδομένων* (data augmentation). Στο στάδιο *πρόβλεψης* (inference) το *Dropout* δεν εφαρμόζεται· η επίδρασή του ενσωματώνεται μέσω κατάλληλης κλιμάκωσης.

**Ταξινομητής (softmax).** Για πολυκατηγορική ταξινόμηση, τα τελικά *logits*  $\mathbf{z} \in \mathbb{R}^C$  μετατρέπονται σε πιθανότητες μέσω *softmax*:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^C e^{z_k}}, \quad \sum_{i=1}^C \text{softmax}(\mathbf{z})_i = 1. \quad (2.4)$$

Η εκπαίδευση χρησιμοποιεί συνήθως *διασταυρούμενη εντροπία* (cross-entropy).

**Σύνοψη.** Ο συνδυασμός των παραπάνω στρώσεων επιτρέπει στα CNN να εξάγουν ιεραρχικά χαρακτηριστικά και να εκτελούν ταξινόμηση μεγάλης κλίμακας με υπολογιστική αποδοτικότητα, αξιοποιώντας χωρική τοπικότητα και μοίρασμα παραμέτρων.

**2.1.3 Πλαίσια Μηχανικής Μάθησης**

Η σύγχρονη πρακτική στη βαθιά μάθηση (Deep Learning) περιστρέφεται γύρω από τρία συμπληρωματικά πλαίσια. Το **PyTorch** προσφέρει *άμεση εκτέλεση με αυτόματη διαφοροποίηση* και διαδρομές μεταγλώττισης για υψηλότερη απόδοση [33]. Το **TensorFlow** παρέχει κλιμακούμενο περιβάλλον με *άμεση εκτέλεση* και *εκτέλεση γραφήματος* μέσω `tf.function`, υποστήριξη πολλαπλών συσκευών και εξαγωγή μέσω *SavedModel*, XLA [2]. Βασισμένο στο TensorFlow, το **Keras** προσφέρει υψηλού επιπέδου API για ορισμό, εκπαίδευση και αξιολόγηση μοντέλων με ελάχιστο περιττό κώδικα. Στο πλαίσιο της παρούσας εργασίας, η εκπαίδευση και η κβαντοποίηση πραγματοποιούνται κυρίως σε Keras, ενώ η μεταφορά σε FPGA γίνεται μέσω *σύνθεσης υψηλού επιπέδου* με *Vitis HLS* και *hls4ml*.

### 2.1.4 Σημαντικές τοπολογίες CNN για ταξινόμηση

Στην εξέλιξη των συνελικτικών νευρωνικών δικτύων έχουν προταθεί αρχιτεκτονικές που ισορροπούν διαφορετικά την ακρίβεια, την πολυπλοκότητα και το υπολογιστικό/μνημονικό κόστος. Παρακάτω συνοψίζονται οι πιο επιδραστικές τοπολογίες, με έμφαση σε ό,τι είναι κρίσιμο για υλοποιήσεις σε *FPGA*.

#### LeNet-5

Πρώιμη, εμβληματική αρχιτεκτονική για αναγνώριση ψηφίων [28]. Χαρακτηρίζεται από μικρούς πυρήνες συνελίξεων, ενδιάμεσο *pooling* και μικρούς πλήρως συνδεδεμένους ταξινομητές. *Για FPGA*: Πολύ μικρό αποτύπωμα μνήμης και υπολογισμού, εξαιρετική αφετηρία για MNIST, υψηλή ροϊκότητα (*pipeline*) με χαμηλό *reuse factor*.

#### AlexNet

Επανάφερε το ενδιαφέρον για βαθιά μάθηση στο ImageNet με ReLU, *dropout* και *data augmentation* [26]. Μεγαλύτεροι πυρήνες στις πρώτες στρώσεις και μεγάλοι ταξινομητές στο τέλος. *Για FPGA*: Υπολογιστικά βαρύτερη, απαιτεί προσοχή στη διαχείριση *on-chip* μνήμης και πιθανή αντικατάσταση μεγάλων *fully-connected* με *global average pooling*.

#### VGG

Βαθιά στοίβα  $3 \times 3$  συνελίξεων με απλότητα σχεδίασης [37]. Υψηλή ακρίβεια αλλά μεγάλος αριθμός παραμέτρων. *Για FPGA*: Καλή χωρική κανονικότητα για *dataflow* χαρτογράφηση, αλλά απαιτεί επιθετική κβαντοποίηση και αραίωση για να χωρέσει *on-chip*.

#### Inception (GoogLeNet)

*Inception modules* με παράλληλα μονοπάτια και εκτεταμένη χρήση συνελίξεων  $1 \times 1$  για μείωση διαστασιμότητας. *Για FPGA*: Τα παράλληλα κλαδιά ταιριάζουν σε *task-level* και *pipeline* παραλληλία, οι  $1 \times 1$  μειώνουν το *bandwidth*.

#### ResNet

Υπολειμματικές συνδέσεις (*residual connections*) επιτρέπουν πολύ βαθιά δίκτυα χωρίς εκφυλισμό [21]. *Για FPGA*: Τα *skip connections* εισάγουν επαναχρησιμοποίηση χαρακτηριστικών και ελέγχους δρομολόγησης, αλλά παραμένουν φιλικά σε  $H=1$  *pipelines* με προσεκτικό *buffering*.

#### MobileNet

Διάσπαση συνελίξεων σε *depthwise* και *pointwise* ( $1 \times 1$ ) μειώνει δραστικά πράξεις και

παραμέτρους [23]. Για FPGA: Εξαιρετικά ελκυστικές για *on-chip* τοποθέτηση, μικρές απαιτήσεις μνήμης, υψηλή ρυθμοαπόδοση με απλοποιημένους πολλαπλασιαστές.

**EfficientNet** Συντελεσμένη κλιμάκωση (compound scaling) πλάτους, βάθους και ανάλυσης με συστηματικό τρόπο [40]. Για FPGA: Προσφέρει άξονα *design-space exploration* που ταιριάζει με επιλογές *reuse factor*, *unrolling*, ακρίβειας (fixed-point) και στόχους πόρων.

## 2.2 Αρχιτεκτονική και Λειτουργία των FPGA

Σε αυτήν την ενότητα παρουσιάζεται μια τεχνική επισκόπηση των *Field-Programmable Gate Arrays* (FPGA — προγραμματιζόμενες συστοιχίες πυλών πεδίου), με έμφαση στα εσωτερικά δομικά στοιχεία και την αρχιτεκτονική τους. Στη συνέχεια συζητούνται οι διαφορές μεταξύ FPGA και *Application-Specific Integrated Circuits* (ASICs), καθώς και η σύγκριση των FPGA με *Central Processing Units* (CPU) και *Graphics Processing Units* (GPU), ειδικά στο πλαίσιο της επιτάχυνσης της προγνωστικής λειτουργίας (inference) νευρωνικών δικτύων. Τέλος, παρουσιάζεται η συγκεκριμένη πλατφόρμα FPGA που χρησιμοποιείται στη διπλωματική και συνοψίζονται τα τεχνικά της χαρακτηριστικά.

### 2.2.1 Εισαγωγή στα FPGA

Τα FPGA είναι ολοκληρωμένα κυκλώματα στα οποία η εσωτερική λογική, η δρομολόγηση (*routing*) και η διασύνδεση (*interconnect*) μπορούν να επαναδιαμορφωθούν μετά την κατασκευή, επιτρέποντας την προσαρμογή του υλικού σε διαφορετικές εφαρμογές με την πάροδο του χρόνου [9]. Αυτή η επαναπρογραμματιζόμενη φύση προσφέρει μεγάλη ευελιξία έναντι συσκευών σταθερής λειτουργίας. Το επαναδιαμορφώσιμο τμήμα του FPGA, γνωστό ως *Programmable Logic* (PL), αποτελείται από συστοιχίες *διαμορφώσιμων λογικών μπλοκ* (Configurable Logic Blocks, CLBs), δίκτυα δρομολόγησης και προγραμματιζόμενη διασύνδεση. Ένα ειδικό για τη συσκευή *bitstream* που φορτώνεται στη συσκευή καθορίζει τον τρόπο με τον οποίο η λογική χαρτογραφείται, δρομολογείται και διασυνδέεται. Πολλά σύγχρονα *System-on-Chip FPGA* (SoC FPGA), όπως η οικογένεια *Kria*, συνδυάζουν PL με *Processing System* (PS), επιτρέποντας διαμοιρασμό εργασιών: προεπεξεργασία/έλεγχο στο PS και υπολογιστικά πυρήνες (kernels) στο PL.

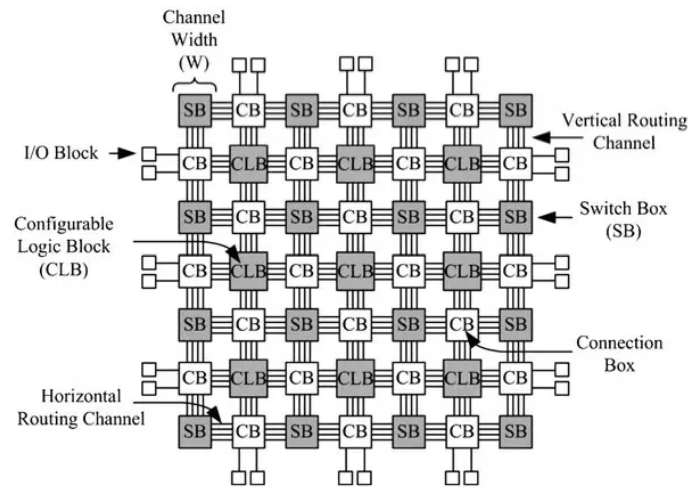
Για να γίνει σαφές πώς επιτυγχάνεται αυτή η ευελιξία, εξετάζουμε τα εσωτερικά δομικά στοιχεία των FPGA και τον τρόπο συνεργασίας τους.

### 2.2.2 Εσωτερική αρχιτεκτονική

Η εσωτερική αρχιτεκτονική των FPGA αποτελείται από ετερογενή δομικά στοιχεία με συμπληρωματικούς ρόλους [9], [27]:

- **Πίνακες αναζήτησης (Look-Up Tables, LUTs):** Υλοποιούν αυθαίρετες Boolean συναρτήσεις μικρού αριθμού εισόδων, αποθηκεύοντας πίνακες αληθείας σε κυψέλες SRAM. Αποτελούν τη βάση της συνδυαστικής λογικής.
- **Flip-flops και καταχωρητές:** Συζευγμένα με LUTs για ακολουθιακή λογική και διατήρηση κατάστασης. Επιτρέπουν διασωλήνωση (*pipelining*) και σύγχρονα ψηφιακά κυκλώματα.
- **Διαμορφώσιμα λογικά μπλοκ (Configurable Logic Blocks, CLBs):** Συστάδες από LUTs/καταχωρητές που αποτελούν τις κύριες προγραμματιζόμενες λογικές μονάδες. Διασυνδέονται για την υλοποίηση πολύπλοκων υποσυστημάτων.
- **Μπλοκ μνήμης (Block RAM, BRAM / UltraRAM):** Ενσωματωμένες μνήμες υψηλού εύρους ζώνης. Διαμορφώνονται ως RAM μονής/διπλής θύρας, FIFO ή ROM. Ορισμένες συσκευές διαθέτουν *UltraRAM* για μεγαλύτερες χωρητικότητες με χαμηλότερο κόστος ανά bit.
- **DSP slices:** Εξειδικευμένες αριθμητικές μονάδες για πράξεις πολλαπλασιασμού–συσσώρευσης (MAC), καίριες για επεξεργασία σήματος και υπολογιστικά φορτία νευρωνικών δικτύων.
- **Προγραμματιζόμενη διασύνδεση (interconnect/routing):** Το δίκτυο που συνδέει LUTs, CLBs, μνήμη και DSPs. Παρέχει ευελιξία, αλλά εισάγει καθυστερήσεις έναντι ASICs. Η επίτευξη *timing closure* επηρεάζεται από βάθος διασωλήνωσης, πυκνότητα δρομολόγησης και κατανομή ρολογιών.
- **Μπλοκ εισόδου/εξόδου (I/O):** Διεπαφές με τον έξω κόσμο: ακροδέκτες γενικής χρήσης, ελεγκτές μνήμης, υψηλού ρυθμού μετατροπείς, και πρότυπα όπως PCIe ή Ethernet.
- **Ενσωματωμένα IP μπλοκ:** Σταθερά υποσυστήματα, όπως ελεγκτές DDR, σκληροί επεξεργαστές και διεπαφές επικοινωνίας, που βελτιώνουν αποδοτικότητα και μειώνουν χρήση PL.

Ο συνδυασμός γενικής ευελιξίας (LUTs/CLBs) με εξειδικευμένη απόδοση (DSP, BRAM/UltraRAM, IP) καθιστά τα FPGA κατάλληλα για επιτάχυνση νευρωνικών δικτύων. Η δομή αυτή ευνοεί *dataflow* χαρτογράφηση με βαθιά διασωλήνωση και ελεγχόμενη επαναχρησιμοποίηση πόρων — κρίσιμη για συνελίξεις και MLPs.



Σχήμα 2.2.1: Βασική δομή ενός FPGA με επισήμανση των θεμελιωδών δομικών στοιχείων.

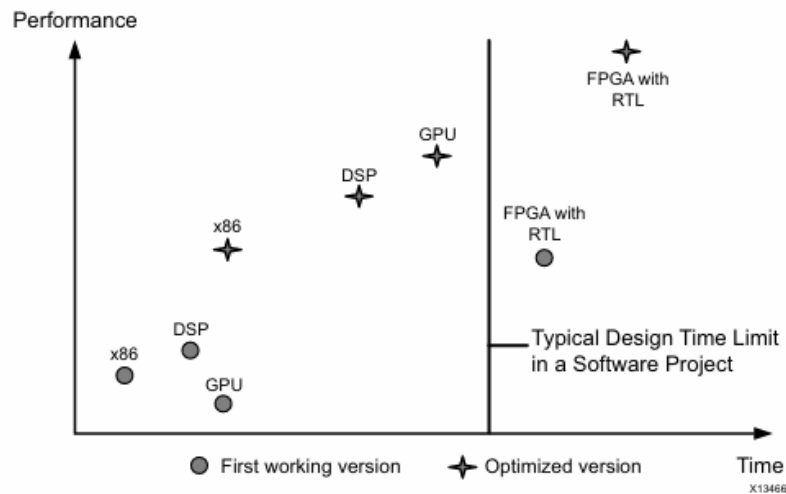
### 2.2.3 FPGA έναντι ASIC

Σε σύγκριση με τα ASICs, τα FPGA προσφέρουν επαναπρογραμματιζόμενη λογική με κόστος σε χώρο, μέγιστη συχνότητα ( $f_{\max}$ ), λανθάνοντα χρόνο (latency) και ενέργεια ανά πράξη. Τα ASICs, πλήρως προσαρμοσμένα σε μία λειτουργία, επιτυγχάνουν μεγαλύτερη πυκνότητα και χαμηλότερη κατανάλωση, αλλά στερούνται ευελιξίας: μόλις κατασκευαστούν, η λειτουργικότητά τους παγιώνεται [27]. Αντίθετα, τα FPGA θυσιάζουν μέρος της «τέλειας» αποδοτικότητας για τη δυνατότητα γρήγορης εξέλιξης/ανάβαθμισης αλγορίθμων και είναι ιδανικά για πρωτότυπα, μεταβαλλόμενα workloads και μικρομεσαίας κλίμακας παραγωγές.

### 2.2.4 FPGA έναντι CPU/GPU για επιτάχυνση νευρωνικών δικτύων

Οι CPU είναι γενικού σκοπού και εύκολες στον προγραμματισμό, αλλά περιορίζονται από τη σχετικά μικρή παραλληλία τους. Οι GPU προσφέρουν μαζικό παραλληλισμό (ιδανικό για πυκνούς πίνακες), όμως η σταθερή τους αρχιτεκτονική και η εξάρτηση από αριθμητική κινητής υποδιαστολής μπορεί να είναι υποβέλτιστες για inference με χαμηλή ακρίβεια ή ροές δεδομένων με αυστηρούς περιορισμούς μνήμης και εύρους ζώνης [45]. Τα FPGA επιτρέπουν λεπτομερή εξειδίκευση υλικού: προσαρμοσμένα πλάτη λέξης (fixed-point), βαθιά pipelines, εκμετάλλευση χωρικού και χρονικού παραλληλισμού και ευθυγράμμιση ροής δεδομένων με την τοπολογία του μοντέλου, εξαλείφοντας instruction overhead [45]. Στο inference, η fixed-point αναπαράσταση και η σωστή διασωλήνωση μειώνουν δραστικά κόστος έναντι FP32 ροών GPU.

<sup>1</sup>Πηγή: <https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls>



Σχήμα 2.2.2: Σύγκριση χρόνου σχεδίασης και απόδοσης μεταξύ x86, DSP, GPU και FPGA.<sup>1</sup>

### 2.2.5 Ροές εργαλείων για FPGA

Η διάδοση των FPGA οφείλεται και στη διαθεσιμότητα ώριμων εργαλείων που μεταφράζουν το σχέδιο σε *bitstream*. Οι δύο κυρίαρχοι κατασκευαστές, AMD (πρώην Xilinx) και Intel (πρώην Altera), προσφέρουν ιδιόκτητα περιβάλλοντα προσαρμοσμένα στις αρχιτεκτονικές τους. Η AMD παρέχει το *Vivado Design Suite* (σχεδίαση RTL/HDL, σύνθεση, τοποθέτηση, δρομολόγηση, IP), ενώ η Intel το *Quartus Prime* με αντίστοιχη λειτουργικότητα και τον *Intel HLS Compiler* (C/C++). Τα εργαλεία αυτά είναι βελτιστοποιημένα για τις συσκευές τους και προσφέρουν ανάλυση χρονισμού και δυνατότητες εντοπισμού σφαλμάτων [45].

Πέρα από τον παραδοσιακό σχεδιασμό RTL, οι σύγχρονες ροές αξιοποιούν *σύνθεση υψηλού επιπέδου* (High-Level Synthesis, HLS) για αύξηση παραγωγικότητας. Το *Vitis HLS* της AMD επιτρέπει περιγραφή επιταχυντών σε C/C++ ή OpenCL και αυτόματη σύνθεση σε RTL. Η αφαίρεση αυτή διευκολύνει προγραμματιστές λογισμικού και σχεδιαστές συστημάτων να στοχεύουν PL, μειώνοντας χρόνο σχεδίασης σε σύνθετες εφαρμογές [9]. Στην παρούσα εργασία, *Vitis HLS* και *hls4ml* χρησιμοποιούνται για τη μεταφορά του CNN στο PL, με κβαντοποίηση *fixed-point* και *pipelining* για χαμηλό λανθάνοντα χρόνο στην πλατφόρμα *Kria KR260*. Συνολικά, οι ροές εργαλείων αποτελούν τον σύνδεσμο μεταξύ περιγραφής λογισμικού και φυσικής υλοποίησης σε FPGA, επηρεάζοντας χρηστικότητα, απόδοση και προσβασιμότητα.

## 2.3 Λειτουργικά Συστήματα και Εργαλεία

Η παρούσα ενότητα συνοψίζει το λειτουργικό περιβάλλον και τα βασικά εργαλεία που χρησιμοποιήθηκαν, καθώς και τις ρυθμίσεις τους. Υπάρχουν πολλαπλές έγκυρες διαμορφώσεις που μπορούν να οδηγήσουν στο ίδιο αποτέλεσμα· ωστόσο, λόγω πρακτικών περιορισμών (διαχείριση πόρων, ευχρηστία, συμβατότητα με ροές FPGA), επιλέχθηκαν συγκεκριμένες λύσεις. Ενδεικτικά, αντί της τυπικής εκτέλεσης *Jupyter notebooks* μέσω φυλλομετρητή, προτιμήθηκε η εκτέλεσή τους μέσα από το *Visual Studio Code (VS Code)* με τις αντίστοιχες επεκτάσεις, ώστε να υπάρχει ενιαίο περιβάλλον ανάπτυξης για host, WSL και απομακρυσμένη συσκευή.

### 2.3.1 Εργαλεία που χρησιμοποιήθηκαν στην εργασία

Στο πλαίσιο της διπλωματικής αξιοποιήθηκαν τρία βασικά εργαλεία για τη ροή HLS→RTL→bitstream: το *Vitis HLS*, το *Vivado* και το *hls4ml*. Στη συνέχεια παρουσιάζονται συνοπτικά ο ρόλος και τα ιδιαίτερα χαρακτηριστικά τους.

#### Vitis HLS

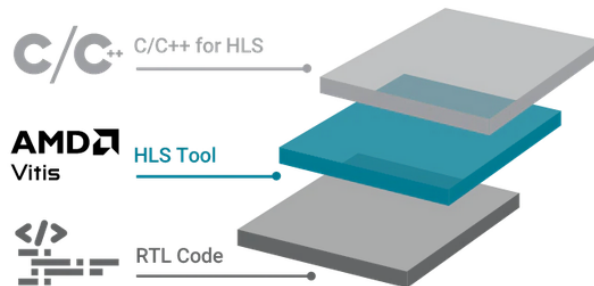
Το *Vitis High-Level Synthesis* είναι εργαλείο σχεδίασης που επιτρέπει τη δημιουργία και υλοποίηση εξειδικευμένων *Intellectual Property (IP)* σε γλώσσες υψηλού επιπέδου όπως C/C++. Παρέχει εργασία σε υψηλό επίπεδο αφαίρεσης χωρίς σημαντικές απώλειες απόδοσης, διευκολύνοντας την ανάπτυξη σύνθετων αλγορίθμων σε FPGA. Προσφέρει *testbenches*, αυτόματες βελτιστοποιήσεις και ενσωμάτωση επιταχυντών υλικού [7].

Η διαδικασία HLS απεικονίζεται στο Σχ. 2.3.1. Αρχικά, ο σχεδιαστής υλοποιεί τον κώδικα σε C/C++. Με *testbenches* επαληθεύεται λειτουργικότητα/ορθότητα και εφαρμόζονται τεχνικές επιτάχυνσης (π.χ. *pipelining*, *loop unrolling*). Ο έλεγχος σε υψηλό επίπεδο είναι απλούστερος σε σχέση με αμιγώς HDL ροές που απαιτούν χαμηλού επιπέδου προσομοιωτές (π.χ. ModelSim).

#### Vivado

Το *Vivado Design Suite* είναι ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment, IDE) της AMD για κώδικα περιγραφής υλικού. Μεταφράζει *Register-Transfer Level (RTL)* (VHDL ή Verilog) σε *FPGA bitstream* και παρέχει αναφορές χρήσης πόρων, χρονισμών και κατανάλωσης. Κεντρικά στάδια: *synthesis*, *placement* και *routing*. Ο σχεδιαστής εξετάζει *trade-offs* απόδοσης, ισχύος και πόρων και εκτελεί *timing*

<sup>2</sup>Πηγή εικόνας: AMD Vitis HLS official page

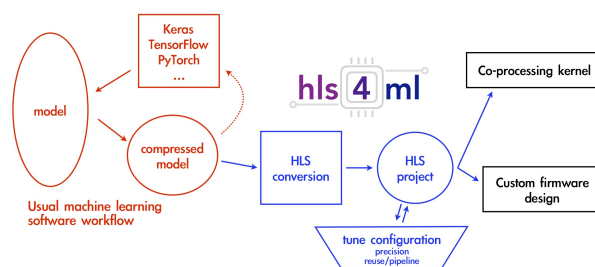


Σχήμα 2.3.1: Απεικόνιση της διαδικασίας σύνθεσης υψηλού επιπέδου (HLS): ο αλγόριθμος περιγράφεται σε C/C++, μετασχηματίζεται από το εργαλείο *AMD Vitis HLS* σε συνθέσιμο RTL κώδικα<sup>2</sup>

*analysis* και *timing closure*. Υποστηρίζονται προσομοιώσεις κυματομορφών και *on-chip debugging* με *Vivado Logic Analyzer*. Μέσω *Vivado IP Integrator* ενσωματώνονται βιβλιοθήκες IP (π.χ. DDR controllers, διεπαφές επικοινωνίας, επεξεργαστές ARM) σε επίπεδο *block diagram*. Συνεργάζεται στενά με το *Vitis* για κλασικές RTL και HLS βασισμένες ροές [7].

### hls4ml

Το *hls4ml* είναι αλυσίδα εργαλείων ανοικτού κώδικα (CERN *Fast Machine Learning*) για υλοποιήσεις βαθιάς μάθησης χαμηλού λανθάνοντος χρόνου σε FPGA. Μετατρέπει μοντέλα *Keras*, *TensorFlow* και *ONNX* σε C/C++ κατάλληλο για HLS και διασυνδέεται με *Vitis HLS*. Η πλήρης αυτόματη δημιουργία *bitstream* υποστηρίζεται επίσημα για συγκεκριμένες πλακέτες και εκδόσεις (π.χ. *Vivado 2020.1*), ενώ πλατφόρμες όπως η *Kria KR260* δεν υποστηρίζονται άμεσα σε αυτό το στάδιο. Οι βασικές δυνατότητες περιλαμβάνουν κβαντοποίηση *fixed-point* και *HLS directives* (*pipelining*, *unrolling*, *reuse factor*). Ο *reuse factor* ελέγχει τον βαθμό σειριοποίησης των πράξεων MAC:  $RF=1$  μεγιστοποιεί τον παραλληλισμό, ενώ μεγάλος  $RF$  μειώνει πόρους με κόστος αυξημένου *latency*.



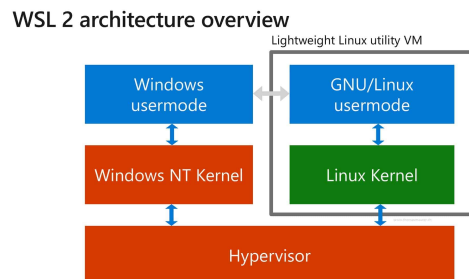
Σχήμα 2.3.2: Διάγραμμα ροής του *hls4ml*<sup>3</sup>

### hls4ml έναντι ανταγωνισμού

Πλαίσια προμηθευτών, όπως **Xilinx Vitis AI** και **FINN**, προσφέρουν βελτιστοποιημένες ροές για κβαντισμένα δίκτυα και κορυφαία απόδοση σε υποστηριζόμενες πλακέτες, με πιο «κλειστές» επιλογές στο επίπεδο HLS. Αντίθετα, το **hls4ml** προσφέρει διαφάνεια και ρητό έλεγχο βελτιστοποιήσεων HLS. Εργαλεία μεταγλώττισης όπως **Apache TVM** και **Intel OpenVINO** δίνουν φορητότητα (CPU/GPU/FPGA) και βελτιστοποιήσεις γραφήματος, αλλά συνήθως δεν παράγουν απευθείας C/C++ φιλικό για HLS, περιορίζοντας τη λεπτομερή χαρτογράφηση πόρων σε FPGA.

### Windows Subsystem for Linux

Το *Windows Subsystem for Linux (WSL)* είναι στρώμα συμβατότητας που επιτρέπει εκτέλεση περιβαλλόντων Linux σε Windows, διαφοροποιούμενο από κλασική εικονικοποίηση (VMs). Στο παρόν έργο χρησιμοποιήθηκε **WSL 2** με **Ubuntu 24**, προσφέροντας ελαφρύ και συμβατό χώρο εργασίας για **hls4ml** και εργαλεία Xilinx σε Linux, χωρίς dual boot. Το WSL 2 επιτρέπει αποδοτική εκτέλεση εργαλείων Linux, γρήγορη εναλλαγή Windows↔Linux και απλουστευμένη κοινή χρήση αρχείων. Η αρχιτεκτονική του WSL 2 αποτυπώνεται στο Σχήμα 2.3.3.



Σχήμα 2.3.3: Οπτικοποίηση της αρχιτεκτονικής του WSL 2<sup>4</sup>

<sup>3</sup>Πηγή: <https://fastmachinelearning.org/hls4ml/intro/introduction.html>

<sup>4</sup>Πηγή: <https://learn.microsoft.com/windows/wsl/compare-versions#wsl-2-architecture>

## Εγκατάσταση WSL 2 σε Windows

Η εγκατάσταση του WSL 2 μπορεί να γίνει από *PowerShell (Run as Administrator)*:

### 1. Ενεργοποίηση WSL:

```
wsl --install
```

Σε Windows 10 ή αν χρειαστεί χειροκίνητα:

```
dism.exe /online /enable-feature
↵ /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all
↵ /norestart
wsl --set-default-version 2
```

### 2. Επανεκκίνηση του συστήματος, αν ζητηθεί.

### 3. Εγκατάσταση **Ubuntu** (π.χ. 22.04/24.04), προαιρετικά:

```
wsl --install -d Ubuntu-24.04
```

### 4. Πρώτη εκκίνηση **Ubuntu** από Start Menu, ορισμός *username/password*.

### 5. Ενημέρωση πακέτων:

```
sudo apt update && sudo apt -y upgrade
```

Προαιρετικά, βασικά εργαλεία:

```
sudo apt -y install build-essential git wget curl unzip ca-certificates
```

## Εγκατάσταση Vivado μέσα από WSL (μέσω τερματικού)

### Σημαντικό:

Η AMD δεν υποστηρίζει επισήμως πλήρως το Vivado μέσα από WSL (ιδίως για άμεση πρόσβαση USB/JTAG). Ωστόσο, λειτουργίες όπως *synthesis*, *analysis* και δημιουργία *bitstream* εκτελούνται κανονικά.

**1) Προετοιμασία αρχείων εγκατάστασης** Κατέβασε τον ενοποιημένο εγκαταστάτη (π.χ. `Xilinx_Unified_20XX.X_..._Linux`) στον Windows host (π.χ. `C:\Users\<User>\Downloads`). Από WSL:

```
cd /mnt/c/Users/<User>/Downloads
mkdir -p ~/xilinx_installer
tar -xf Xilinx_Unified_20*.tar.gz -C ~/xilinx_installer
```

**2) Εγκατάσταση βιβλιοθηκών συμβατότητας με `installLibs.sh`.** Στον φάκελο του εγκαταστάτη υπάρχει script `installLibs.sh` που εγκαθιστά αυτόματα απαιτούμενες βιβλιοθήκες για Ubuntu/WSL:

```
cd ~/xilinx_installer
sudo ./installLibs.sh
sudo apt update && sudo apt -y upgrade
```

**3) Εκτέλεση εγκατάστασης (GUI ή batch).** Σε Windows 11 με WSLg, μπορεί να ανοίξει το *GUI*:

```
cd ~/xilinx_installer
./xsetup
```

Εναλλακτικά *batch/silent*:

```
cd ~/xilinx_installer
sudo ./xsetup --batch Install --agree XilinxEULA,3rdPartyEULA --location
↵ /opt/Xilinx
```

(Προτεινόμενη διαδρομή: `/opt/Xilinx`, απαιτήσεις χώρου:  $\geq 60\text{--}80$  GB.)

**4) Προσθήκη μεταβλητών περιβάλλοντος στο `.bashrc`.**

```
echo 'export XILINX_VIVADO=/opt/Xilinx/Vivado/20XX.X' >> ~/.bashrc
echo 'export XILINX_VITIS_HLS=/opt/Xilinx/Vitis_HLS/20XX.X' >> ~/.bashrc
echo 'export PATH=$PATH:$XILINX_VIVADO/bin:$XILINX_VITIS_HLS/bin' >> ~/.bashrc
source ~/.bashrc
```

Έπειτα:

```
vivado & # Vivado
vitis_hls & # Vitis HLS
```

**5) Προγραμματισμός συσκευής (εναλλακτικές).** Αν η USB/JTAG δεν αναγνωρίζεται στο WSL:

- **HW server στα Windows:** τρέξε `hw_server` στον Windows host και σύνδεσε από Vivado (WSL) μέσω *Open Target* □ *Host: localhost, Port: 3121*.
- **Προγραμματισμός από Windows:** δημιουργία `.bit` στο WSL και φόρτωση από Vivado στα Windows ή μέσω embedded Linux στο board (π.χ. Kria KR260).

*Πρακτική ροή:* Στην παρούσα εργασία, HLS/σύνθεση (C/C++ □ RTL) εκτελούνται στο WSL (*Vitis HLS/hls4ml*) και το Vivado στα Windows χρησιμοποιείται για προγραμματισμό και *on-chip debug*, αποφεύγοντας ζητήματα USB/JTAG στο WSL.

## Βιβλιοθήκες και εργαλεία Python

Συνοψίζονται τα εργαλεία Python που αξιοποιήθηκαν (αναπαραγωγιμότητα/ενσωμάτωση με ροή FPGA):

- **hls4ml** 1.1.0[profiling]: μετατροπή μοντέλων Keras/TensorFlow σε C/C++ κατάλληλο για HLS και διασύνδεση με Vitis HLS (παραγωγή RTL).
- **TensorFlow** 2.14.x (με **Keras**): ορισμός αρχιτεκτονικών και εκπαίδευση μοντέλων.
- **QKeras**: κβαντισμένα επίπεδα και εξαγωγή κβαντισμένων μοντέλων συμβατών με *hls4ml*.
- **matplotlib**: οπτικοποίηση δεδομένων και αποτελεσμάτων.
- **scikit-learn**: μετρικές και βοηθητικά ML εργαλεία.
- **pandas**: πίνακες δεδομένων και καταγραφή πειραμάτων.
- **Miniconda**: διαχείριση περιβαλλόντων/εκδόσεων σε host/WSL (σε συνδυασμό με επέκταση *Python* του VS Code).
- **Επεκτάσεις VS Code**: *Remote Development* (WSL/SSH), *Jupyter*, *Python* — εκτέλεση notebooks εντός VS Code και ομαλή ανάπτυξη σε WSL/απομακρυσμένη Kria.

### 2.3.2 Γλώσσες Περιγραφής Υλικού

#### VHDL

Η VHDL εμφανίστηκε στα μέσα της δεκαετίας του 1980 μέσα από προγράμματα τυποποίησης σχεδίασης ψηφιακών συστημάτων και καθιερώθηκε ως γλώσσα με έμφαση στη σαφήνεια και την αναγνωσιμότητα. Η διάδοσή της ενισχύθηκε από την ακαδημαϊκή κοινότητα και κλάδους όπως η αεροδιαστημική και η άμυνα, όπου η τεκμηρίωση και η αυστηρότητα είναι κρίσιμες. Χρησιμοποιείται διεθνώς για μοντελοποίηση, προσομοίωση και RTL υλοποίηση, με ευρεία υποστήριξη από εργαλεία EDA και ισχυρή παρουσία σε FPGA/ASIC.

#### Verilog

Η Verilog αναπτύχθηκε επίσης στα μέσα της δεκαετίας του 1980 ως εμπορική γλώσσα προσομοίωσης και υλοποίησης ψηφιακών κυκλωμάτων και υιοθετήθηκε εκτενώς από τη βιομηχανία μικροηλεκτρονικής. Η πιο «λιτή», οικεία σύνταξή της συνέβαλε

στη δημοτικότητα της σε ομάδες σχεδίασης chips και ροές μεγάλης κλίμακας. Σήμερα χρησιμοποιείται ευρέως για περιγραφή RTL, επαλήθευση και ενσωμάτωση IP, αποτελώντας βασικό πυλώνα των σύγχρονων ροών σχεδίασης FPGA.

## 2.4 Μετρήσεις και Ορισμοί

Σε αυτή την ενότητα παρουσιάζονται οι μετρικές που χρησιμοποιούνται για την αξιολόγηση των προτεινόμενων υλοποιήσεων *Convolutional Neural Networks (CNN)* σε διαφορετικές πλατφόρμες υλικού. Καλύπτονται τόσο οι μετρικές ποιότητας ταξινόμησης, που περιγράφουν την προγνωστική απόδοση του μοντέλου, όσο και οι μετρικές απόδοσης υλικού, που αποτυπώνουν καθυστέρηση (*latency*), ρυθμοαπόδοση (*throughput*), κατανάλωση ισχύος/ενέργειας και χρήση πόρων FPGA. Στόχος είναι η ύπαρξη ενιαίων, συγκρίσιμων ορισμών για όλα τα πειραματικά σενάρια.

### 2.4.1 Μετρικές Ταξινόμησης

Οι μετρικές ταξινόμησης αποτελούν καθιερωμένο τρόπο αξιολόγησης της ποιότητας προβλέψεων ενός μοντέλου [17], [34]. Βασίζονται στον πίνακα σύγχυσης:

$TP$  : αληθώς θετικά,  $TN$  : αληθώς αρνητικά,  $FP$  : ψευδώς θετικά,  $FN$  : ψευδώς αρνητικά.

Πέρα από τη συνολική ακρίβεια, μετρικές όπως *precision*, *recall* και *F1-score* παρέχουν πιο λεπτομερή εικόνα, ιδίως σε ανισόρροπα σύνολα δεδομένων ή όταν ορισμένοι τύποι σφαλμάτων έχουν μεγαλύτερο βάρος.

- **Accuracy (Ακρίβεια):**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \quad (2.5)$$

Χρήσιμη συνολικά, αλλά μπορεί να είναι παραπλανητική σε μη ισορροπημένες κλάσεις.

- **Precision (Ακρίβεια Θετικών):**

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (2.6)$$

Υψηλή τιμή υποδηλώνει λίγα ψευδώς θετικά.

- **Recall (Ανάκληση):**

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (2.7)$$

Υψηλή τιμή σημαίνει λίγα ψευδώς αρνητικά.

- **F1-score:**

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (2.8)$$

Ο αρμονικός μέσος precision–recall, χρήσιμος σε ανισόρροπα σύνολα δεδομένων.

### ROC Καμπύλη και AUC Μετρική

Εκτός από τις παραπάνω βασικές μετρικές, η απόδοση ενός ταξινομητή μπορεί να εκτιμηθεί μέσω της καμπύλης **ROC (Receiver Operating Characteristic)** και της αντίστοιχης μετρικής **AUC (Area Under Curve)**.

- **ROC Καμπύλη:** Απεικονίζει τη σχέση μεταξύ του *True Positive Rate (TPR)* και του *False Positive Rate (FPR)* για διαφορετικά κατώφλια απόφασης:

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}.$$

Μια ιδανική καμπύλη ROC πλησιάζει την πάνω αριστερή γωνία του γραφήματος, δείχνοντας ότι ο ταξινομητής έχει υψηλή ικανότητα διάκρισης μεταξύ θετικών και αρνητικών κλάσεων.

- **AUC (Area Under the Curve):** Η αριθμητική τιμή του εμβαδού κάτω από την καμπύλη ROC. Κυμαίνεται στο διάστημα  $[0, 1]$ , όπου:

$$AUC = 1 \Rightarrow \text{τέλεια ταξινόμηση}, \quad AUC = 0,5 \Rightarrow \text{τυχαίος ταξινομητής}.$$

Τιμές AUC άνω του 0,95 θεωρούνται εξαιρετικές, δείχνοντας ότι το μοντέλο διατηρεί υψηλή διακριτική ικανότητα ακόμη και μετά από κβάντιση ή βελτιστοποιήσεις υλικού.

#### Σχόλιο:

Οι ROC και AUC μετρικές χρησιμοποιήθηκαν στην παρούσα εργασία (Ενότητα 5.1.2) για τη σύγκριση των εκδόσεων Keras, QKeras και hls4ml, επιβεβαιώνοντας ότι η ροή μετατροπής HLS διατηρεί πλήρως τη διακριτική ικανότητα του αρχικού μοντέλου.

### 2.4.2 Μετρικές απόδοσης

Για τις υλοποιήσεις υλικού, η βασική μετρική είναι η **καθυστέρηση (latency)**, ο χρόνος που απαιτείται για ένα πλήρες inference.

- *End-to-End (E2E) latency*: περιλαμβάνει όλη τη ροή δεδομένων (είσοδο μέσω AXI-Stream/DMA, υπολογισμό, επιστροφή εξόδου). Αντιπροσωπεύει τη συνολική καθυστέρηση που παρατηρεί το σύστημα.
- *Core latency*: μετρά μόνο τον χρόνο εκτέλεσης του νευρωνικού πυρήνα, εξαιρώντας γενικά έξοδα επικοινωνίας.

$$\text{Latency}_{\text{core}} = \frac{\text{cycles}}{f_{\text{clk}}}, \quad f_{\text{clk}} [\text{Hz}]. \quad (2.9)$$

Μία συμπληρωματική μετρική είναι η **ρυθμοαπόδοση (throughput)**, δηλαδή ο αριθμός inferences ανά δευτερόλεπτο. Για σειριακή ροή ( $\text{batch}=1$ ):

$$\text{Throughput} = \frac{1}{\text{Latency}} \left[ \frac{1}{\text{s}} \right]. \quad (2.10)$$

Για αγωγοποιημένες (*pipelined*) ροές:

$$\text{Throughput} \approx \frac{f_{\text{clk}}}{\Pi}, \quad (2.11)$$

όπου *Initiation Interval (II)* είναι οι κύκλοι μεταξύ διαδοχικών εισόδων στον αγωγό. Στις αναφορές δηλώνεται αν πρόκειται για *single-stream* ή *steady-state pipelined* ρυθμοαπόδοση.

### 2.4.3 Ενεργειακές μετρήσεις

Η ενεργειακή απόδοση (*energy efficiency*) είναι κρίσιμη σε πραγματικού χρόνου και ενσωματωμένες εφαρμογές. Εξετάζονται:

- **Ισχύς (Power, W/mW)**: μετράται από αναφορές Vivado μετά την υλοποίηση και *on-board* μέσω αισθητήρων.
- **Αποδοτικότητα (1/J)**:

$$\eta = \frac{\text{Throughput}}{\text{Power}} \left[ \frac{1}{\text{J}} \right], \quad (2.12)$$

ενώ η *ενέργεια ανά inference* είναι  $E_{\text{inf}} = 1/\eta$  [J/inference].

Στην πλατφόρμα Kria KR260, οι *on-chip* μετρήσεις προέρχονται από τον αισθητήρα INA260 [41], ο οποίος μετρά ρεύμα και τάση για άμεσο υπολογισμό ισχύος. Ο αισθητήρας παρακολουθεί την κατανάλωση του υποσυστήματος προγραμματιζόμενης λογικής

(PL) και παρέχει εκτίμηση της *on-chip* ισχύος, χωρίς να περιλαμβάνει το ενεργειακό αποτύπωμα των εσωτερικών *Block RAMs* (BRAMs). Οι μετρήσεις αυτές διασταυρώνονται με τις εκτιμήσεις του *Vivado Power Analyzer* για επικύρωση.

**Σημείωση:** Η ενεργειακή σύγκριση περιορίζεται σε GPU και FPGA. Για CPU, οι εκτιμήσεις ισχύος χωρίς εξωτερικά όργανα είναι λιγότερο αξιόπιστες και χρησιμοποιούνται μόνο ενδεικτικά όπου αναφέρεται ρητά.

#### 2.4.4 Μετρικές χρήσης και επαναχρησιμοποίησης πόρων

Η *χρήση πόρων* (*resource utilization*) αποτυπώνει πόσο «γεμίζει» το λογικό ύφασμα της FPGA και πώς αυτό σχετίζεται με την απόδοση. Αναφέρονται πάντα απόλυτες τιμές και ποσοστά επί του μέγιστου δυνατού της συσκευής, ώστε οι υλοποιήσεις να είναι συγκρίσιμες.

##### Πόροι που αναφέρονται συστηματικά

- **Flip-Flops (FFs):** καταχωρητές κατάστασης που επιτρέπουν διασωλήνωση (*pipelining*). Υψηλή χρήση συνεπάγεται βαθιά αγωγοποίηση και μεγαλύτερη συχνότητα λειτουργίας ( $f_{max}$ ), αλλά αυξημένη πυκνότητα δρομολόγησης.
- **Look-Up Tables (LUTs):** υλοποιούν συνδυαστική λογική. Υπέρμετρη χρήση υποδηλώνει μη ιδανική χαρτογράφηση ή έλλειψη πόρων μνήμης και DSP.
- **Block RAM (BRAM) & UltraRAM (URAM):** τοπικές μνήμες υψηλού εύρους ζώνης. BRAM για μικρότερους buffers/βάρη, URAM για μεγαλύτερα tensors.
- **DSP Slices:** αριθμητικές μονάδες *Multiply-Accumulate* (MAC). Η χρήση τους σχετίζεται με τον βαθμό παραλληλισμού και την ακρίβεια (INT8/INT16/FP32).

##### Δείκτες σχετιζόμενοι με απόδοση

Για τη συσχέτιση πόρων–απόδοσης, αναφέρονται:

- **Initiation Interval (II):** πλήθος κύκλων μεταξύ διαδοχικών εισόδων στον αγωγό.  $II=1$  υποδηλώνει πλήρη ροή (ένα δείγμα/κύκλο).
- **Reuse Factor (RF):** βαθμός σειριοποίησης MACs.  $RF=1$  μεγιστοποιεί παραλληλισμό, μεγαλύτερο  $RF$  μειώνει DSPs αλλά αυξάνει καθυστέρηση.
- **Συχνότητα ρολογιού  $f_{clk}$ :** αναφέρεται η επιτευχθείσα (όχι στόχος) συχνότητα και το *timing slack* (ns), για αξιολόγηση του *timing closure*.

## 2.5 Αρχιτεκτονική SoC FPGA και Πλατφόρμα AMD Kria KR260

Η κατανόηση της αρχιτεκτονικής ενός *System-on-Chip Field-Programmable Gate Array* (SoC FPGA) και των διαθέσιμων πόρων της πλατφόρμας *AMD Kria KR260* είναι θεμελιώδης για την αποδοτική υλοποίηση και χαρτογράφηση συνελκτικών νευρωνικών δικτύων (Convolutional Neural Networks, CNNs) σε υλικό. Η ενότητα αυτή παρουσιάζει τα βασικά υποσυστήματα του *Zynq UltraScale+ MPSoC*, τα πρωτόκολλα διασύνδεσης, την ιεραρχία μνήμης και τους διαθέσιμους πόρους, θέτοντας το θεωρητικό υπόβαθρο για τις επόμενες φάσεις ανάπτυξης.

### 2.5.1 Πλατφόρμα AMD Kria KR260

Η *AMD Kria KR260* αποτελεί ολοκληρωμένο πακέτο ανάπτυξης που συνδυάζει:

1. το *Kria K26 System-on-Module (SoM)*, το οποίο ενσωματώνει το *Zynq UltraScale+ MPSoC (XCK26)*, μνήμη DDR4, τροφοδοσία και ρολόγια, και
2. το *carrier board*, που παρέχει διεπαφές εισόδου/εξόδου (Ethernet, USB, DisplayPort, SFP+, RPi HAT, PMOD κ.ά.).

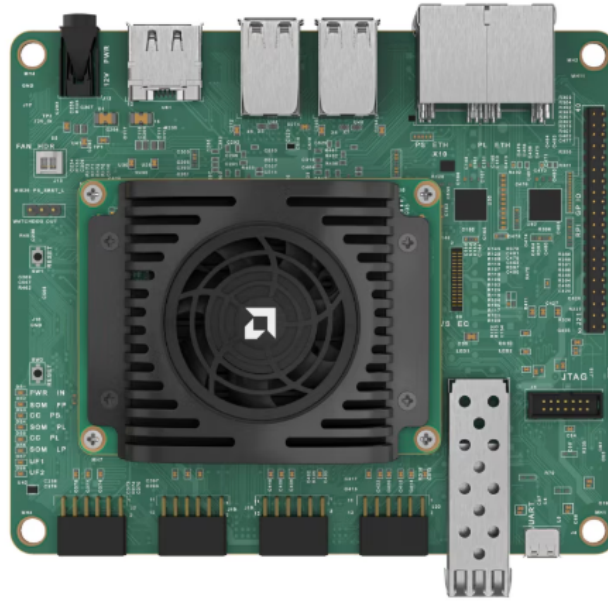
Η διάκριση μεταξύ SoM και carrier board επιτρέπει την ταχεία ανάπτυξη εφαρμογών χωρίς την πολυπλοκότητα σχεδίασης PCB υψηλών ταχυτήτων [4], [5], [47].

**SoC έναντι SoM.** Ένα *System-on-Chip (SoC)* είναι ολοκληρωμένο κύκλωμα που ενσωματώνει στο ίδιο πυρίτιο επεξεργαστές γενικού σκοπού (CPU cores), προγραμματιζόμενη λογική (FPGA fabric) και περιφερειακά υποσυστήματα, παρέχοντας πλήρες υπολογιστικό σύστημα. Αντίθετα, το *System-on-Module (SoM)* συσκευάζει το SoC μαζί με κρίσιμα στοιχεία όπως μνήμη DRAM, αποθήκευση flash, ρυθμιστές ισχύος και διασυνδέσεις I/O σε μία συμπαγή πλακέτα, η οποία τοποθετείται πάνω σε carrier board. Έτσι, το SoM απλοποιεί τη διαδικασία ανάπτυξης, προσφέροντας έτοιμο υπολογιστικό υπόβαθρο και αφήνοντας στο carrier board τη διαχείριση ισχύος και την προσαρμογή στην εκάστοτε εφαρμογή.

### 2.5.2 Processing System (PS) και Programmable Logic (PL)

Το *Processing System (PS)* περιλαμβάνει πυρήνες *Arm Cortex-A53* (για λειτουργικά συστήματα και εφαρμογές) και *Cortex-R5F* (για real-time επεξεργασία), καθώς και

<sup>5</sup>Πηγή εικόνας: <https://www.amd.com/en/products/system-on-modules/kria/k26/kr260-robotics-starter-kit.html>



Σχήμα 2.5.1: Ενδεικτική εικόνα πλατφόρμας KR260, με το SoM τοποθετημένο επάνω στο carrier board.<sup>5</sup>

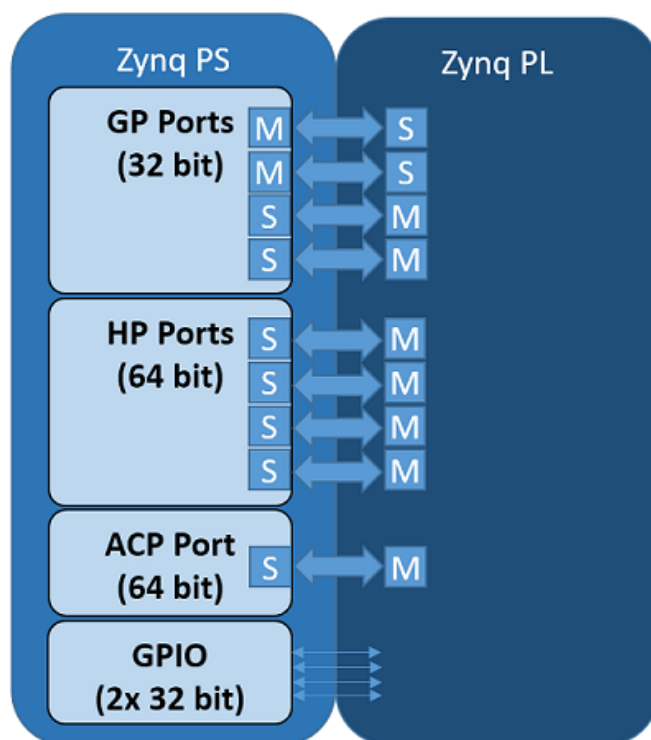
GPU/Video Codec και ελεγκτές DDR και περιφερειακών. Η *Programmable Logic (PL)* αποτελεί τον προγραμματιζόμενο ιστό από LUTs, flip-flops, BRAM, URAM και DSP slices, όπου υλοποιούνται επιταχυντές, όπως τα επιμέρους στρώματα CNN.

Η συνεργασία PS-PL επιτρέπει την αποδοτική κατανομή εργασιών μεταξύ λογισμικού και υλικού, όπως συνοψίζεται στον Πίνακα 2.5.1.

Πίνακας 2.5.1: Κατανομή ρόλων μεταξύ PS και PL στο Zynq UltraScale+ MPSoC.

| Processing System (PS)     | Programmable Logic (PL)   |
|----------------------------|---------------------------|
| Εκτέλεση Linux / εφαρμογών | Υλοποίηση επιταχυντών CNN |
| Διαχείριση DDR buffers     | Παράλληλη εκτέλεση MACs   |
| Έλεγχος DMA και I/O        | Αγωγοποίηση και dataflow  |

<sup>5</sup>Πηγή εικόνας: <https://fastmachinelearning.org/hls4ml/backend/accelerator.html>



Σχήμα 2.5.2: Επισκόπηση θυρών PS–PL σε Zynq UltraScale+: GP (32-bit), HP/HPC (64-bit), ACP.<sup>6</sup>

### 2.5.3 Πρωτόκολλα Διασύνδεσης: AMBA AXI4 και AXI4-Stream

Η επικοινωνία μεταξύ PS και PL βασίζεται στην οικογένεια πρωτοκόλλων *AMBA AXI*, με δύο κύριες μορφές:

- **AXI4 (memory-mapped)**: υποστηρίζει πρόσβαση σε DDR ή περιφερειακά μέσω των θυρών HP/HPC, προσφέροντας υψηλό εύρος ζώνης και υποστήριξη *burst transactions*.
- **AXI4-Stream (dataflow)**: παρέχει ροή δεδομένων χωρίς διευθύνσεις μνήμης, ιδανική για αγωγοποιημένες αρχιτεκτονικές CNN, όπου τα pixels ή τα feature maps μεταδίδονται συνεχώς μέσω των διαύλων.

### 2.5.4 Ιεραρχία Μνήμης και Μετακίνηση Δεδομένων (DMA)

Η αποδοτικότητα ενός CNN accelerator εξαρτάται καθοριστικά από τη ροή δεδομένων και τη μείωση των προσβάσεων στη μνήμη DDR. Κύρια στοιχεία της ιεραρχίας:

- **On-chip μνήμες (BRAM/URAM)**: χρησιμοποιούνται για buffers εισόδου/εξόδου, weights και activations, καθώς και για *line buffers* στις συνελίξεις.
- **DDR μέσω PS**: αποθηκεύει μεγάλα feature maps ή batches που δεν χωρούν on-chip.

- **DMA engines:** μονάδες όπως *AXI DMA*, *AXI CDMA* και *VDMA* επιτρέπουν αποδοτική μεταφορά δεδομένων PS↔PL με ελάχιστο CPU overhead.

Σχεδιαστικός στόχος αποτελεί η παραμονή των κρίσιμων δεδομένων (*weights* και *tiles* ενεργοποιήσεων) στις on-chip μνήμες, ώστε να μειώνεται ο *λανθάνων χρόνος* (*latency*) και η κατανάλωση ισχύος, ενώ η DDR χρησιμοποιείται μόνο για αρχικοποίηση και αποθήκευση των τελικών αποτελεσμάτων.

### 2.5.5 Βασικά Χαρακτηριστικά του K26 SoM

Ενδεικτικά τεχνικά χαρακτηριστικά της προγραμματιζόμενης λογικής του K26 SoM [4], [5] παρουσιάζονται στον Πίνακα 2.5.2.

Πίνακας 2.5.2: Ενδεικτικοί πόροι της PL στο K26 SoM.

| Πόρος                    | Τάξη μεγέθους   |
|--------------------------|-----------------|
| CLB LUTs                 | ~117k           |
| CLB flip-flops           | ~234k           |
| Block RAM (36 Kb blocks) | 144 ( ~5,1 Mb ) |
| UltraRAM blocks          | 64              |
| DSP slices               | 1.248           |
| GTH transceivers         | 4               |

### 2.5.6 Καταλληλότητα της KR260 για CNN Accelerators

Η *Kria KR260* αποτελεί ιδανική επιλογή για υλοποίηση CNN επιταχυντών λόγω των παρακάτω χαρακτηριστικών:

- Συνδυασμός **SoM + carrier board**, που επιτρέπει γρήγορη ανάπτυξη χωρίς περίπλοκο σχεδιασμό PCB.
- **Επαρκείς πόροι PL** (DSP, BRAM, URAM) για μικρού έως μεσαίου μεγέθους CNNs με *fixed-point* κβαντοποίηση.
- Πλούσιες διασυνδέσεις **PS–PL** (HP/HPC/ACP) για αποδοτική μεταφορά δεδομένων.
- Πλήρης συμβατότητα με εργαλεία **HLS** (*Vitis HLS*, *hls4ml*) και τη ροή *Vivado*, επιτρέποντας ταχεία μετάβαση από λογισμικό σε υλικό.

## Σύνοψη

Στο Κεφάλαιο 2 παρουσιάστηκε το θεωρητικό και τεχνολογικό υπόβαθρο που απαιτείται για την ανάπτυξη και υλοποίηση συνελκτικών νευρωνικών δικτύων (CNNs) σε πλατφόρμες τύπου FPGA. Αρχικά αναλύθηκαν οι θεμελιώδεις αρχές λειτουργίας των CNN, τα βασικά τους δομικά στοιχεία (στρώσεις συνελίξεων, συγκέντρωσης, πλήρως συνδεδεμένες, συναρτήσεις ενεργοποίησης) καθώς και οι μετρικές που χρησιμοποιούνται για την αξιολόγηση της ακρίβειας ταξινόμησης. Εξετάστηκαν επίσης τεχνικές συμπίεσης, όπως η κβαντοποίηση και το κλάδεμα, που επιτρέπουν τη μείωση της πολυπλοκότητας των μοντέλων και τη χαρτογράφησή τους σε υλικό με περιορισμένους πόρους.

Παράλληλα, παρουσιάστηκε η αρχιτεκτονική των SoC FPGA με επίκεντρο την πλατφόρμα *AMD Kria KR260*, όπου συνδυάζεται το *Processing System (PS)* με την *Programmable Logic (PL)*. Αναλύθηκαν οι διαθέσιμοι πόροι της συσκευής, τα πρωτόκολλα επικοινωνίας *AXI4* και *AXI4-Stream*, καθώς και η ιεραρχία μνήμης, στοιχεία που καθορίζουν την πρακτική απόδοση και την αποδοτικότητα ενός επιταχυντή CNN.

Συνολικά, το Κεφάλαιο 2 διαμόρφωσε τη θεωρητική και τεχνολογική βάση πάνω στην οποία στηρίζεται η επόμενη φάση της εργασίας, δηλαδή η μεθοδολογία σχεδίασης, μεταφοράς και αξιολόγησης του CNN σε διαφορετικές πλατφόρμες υλικού που αναπτύσσεται στο επόμενο κεφάλαιο.

# Κεφάλαιο 3

## Μεθοδολογία

Σε αυτό το κεφάλαιο παρουσιάζεται η μεθοδολογία που ακολουθήθηκε για τη σχεδίαση, εκπαίδευση και υλοποίηση ενός συνελκτικού νευρωνικού δικτύου (CNN) σε πλατφόρμα FPGA, με χρήση εργαλείων Υψηλού Επιπέδου Σύνθεσης (HLS) και της ροής `hls4ml`. Η ανάλυση περιλαμβάνει τα στάδια από την ανάπτυξη και την εκπαίδευση του μοντέλου σε περιβάλλον Python, έως τη μετατροπή του σε RTL και την ενσωμάτωσή του στο Vivado ως επαναχρησιμοποιήσιμο IP core.

Αρχικά, παρουσιάζονται οι επιλογές σχεδιασμού του μοντέλου, συμπεριλαμβανομένων των χαρακτηριστικών του συνόλου δεδομένων, της αρχιτεκτονικής και των τεχνικών εκπαίδευσης. Στη συνέχεια, αναλύονται βελτιστοποιήσεις φιλικές προς υλικό, όπως η κβάντιση και το κλάδεμα, οι οποίες μειώνουν την κατανάλωση πόρων χωρίς σημαντική υποβάθμιση της ακρίβειας. Ακολουθεί η περιγραφή της ροής HLS, οι παραμετροποιήσεις του `hls4ml` και οι διαφορετικές παραλλαγές υλοποίησης που διερευνήθηκαν. Τέλος, παρουσιάζεται η ανάπτυξη ενός top-level AXI-Stream wrapper, που διασφαλίζει την επικοινωνία με το υπόλοιπο σύστημα και παρέχει μηχανισμούς μέτρησης απόδοσης.

### 3.1 Περίληψη της Προσέγγισης

Σε αυτή την ενότητα παρουσιάζεται συνοπτικά η μεθοδολογία που ακολουθήθηκε για την υλοποίηση και αξιολόγηση ενός συνελκτικού νευρωνικού δικτύου (CNN) σε FPGA, χρησιμοποιώντας μια ροή βασισμένη σε HLS. Η προσέγγιση είναι *end-to-end*, ξεκινώντας από την εκπαίδευση του μοντέλου σε Python και καταλήγοντας στην ανάπτυξη και μέτρηση επιδόσεων στο υλικό.

### 3.1.1 Στρατηγική

Η προτεινόμενη ροή εκκινεί με τη σχεδίαση και εκπαίδευση του μοντέλου σε Python, ακολουθείται από την αυτόματη μετατροπή του σε συμβατό με HLS κώδικα C/C++ και τη σύνθεσή του σε RTL. Το τελικό στάδιο περιλαμβάνει την ενσωμάτωση και ανάπτυξη στην πλατφόρμα FPGA-στόχο.

#### Εκπαίδευση και Κβάντιση

Υιοθετήθηκαν καθιερωμένα πλαίσια Python για τη συγγραφή μοντέλων (TensorFlow, Keras, PyTorch, ONNX υποστηρίζονται από το `hls4ml`), με επιλογή του **Keras** λόγω του ώριμου οικοσυστήματος βελτιστοποίησης και της άμεσης ενσωμάτωσης με το `QKeras`, το οποίο υποστηρίζει σχεδιασμό με επίγνωση κβαντισμού. Μετά τη βασική εκπαίδευση στο MNIST [29], εφαρμόστηκε κβάντιση στα βάρη και τις ενεργοποιήσεις ώστε να αντιστοιχούν σε μορφές σταθερού σημείου κατάλληλες για ανάπτυξη σε FPGA.

### 3.1.2 Γενική Ροή Εργασίας

Το εκπαιδευμένο δίκτυο μετατρέπεται με το `hls4ml` σε περιγραφή C/C++ συμβατή με HLS [14]. Στη συνέχεια, χρησιμοποιείται το **Vitis HLS backend** για τη σύνθεση σε RTL (VHDL/Verilog), με αξιοποίηση οδηγιών όπως *pipelining*, *loop unrolling* και *reuse factor* για την διερεύνηση των συμβιβασμών μεταξύ καθυστέρησης και χρήσης πόρων [6]. Η προσέγγιση αυτή διαφοροποιείται από προηγούμενα παραδείγματα που επικεντρώνονταν στο Vivado 2020.1, καθώς στοχεύει την πλατφόρμα **Kria KR260**, η οποία υποστηρίζεται σε νεότερες εκδόσεις Vivado/Vitis (μετά το 2022), γεγονός που απαιτεί προσαρμογές στη τυπική ροή `hls4ml` [5].

#### System Wrapper και Μετρήσεις

Για την ανάπτυξη ενός επεκτάσιμου επιταχυντή, υλοποιήθηκε ένα ελαφρύ *top-level wrapper*, το οποίο: (i) εκθέτει τις διεπαφές AXI4-Stream και AXI DMA για επικοινωνία CPU-επιταχυντή και (ii) ενσωματώνει μετρητές απόδοσης για μέτρηση καθυστέρησης με ακρίβεια κύκλου [8]. Το wrapper και ο πυρήνας του CNN συν-επαληθεύτηκαν μέσω προσομοιώσεων C (CSIM) και C/RTL (CoSIM). Παράλληλα, επεκτάθηκε η προεπιλεγμένη ροή TCL του `hls4ml`, ώστε να συμπεριλάβει τη νέα οντότητα κορυφαίου επιπέδου και τα testbenches, εξασφαλίζοντας λειτουργικούς και χρονικούς ελέγχους πριν την ενσωμάτωση.

### Ενσωμάτωση στο Vivado

Το συνθετικό IP που παράγεται ενσωματώνεται στο Vivado, όπου συνδέονται οι διασυνδέσεις AXI και το AXI DMA, ενώ ρυθμίζονται οι βασικές παράμετροι συστήματος (π.χ.  $f_{clk}$ ), με αρχική καθοδήγηση από τις αναφορές HLS. Το τελικό αποτέλεσμα είναι ένα *bitstream* έτοιμο για ανάπτυξη στο kit εκκίνησης KR260, επιτρέποντας την από άκρη σε άκρη αξιολόγηση καθυστέρησης και κατανάλωσης ισχύος στο υλικό.

### Σημειώσεις σχετικά με το Πεδίο Εφαρμογής

Η μεθοδολογία εστιάζει στην *χαμηλής καθυστέρησης* εκτέλεση inference με ρεαλιστικές εισόδους/εξόδους. Διακρίνεται η καθυστέρηση *core* από την καθυστέρηση *end-to-end* (συμπεριλαμβανομένων των επιβαρύνσεων DMA και πρωτοκόλλου), σύμφωνα με την κατεύθυνση του `hls4ml` για αποδοτικές και hardware-aware υλοποιήσεις, που παραμένουν εφαρμόσιμες σε ενσωματωμένες ΑΙ εφαρμογές [14].

## 3.2 Σχεδιασμός του Μοντέλου

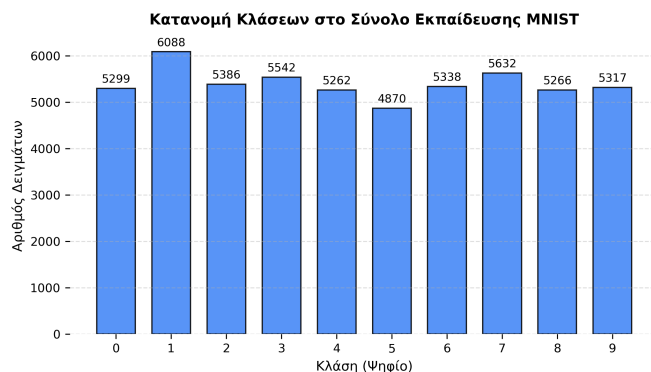
Σε αυτή την ενότητα παρουσιάζονται οι επιλογές σχεδιασμού για το συνελικτικό νευρωνικό δίκτυο (Convolutional Neural Network, CNN), ξεκινώντας από τα χαρακτηριστικά του συνόλου δεδομένων και καταλήγοντας σε βελτιστοποιήσεις με επίγνωση του υλικού, όπως η κβαντοποίηση (quantization) και το κλάδεμα (pruning), οι οποίες καθιστούν το μοντέλο κατάλληλο για υλοποίηση σε FPGA.

### 3.2.1 Σύνολο Δεδομένων

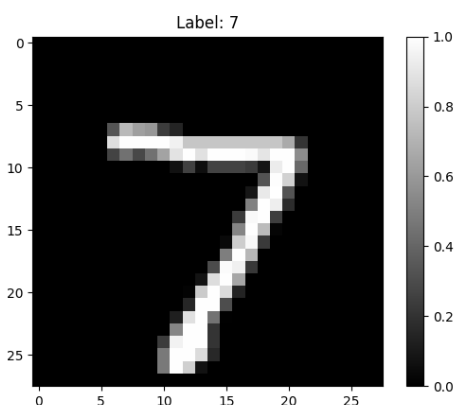
Το επιλεγμένο σύνολο δεδομένων είναι η βάση **MNIST** χειρόγραφων ψηφίων, που εισήγαγαν οι LeCun *et al.* (1998) [29]. Αποτελείται από 70.000 εικόνες σε κλίμακα του γκρι ( $28 \times 28$  pixels, 1 κανάλι) με ψηφία 0–9, κατανεμημένες σε 60.000 δείγματα εκπαίδευσης και 10,000 δείγματα δοκιμής. Η απλότητα του MNIST, όπως φαίνεται στο Σχ. 3.2.2 το καθιστά ένα από τα πιο ευρέως χρησιμοποιούμενα benchmarks για αξιολόγηση αρχιτεκτονικών ταξινόμησης [31].

Η σχετικά χαμηλή πολυπλοκότητα του συνόλου επιτρέπει τη χρήση συμπαγών μοντέλων και ευνοεί επιθετικές βελτιστοποιήσεις φιλικές προς υλικό, όπως μικρούς πυρήνες και αριθμητική σταθερού σημείου (fixed-point). Οι μικρές διαστάσεις εισόδου διευκολύνουν επίσης την παράλληλη χαρτογράφηση σε FPGA.

Η κατανομή των δειγμάτων ανά κατηγορία φαίνεται στο Σχήμα 3.2.1, όπου παρουσιάζεται η συχνότητα εμφάνισης ψηφίων 0–9 στο σύνολο δεδομένων. Η σχεδόν ομοιόμορφη κατανομή επιβεβαιώνει την απουσία σημαντικής ανισορροπίας, διευκολύνοντας την εκπαίδευση και τη σύγκριση μεταξύ παραλλαγών του μοντέλου.



Σχήμα 3.2.1: Κατανομή πλήθους δειγμάτων ανά κλάση στο σύνολο δεδομένων MNIST



Σχήμα 3.2.2: Παράδειγμα εικόνας του MNIST

### 3.2.2 Αρχιτεκτονική

Δεδομένης της απλότητας του MNIST, ακόμη και ρηχά MLP μπορούν να επιτύχουν υψηλή ακρίβεια· ωστόσο, τα CNN προσφέρουν ανθεκτικότερη εξαγωγή χαρακτηριστικών και καλύτερη συμπεριφορά σε παραμορφώσεις εισόδου.

Η τελική αρχιτεκτονική αποτελείται από **δύο συνελκτικά μπλοκ**, καθένα με:

Conv2D → BatchNorm → ReLU → MaxPooling.

- **Conv2D:** πυρήνες  $3 \times 3$ ,  $\text{stride} = 1$ , χωρίς padding (*valid*)· αυτό μειώνει τις χωρικές διαστάσεις κατά 2 pixels ανά διάσταση και απλοποιεί την υλοποίηση σε HLS/FPGA.

- **Batch Normalization (BN):** σταθεροποιεί την εκπαίδευση και βελτιώνει τη συμπεριφορά υπό κβαντοποίηση.
- **MaxPooling:** παράθυρο  $2 \times 2$  με stride 2, υποδιπλασιάζει το spatial μέγεθος και μειώνει πράξεις/μνήμη.

**Εξέλιξη διαστάσεων feature map :**

$$28 \times 28 \times 1 \rightarrow 26 \times 26 \times 8 \rightarrow 13 \times 13 \times 8 \rightarrow 11 \times 11 \times 16 \rightarrow 5 \times 5 \times 16.$$

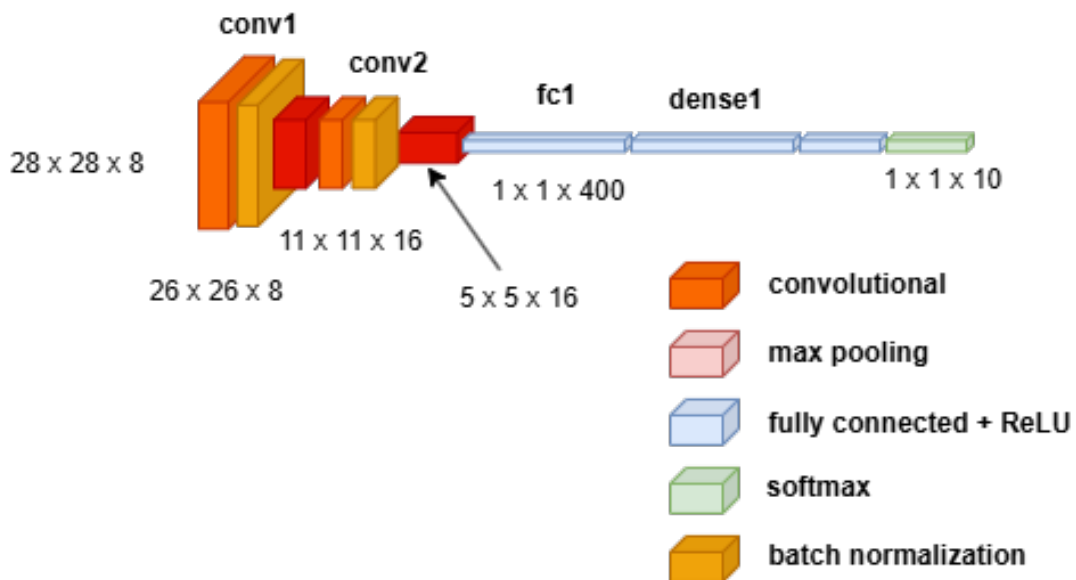
(Πρώτο μπλοκ:  $1 \rightarrow 8$  κανάλια, δεύτερο μπλοκ:  $8 \rightarrow 16$  κανάλια.)

Δοκιμάστηκαν φίλτρα  $(16, 8)$ ,  $(16, 16)$  και  $(8, 16)$ , με το τελευταίο να προσφέρει την καλύτερη ισορροπία ακρίβειας και σταθερότητας υπό κβαντοποίηση.

Μετά τα συνελκτικά μπλοκ, ακολουθεί **Flatten** και δύο **πλήρως συνδεδεμένα (Dense)** στρώματα με 48 και 24 νευρώνες:

Dense  $\rightarrow$  BatchNorm  $\rightarrow$  ReLU.

Τέλος, ένα softmax με 10 εξόδους παράγει πιθανότητες κλάσεων (Σχ. 3.2.3).



Σχήμα 3.2.3: Απεικόνιση της αρχιτεκτονικής του μοντέλου

### 3.2.3 Εκπαίδευση

Η διαδικασία εκπαίδευσης σχεδιάστηκε ώστε να είναι συμβατή με τις ιδιαιτερότητες του MNIST και ανθεκτική σε βελτιστοποιήσεις φιλικές προς υλικό (κβαντοποίηση, κλά-

δεμα). Πριν την εκπαίδευση, τα δεδομένα κανονικοποιούνται σε  $[0, 1]$  και (προαιρετικά) εφαρμόζεται ελαφρά αύξηση δεδομένων (data augmentation) για ανθεκτικότητα.

- **Συνάρτηση σφάλματος (loss function):** *categorical cross-entropy*, καθιερωμένη για πολυταξινομικά προβλήματα με διακριτές κατηγορίες.
- **Βελτιστοποιητής:** *stochastic gradient descent (SGD)* με ρυθμό μάθησης  $3 \times 10^{-3}$ . ο SGD προσφέρει σταθερή συμπεριφορά έναντι πιο «επιθετικών» επιλογών (π.χ. Adam).
- **Μετρική:** *accuracy*, κατάλληλη για ισορροπημένες κλάσεις όπως στο MNIST.
- **Callbacks:**
  1. **EarlyStopping** (*patience*= 10) για πρόληψη υπερπροσαρμογής,
  2. **ReduceLROnPlateau** (μείωση *LR* κατά 0,5 όταν δεν βελτιώνεται το *validation\_loss*),
  3. **UpdatePruningStep** για ομαλή εφαρμογή του χρονοπρογράμματος pruning.
- **Διαδικασία:** Εκπαίδευση για  $n_{\text{epochs}}$  με ξεχωριστά σύνολα train/validation (τυπικά split 90/10). Αποθηκεύεται το καλύτερο μοντέλο (.h5) βάσει *val\_loss*.

**Σημείωση.** Η επιλογή υπερ-παραμέτρων έγινε με βάση την απλότητα του MNIST και την ανάγκη σταθερότητας υπό επιθετική κβαντοποίηση/κλάδεμα.

**Αποθήκευση και φόρτωση.** Το κλαδεμένο/κβαντισμένο μοντέλο αποθηκεύεται ως `pruned_cnn_model.h5` και φορτώνεται με *custom objects* (QKeras quantizers, PruneLowMagnitude), διασφαλίζοντας αναπαραγωγικότητα.

### 3.2.4 Βελτιστοποιήσεις

Η παραπάνω αρχιτεκτονική είναι σημείο εκκίνησης· για υλοποίηση σε FPGA απαιτούνται περαιτέρω βελτιστοποιήσεις με στόχους: (i) περιορισμό χρήσης πόρων (DSPs, LUTs, BRAM) και (ii) διατήρηση ακρίβειας υπό περιορισμένη αριθμητική.

### 3.2.5 Κβαντοποίηση

Η κβαντοποίηση (quantization) μειώνει βάρη/ενεργοποιήσεις από κινητής υποδιαστολής σε σταθερού σημείου, μειώνοντας μνήμη και καθυστέρηση [15], [25]. Δοκιμάστηκαν *post-training quantization (PTQ)* και *quantization-aware training (QAT)* μέσω QKeras [11].

Η προτιμώμενη μορφή ανάπτυξης ήταν  $Q6.10$  (16-bit fixed-point) με ανάλυση  $2^{-10}$ , ενώ σε QAT χρησιμοποιήθηκαν ακόμη χαμηλότερα bitwidths (π.χ.  $Q0.6$ , `quantized_relu(6)`) για να αντικατοπτρίζονται οι περιορισμοί υλικού. Η κβαντοποίηση δεν αλλάζει τον αριθμό MACs, αλλά μειώνει το κόστος υλοποίησης ανά MAC, αξιοποιώντας καλύτερα τα DSP slices της FPGA.

### Κλάδεμα

Εφαρμόστηκε δομημένο κλάδεμα (structured pruning) μηδενίζοντας παραμέτρους μικρού μεγέθους [19]. Το pruning ακολούθησε πολυωνυμικό χρονοδιάγραμμα και οδήγησε σε sparsity  $\sim 50\%$  στο τέλος της εκπαίδευσης. Κλαδεύτηκαν τόσο τα συνελικτικά όσο και τα πλήρως συνδεδεμένα στρώματα (εκτός εξόδου). Το `hls4ml` εκμεταλλεύεται την αραιότητα, παραλείποντας μηδενισμένα βάρη και μειώνοντας LUTs/BRAM χωρίς ουσιαστική απώλεια ακρίβειας.

### 3.2.6 Συνολική Εκτίμηση

Ο συνδυασμός κβαντοποίησης και κλαδέματος διατήρησε ανταγωνιστική ακρίβεια ( $> 99\%$  στο MNIST) και μείωσε αισθητά DSPs/LUTs/μνήμη, ώστε το CNN να χωρά στον προϋπολογισμό πόρων της Kria KR260 και να είναι κατάλληλο για αξιολόγηση σε πραγματικές συνθήκες υλικού.

### 3.2.7 Κατασκευή Βασικού Μοντέλου (Baseline)

Το βασικό CNN υλοποιήθηκε σε Keras μέσω της `build_cnn`. Η συνάρτηση παραμετροποιεί αριθμό φίλτρων συνελίξεων, μεγέθη πυκνών στρωμάτων και διαστάσεις εισόδου, και επιστρέφει *compiled* μοντέλο κατάλληλο για εκπαίδευση και βελτιστοποιήσεις (κβαντοποίηση, κλάδεμα).

```

1 def build_cnn(filter, neurons, input, n_classes=10):
2     x = x_in = Input(input)
3     for i, f in enumerate(filter):
4         print(('Adding convolutional block {} with N={} filters').format(i,
5             ↪ f))
6         x = Conv2D(int(f), kernel_size=(3, 3), strides=(1, 1),
7             ↪ kernel_initializer='lecun_uniform', kernel_regularizer=l1(0.0001),
8             ↪ use_bias=False, name='conv_{}'.format(i),
9             ↪ )(x)
10        x = BatchNormalization(name='bn_conv_{}'.format(i))(x)

```

```

8     x = Activation('relu', name='conv_act_%i' % i)(x)
9     x = MaxPooling2D(pool_size=(2, 2), name='pool_{}'.format(i))(x)
10
11    x = Flatten()(x)
12    for i, n in enumerate(neurons):
13        print(('Adding dense block {} with N={} neurons').format(i, n))
14        x = Dense(n, kernel_initializer='lecun_uniform',
15                ↪ kernel_regularizer=l1(0.0001), name='dense_%i' % i,
16                ↪ use_bias=False
17                )(x)
18        x = BatchNormalization(name='bn_dense_{}'.format(i))(x)
19        x = Activation('relu', name='dense_act_%i' % i)(x)
20
21    x = Dense(int(n_classes), name='output_dense')(x)
22    x_out = Activation('softmax', name='output_softmax')(x)
23
24    model = Model(inputs=[x_in], outputs=[x_out], name='keras_baseline')
25    return model

```

### Δομή κώδικα.

- **Είσοδος & συνελικτικά μπλοκ:** Για κάθε στοιχείο της *filters* δημιουργείται μπλοκ Conv2D→BN→ReLU→MaxPooling2D. Πυρήνες  $3 \times 3$ ,  $L_1$  κανονικοποίηση ( $\lambda = 10^{-4}$ ).
- **Flatten & Dense:** Μετά το *flatten*, προστίθενται Dense μπλοκ (Dense→BN→ReLU) όπως ορίζονται στη *neurons*.
- **Έξοδος:** Dense με  $n_{\text{classes}} = 10$  και softmax.

**Αιτιολόγηση σχεδιασμού.** Αρχικοποίηση *lecun\_uniform*. *use\_bias=False* όπου η BN απορροφά μετατοπίσεις. Η modular δομή επιτρέπει εύκολα πειράματα βάθους/πλάτους.

**Σχετικότητα με υλικό (hardware).** Αποφεύγονται μη υποστηριζόμενες δομές (π.χ. dropout), τα στρώματα είναι καθαρά διαχωρισμένα, και η συστηματική BN διευκολύνει *BN folding* και fixed-point χαρτογράφηση στο *hls4ml*.

### 3.2.8 Κατασκευή Κβαντισμένου Μοντέλου (QKeras)

Για υλοποίηση έτοιμη για FPGA, κατασκευάστηκε κβαντισμένη εκδοχή με QKeras. Η `build_q_cnn` διατηρεί την ίδια μακρο-αρχιτεκτονική, εισάγοντας low-bit quantizers σε βάρη/ενεργοποιήσεις.

```

1 def build_q_cnn(filter, neurons, input, n_classes=10,
  ↳ quantizer_act='quantized_relu(6)',
  ↳ quantizer_dense="quantized_bits(6,0,alpha=1)",
  ↳ quantizer_conv="quantized_bits(6,0,alpha=1)":
2     x = x_in = Input(shape=input)
3     for i, f in enumerate(filter):
4         x = QConv2DBatchnorm(int(f), kernel_size=(3, 3), strides=(1, 1),
  ↳ kernel_quantizer=quantizer_conv, bias_quantizer=quantizer_conv,
  ↳ kernel_initializer='lecun_uniform', kernel_regularizer=l1(0.0001),
  ↳ use_bias=True, name=f'fused_convbn_{i}',
5         )(x)
6         # Quantized ReLU with same format
7         x = QActivation(quantizer_act, name=f'conv_act_{i}')(x)
8         x = MaxPooling2D(pool_size=(2, 2), name=f'pool_{i}')(x)
9
10    x = Flatten()(x)
11    for i, n in enumerate(neurons):
12        x = QDense(n, kernel_quantizer=quantizer_dense, bias_quantizer=None,
  ↳ kernel_initializer='lecun_uniform', kernel_regularizer=l1(0.0001),
  ↳ name=f'dense_{i}', use_bias=False,
13        )(x)
14        x = BatchNormalization(name=f'bn_dense_{i}')(x)
15        x = QActivation(quantizer_act, name=f'dense_act_{i}')(x)
16
17    x = Dense(int(n_classes), name='output_dense')(x)
18    x_out = Activation('softmax', name='output_softmax')(x)
19
20    qmodel = Model(inputs=[x_in], outputs=[x_out], name='qkeras')
21    return qmodel

```

#### Δομή και αιτιολόγηση.

- **Χαμηλής ακρίβειας quantizers:** `quantized_bits(6,0)` (Q0.6) για βάρη· βήμα  $2^{-6} \approx 0,016$ .

- **Περικοπή ενεργοποιήσεων:** `QActivation('quantized_relu(6)')` για 6-bit clamp και σταθερό scaling.
- **BN folding:** `QConv2DBatchnorm` συνενώνει `Conv+BN` ώστε το BN να «διπλώνει» σε scale/shift στο inference.
- **Bias handling:** Biases κβαντίζονται/απορροφώνται από BN όπου δυνατόν, για απλούστερη RTL χαρτογράφηση.

**Επιπτώσεις στο υλικό.** Το QKeras «κλειδώνει» τα bitwidths στο training (QAT), ελαχιστοποιεί το accuracy gap training↔inference, απλοποιεί ρυθμίσεις `hls4ml` και μειώνει πίεση σε DSPs έναντι FP32, με μικρές απώλειες ακρίβειας (βλ. Ενότητα 5.1.2).

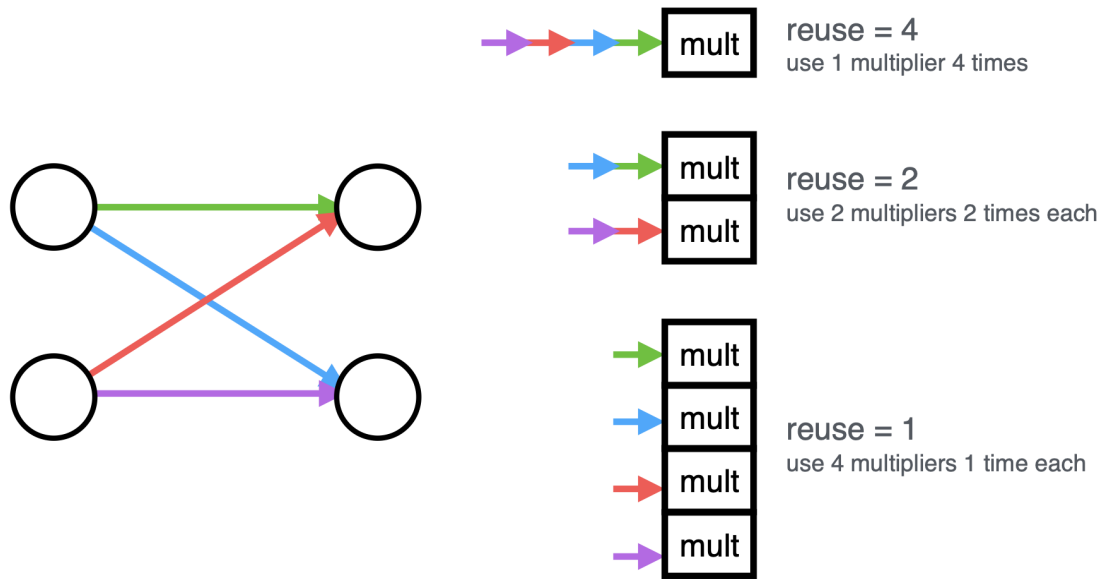
### 3.3 Μετατροπή Μοντέλου και Ροή HLS

Η ενότητα αυτή περιγράφει τη μεταφορά του εκπαιδευμένου μοντέλου από την αναπαράσταση Keras/QKeras σε επίπεδο *Register-Transfer Level (RTL)* μέσω `hls4ml` και *High-Level Synthesis (HLS)*. Παρουσιάζονται οι βασικές παράμετροι παραμετροποίησης, οι παραλλαγές που παράγουν διαφορετικές υλοποιήσεις του ίδιου μοντέλου, και η πλήρης ροή εργαλείων που καταλήγει σε IP έτοιμο για ενσωμάτωση στο Vivado.

#### 3.3.1 Βασική παραμετροποίηση `hls4ml`

Το `hls4ml` προσφέρει ρυθμίσεις που επηρεάζουν άμεσα το trade-off απόδοσης–πόρων:

- **Ακρίβεια (Precision):** σταθερού σημείου (π.χ. Q6.10) ή auto. Οι συσσωρευτές (accumulators) απαιτούν ευρύτερο πλάτος bit για αποφυγή υπερχειλίσσης.
- **Στρατηγική βελτιστοποίησης:** Resource (εξοικονόμηση πόρων) ή Latency (ελαχιστοποίηση καθυστέρησης), με αυτόματη εισαγωγή HLS pragmas στον C/C++ κώδικα.
- **Reuse Factor (RF):** βαθμός επαναχρησιμοποίησης πόρων.  $RF=1$  μεγιστοποιεί τον παραλληλισμό (περισσότερα DSP/LUTs, μικρότερη καθυστέρηση). Μεγαλύτερο RF αυξάνει τη σειριοποίηση (μικρότερη κατανάλωση DSP, μεγαλύτερη καθυστέρηση). Στην πράξη ο RF επιλέγεται ώστε να διαιρεί τον αριθμό MACs του εκάστοτε στρώματος Σχήμα 3.3.1.
- **Τύπος I/O:** `io_stream` για CNNs, με *FIFO* και *line buffers* (shift registers) ώστε να σχηματίζονται παράθυρα συνελίξεων σε ροϊκή (streaming) επεξεργασία.

Σχήμα 3.3.1: Οπτικοποίηση του *reuse factor* στο hls4ml

### 3.3.2 Ανάλυση Κώδικα Υπολογισμού *Reuse Factor*

Ο παρακάτω κώδικας υλοποιεί τη συνάρτηση `reuse_percentage_to_factors`, η οποία παράγει έναν έγκυρο παράγοντα επαναχρησιμοποίησης (*Reuse Factor*, *RF*) για κάθε στρώμα του μοντέλου, με βάση έναν επιθυμητό βαθμό σειριοποίησης (`serial_pct`). Στόχος της συνάρτησης είναι να υπολογίσει για κάθε στρώμα ένα *RF* που εκφράζει τον βαθμό παραλληλισμού ή σειριοποίησης των πράξεων MAC, διατηρώντας παράλληλα τη συμβατότητα με τους περιορισμούς του hls4ml.

```

1 def reuse_percentage_to_factors(model, serial_pct: float = 1.0):
2     if not 0.0 <= serial_pct <= 1.0:
3         raise ValueError("serial_pct must be in [0, 1]")
4     reuse = {}
5
6     def legal_divisor(n, candidate):
7         while candidate > 1 and n % candidate:
8             candidate -= 1
9         return max(1, candidate)
10
11    for layer in model.layers:
12        if isinstance(layer, (Dense, QDense)):
13            total = layer.input_shape[-1] * layer.units      # Nin × Nout
14        elif isinstance(layer, (Conv2D, QConv2D)):
15            kh, kw = layer.kernel_size
16            cin = layer.input_shape[-1]
17            cout = layer.filters
18            total = kh * kw * cin * cout                    # per-pixel MACs
19
20        elif isinstance(layer, (DepthwiseConv2D, QDepthwiseConv2D)):
21            kh, kw = layer.kernel_size
22            cin = layer.input_shape[-1]
23            total = kh * kw * cin                            # per-pixel MACs
24
25        else:
26            continue # no MACs → nothing to tune
27
28        # serial_pct = 1.0 → target_rf = total (most serial)
29        # serial_pct = 0.0 → target_rf = 1 (fully parallel)
30        target_rf = int(round(total * serial_pct))
31        target_rf = min(max(1, target_rf), total)
32
33
34        rf = legal_divisor(total, target_rf)
35
36        reuse[layer.name] = rf
37        print(f"{layer.name:20s} MACs={total:6d} ReuseFactor={rf}")
38
39    return reuse

```

**Λειτουργία.** Η συνάρτηση λαμβάνει δύο παραμέτρους:

- `model`: το εκπαιδευμένο Keras/QKeras μοντέλο,
- `serial_pct`: ποσοστό σειριοποίησης στο διάστημα  $[0, 1]$ , όπου:
  - $s = 0,0$  αντιστοιχεί σε πλήρη παραλληλία ( $RF = 1$ ),
  - $s = 1,0$  σε πλήρη σειριοποίηση ( $RF = \text{total MACs}$ ).

Για κάθε στρώμα του μοντέλου υπολογίζεται ο συνολικός αριθμός πράξεων MAC (`total`), ανάλογα με τον τύπο του:

- **Dense/QDense**:  $\text{total} = \text{Nin} \times \text{Nout}$ , δηλαδή πλήθος πράξεων ανά έξοδο.
- **Conv2D/QConv2D**:  $\text{total} = \text{kh} \times \text{kw} \times \text{Cin} \times \text{Cout}$ , δηλαδή MACs ανά pixel εξόδου.
- **DepthwiseConv2D/QDepthwiseConv2D**:  $\text{total} = \text{kh} \times \text{kw} \times \text{Cin}$ .

Στη συνέχεια υπολογίζεται ο επιθυμητός RF:

$$\text{target\_rf} = \text{round}(s \cdot \text{total}), \quad 1 \leq \text{target\_rf} \leq \text{total}.$$

Επειδή το `target_rf` μπορεί να μην είναι διαιρέτης του `total`, χρησιμοποιείται η βοηθητική συνάρτηση `legal_divisor()`, η οποία μειώνει σταδιακά το `candidate` μέχρι να βρεθεί ένας ακέραιος διαιρέτης του `total`. Έτσι εξασφαλίζεται ότι:

$$RF \mid \text{total\_MACs},$$

επιτρέποντας στο εργαλείο HLS να πραγματοποιεί έγκυρη ανάλυση βρόχων (*loop unrolling*) χωρίς υπολειμματικές πράξεις.

**Επιστρεφόμενα αποτελέσματα.** Για κάθε στρώμα του μοντέλου, η συνάρτηση επιστρέφει ένα λεξικό `reuse{layer.name: RF}`, ενώ παράλληλα εμφανίζει πληροφορίες στο τερματικό:

```
layer_name          MACs=XXXXX  ReuseFactor=YYY
```

**Ερμηνεία του RF.**

- Μικρές τιμές RF ( $\approx 1$ ) σημαίνουν πλήρη παραλληλία: κάθε πράξη MAC εκτελείται ταυτόχρονα, οδηγώντας σε ελάχιστη καθυστέρηση αλλά αυξημένη χρήση DSPs.
- Μεγάλες τιμές RF ( $\approx \text{total MACs}$ ) σημαίνουν πλήρη σειριοποίηση: οι πράξεις εκτελούνται σειριακά, μειώνοντας τη χρήση DSPs εις βάρος της καθυστέρησης.

**Παράδειγμα Χρήσης.** Για  $s \in \{100\%, 50\%, 25\%, 12,5\%, 6,25\%, 0\%\}$ , υλοποιούνται έξι παραλλαγές σχεδίασης:

- $s = 0,0 \Rightarrow RF = 1$  (μέγιστος παραλληλισμός),
- $s = 1,0 \Rightarrow RF = \text{total MACs}$  (πλήρης σειριοποίηση),
- ενδιάμεσες τιμές  $s$  δίνουν αναλογικά επίπεδα παραλληλισμού.

Οι τιμές RF μεταφέρονται στο `hls4ml configuration file` (YAML) για κάθε layer, καθορίζοντας έτσι το επίπεδο αναδίπλωσης (`ReuseFactor`) που εφαρμόζεται κατά τη σύνθεση.

**Σχόλια Υλοποίησης.** Η επιλογή του RF αποτελεί κρίσιμο παράγοντα στην ισορροπία μεταξύ *latency* και *χρήσης πόρων*. Μικρό RF επιτυγχάνει υψηλό throughput αλλά απαιτεί περισσότερους DSPs, ενώ μεγάλο RF εξοικονομεί πόρους εις βάρος της ταχύτητας. Ο κώδικας επιτρέπει τη συστηματική διερεύνηση αυτών των συμβιβασμών, όπως παρουσιάζεται στα αποτελέσματα του Κεφαλαίου 5.

### 3.3.3 Ροή μετάφρασης σε C/C++ και HLS

Η μετάφραση και σύνθεση οργανώνονται μέσω `build_prj.tcl` (Vitis HLS  $\geq 2024.1$ ):

C Simulation  $\rightarrow$  C Synthesis  $\rightarrow$  C/RTL Co-Simulation  $\rightarrow$  Validation.

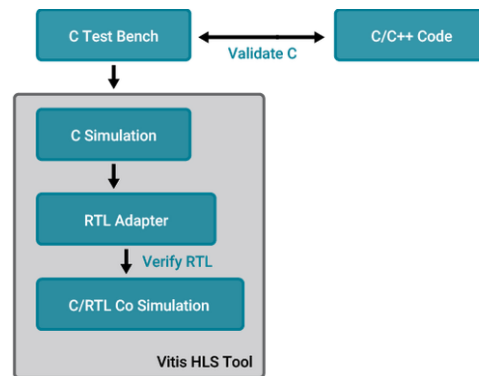
#### Δημιουργία project

Το script δημιουργεί HLS project, ορίζει *top-level* συνάρτηση, προσθέτει πηγές και *testbench*, και καθορίζει *FPGA part* και συχνότητα ρολογιού (`create_clock`). Σε αυτό το στάδιο δηλώνονται πολιτικές (π.χ. partitioning πινάκων, χρήση DSPs) ώστε η σύνθεση να αντανakλά τους περιορισμούς του υλικού.

#### Πλάνο Ελέγχου και Επαλήθευσης

Το πλάνο ελέγχου και επαλήθευσης (*Verification Plan*) περιλαμβάνει όλα τα στάδια δοκιμής που εφαρμόζονται στη ροή Vitis HLS, όπως συνοψίζονται στο Σχήμα 3.3.2. Ο στόχος είναι η επιβεβαίωση της λειτουργικής ορθότητας και της επίδοσης του σχεδίου πριν από την εξαγωγή του σε επίπεδο RTL και την ενσωμάτωσή του στο Vivado.

- **C Simulation (CSIM):** Έλεγχος λειτουργικότητας σε επίπεδο αλγορίθμου, με χρήση κβαντοποιημένων δεδομένων Q6.10. Οι έξοδοι συγκρίνονται με το *golden model* από QKeras, εξασφαλίζοντας ότι η μαθηματική συμπεριφορά είναι ταυτόσημη.

Σχήμα 3.3.2: Επαλήθευση υλοποίησης στο Vitis HLS<sup>1</sup>

- **C Synthesis (CSYN):** Ανάλυση αποδοτικότητας και αναφορά κατανάλωσης πόρων (LUT, FF, DSP, BRAM). Το στάδιο αυτό επαληθεύει την εφικτότητα του σχεδίου και επιβεβαιώνει τη σωστή χαρτογράφηση των υπολογιστικών πράξεων σε υλικό.
- **Co-Simulation (COSIM):** Εκτέλεση κύκλο-προς-κύκλο σύγκρισης μεταξύ C και RTL για επιβεβαίωση ισοδυναμίας. Παράγονται VCD traces, μετρικές καθυστέρησης (latency/II) και επιβεβαίωση ότι οι έξοδοι RTL είναι bit-accurate με το reference.
- **Validation και Αναφορές:** Τα αποτελέσματα των παραπάνω σταδίων συγκεντρώνονται σε ενιαία αναφορά επαλήθευσης, όπου επιβεβαιώνονται οι στόχοι χρονισμού, καθυστέρησης και πόρων πριν την παραγωγή IP.

Η διαδικασία αυτή διασφαλίζει ότι η μετάβαση από το επίπεδο C στο επίπεδο RTL είναι πλήρως επαληθευμένη, ελαχιστοποιώντας τα σφάλματα πριν από την υλοποίηση και επιταχύνοντας το *timing closure* στο Vivado.

### Εξαγωγή IP

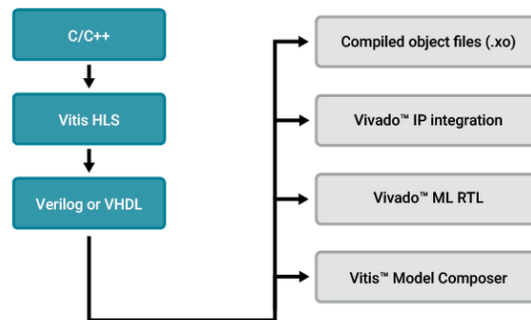
Το RTL design εξάγεται ως IP (export\_design) και ενσωματώνεται στο Vivado, όπου συνδέονται οι διεπαφές AXI4/AXI4-Stream και το AXI DMA. Το αποτέλεσμα είναι *bitstream* έτοιμο για ανάπτυξη όπως φαίνεται στο Σχήμα 3.3.3.

### 3.3.4 Validation και ανάλυση testbench

Για την επαλήθευση της ορθής λειτουργίας και τη γεφύρωση software-hardware χρησιμοποιήθηκε πολυεπίπεδο *testbench*:

<sup>1</sup>Πηγή εικόνας: AMD Vitis HLS official page

<sup>2</sup>Πηγή εικόνας: AMD Vitis HLS official page

Σχήμα 3.3.3: Εξαγωγή IP στο Vitis HLS<sup>2</sup>

1. **Λογισμικού (Python/Keras to HLS):** σύγκριση εξόδων του κβαντισμένου μοντέλου (Python) με το HLS C-model σε κοινά test vectors (MNIST), με μετρικές όπως *max abs diff*, *MSE* και *categorical cross entropy*.
2. **COSIM:** λειτουργικός έλεγχος του HLS C/C++ κώδικα, έγκαιρη ανίχνευση σφαλμάτων ροής δεδομένων.
3. **COSIM (C/RTL):** ισοδυναμία C και RTL, εξαγωγή *latency* (κύκλοι), *initiation interval (II)* και επιβεβαίωση ότι οι *streams/FIFOs* δεν υπο- ή υπερ-πληρώνονται.
4. **On-board (συνοπτικά):** μετά το Vivado, τα ίδια vectors διοχετεύονται μέσω AXI DMA προς τον επιταχυντή. Μετريέται *core latency* (με μετρητές ρολογιού) και *end-to-end latency* (DMA+πρωτόκολλα). Τα αποτελέσματα διασταυρώνονται με COSIM.

Η δομημένη αυτή επαλήθευση διασφαλίζει λειτουργική ισοδυναμία και δίνει μετρήσιμα στοιχεία *latency* και *II* πριν την τελική αξιολόγηση στο υλικό.

### 3.3.5 Σύνθεση με Vivado

Το Vivado εκτελεί *synthesis*, *placement*, *routing* και παράγει ρεαλιστικές αναφορές (*f\_max*, *slack*, χρήση πόρων), γεφυρώνοντας τις εκτιμήσεις HLS με το τελικό hardware. Το παραγόμενο .bit χρησιμοποιείται για προγραμματισμό της πλατφόρμας και για δοκιμές *end-to-end*.

## 3.4 Ανάλυση Κώδικα

Σε αυτή την ενότητα αναλύουμε τον *wrapper* που υλοποιήσαμε γύρω από τα μοντέλα που παράγει το `hls4ml`. Ο *wrapper* ενοποιεί τις παραλλαγές του πυρήνα, χειρίζεται την

επικοινωνία, τροφοδοτεί το AXI4-Stream μέσω FIFO ουρών και ενσωματώνει μετρητές καθυστέρησης.

### 3.4.1 Ανάλυση του AXI-Stream Wrapper

Το παρακάτω *header* ορίζει το ανώτερο HLS wrapper γύρω από τον CNN πυρήνα που παράγεται από το `hls4ml`. Προσαρμόζει το σχέδιο στις διεπαφές AXI4-Stream, εκθέτει προαιρετικούς μετρητές απόδοσης και ορίζει καθαρό όριο ανάμεσα στην *Programmable Logic (PL)* και το υποσύστημα *Processing System (PS)*/AXI DMA.

`doc/code/wrapper_header.tex`

**Κύρια σημεία σχεδίασης.**

- **Πλάτος διαύλου δεδομένων (16 bit).** Ο τύπος `axis_word_t` ορίζει TDATA 16-bit με TKEEP και TLAST ενεργά. Ταιριάζει με το fixed-point format του CNN (Q6.10), ώστε κάθε pixel του MNIST να αντιστοιχεί σε ένα AXI beat.
- **Πλαισίωση και εγκυρότητα.** Το TKEEP δηλώνει ποια bytes είναι έγκυρα στο beat, ενώ το TLAST σηματοδοτεί το τέλος frame (frame boundary), διευκολύνοντας συγχρονισμό με AXI DMA και λογισμικό PS.
- **Σημασιολογία ροής (flow semantics).** Οι `s_axis/m_axis` χαρτογραφούνται σε AXI4-Stream με TVALID/TREADY (handshake). Στο `.cpp` ο πυρήνας είναι πλήρως pipelined με `#pragma HLS DATAFLOW` για II=1.
- **Free-running επιταχυντής.** Η απουσία `ap_ctrl_*` (`ap_ctrl_none`) επιτρέπει συνεχή κατανάλωση/παραγωγή δεδομένων χωρίς ρητά start/stop.
- **Μέτρηση απόδοσης.** Δύο τρόποι: (i) `USE_AXIS_PERF` με τρίτη ροή `m_axis_perf` (stream-based counters), (ii) `ap_none` scalar έξοδοι `cycles_core_out` `cycles_e2e_out`.
- **Ενσωμάτωση στο Vivado.** `s_axis` → AXI DMA MM2S, `m_axis` → AXI DMA S2MM. Οι μετρήσεις μπορούν να οδηγηθούν σε AXI GPIO ή σε monitors για simulation.

**Επιπτώσεις στο υλικό.**

1. **I/O ευθυγραμμισμένο με ακρίβεια.** Το 16-bit stream ταιριάζει με την κβαντισμένη ακρίβεια, αποφεύγοντας επιπλέον *packing/unpacking*.
2. **Ντετερμινιστική μέτρηση.** Οι dual-mode counters λειτουργούν τόσο σε *cosim* όσο και σε *hardware runs*.

3. **Έτοιμο για ενσωμάτωση.** Η σημασιολογία AXI4-Stream και το `ap_ctrl_none` ταιριάζουν στη ροή Kria KR260 με AXI DMA.

**Περίληψη.** Ο wrapper είναι *synthesis-friendly*, κρύβει λεπτομέρειες AXI-Stream και εκθέτει μετρητές, αφήνοντας τον CNN πυρήνα ανέπαφο. Αποτελεί κρίσιμη γέφυρα ανάμεσα στους PYNQ drivers και τον επιταχυντή στην PL.

### 3.4.2 AXI-Stream Wrapper: Δομή και Αιτιολόγηση

Το *top-level* wrapper μετατρέπει τον CNN πυρήνα σε επιταχυντή συμβατό με AXI4-Stream, ενσωματώνοντας μετρητές κύκλων και πλήρως *pipelined* αρχιτεκτονική. Ο σχεδιασμός οργανώνεται σε τέσσερα στάδια (*ingest*, *core*, *egress*, *counter*) που εκτελούνται ταυτόχρονα με DATAFLOW. Έτσι, όσο εξάγονται τα αποτελέσματα ενός frame, το επόμενο ήδη εισάγεται (*overlapped execution*).

#### Προκαταρκτικά και γενικές παράμετροι

- **Frame sizes.** MNIST: `N_INPUT=784 pixels`, `N_OUTPUT=10 κλάσεις`, σταθερές που απλοποιούν τους βρόχους και την HLS ανάλυση.
- **Χρονισμός.** Ο μετρητής ξεκινά στο πρώτο input beat (συνεπής ορισμός *end-to-end*).

```
#include "firmware/ project_name _stream.h"    // axis_word_t + top prototype
#define N_INPUT    784    // 28x28
#define N_OUTPUT    10    // 10 classes
#define START_ON_LAST_IN    0    // start on first beat (0)    TLAST (1)
```

#### Στάδιο A: Ingest (AXIS to Pixels)

Κάθε pixel (Q6.10) αντιστοιχεί σε ένα AXI beat και μετατρέπεται σε τύπο `input_t`. Παράγεται παλμός *start* στην αρχή frame. Μια FIFO ανάμεσα σε *ingest* και *core* εξομαλύνει διακυμάνσεις ρυθμού.

```
static void ingest_pixels(hls::stream<axis_word_t> &s_axis,
                        hls::stream<input_t> &img_fifo,
                        hls::stream<ap_uint<1>> &start_evt_s)
```

```
#pragma HLS INLINE off
READ_PIXELS: for (unsigned i = 0; i < N_INPUT; ++i)
#pragma HLS PIPELINE II=1
axis_word_t w = s_axis.read();
if (START_ON_LAST_IN) if (w.last) start_evt_s.write(1);
else if (i == 0) start_evt_s.write(1);
input_t pix; pix[0].range(15,0) = w.data; img_fifo.write(pix);
```

### Στάδιο B: Core (Inference)

Καλεί τον CNN πυρήνα που παράγεται από hls4ml και σηματοδοτεί *first-availability event* όταν εμφανιστεί η πρώτη έξοδος, επιτρέποντας διαχωρισμό *core* από *downstream* καθυστερήσεις.

```
static void run_core(hls::stream<input_t> &img_fifo,
                    hls::stream<result_t> &res_fifo,
                    hls::stream<ap_uint<1>> &done_core_evt_s)
```

```
#pragma HLS INLINE off
hls::stream<result_t> core_out("core_out");
#pragma HLS STREAM variable=core_out depth=2
#pragma HLS DATAFLOW
project_name (img_fifo, core_out);
result_t logits = core_out.read();
done_core_evt_s.write(1);
res_fifo.write(logits);
```

### Στάδιο Γ: Egress (Logits to AXIS)

Μετατρέπει τα *logits* σε AXI4-Stream. Παράγει 10 beats των 16-bit, με TKEEP για εγκυρότητα και TLAST στο τελευταίο beat. Στο τέλος γράφεται παλμός *done*.

```
static void egress_logits(hls::stream<result_t> &res_fifo,
                          hls::stream<axis_word_t> &m_axis,
                          hls::stream<ap_uint<1>> &done_e2e_evt_s)
```

```
#pragma HLS INLINE off
result_t logits = res_fifo.read();
WRITE_LOGITS: for (unsigned c = 0; c < N_OUTPUT; ++c)
#pragma HLS PIPELINE II=1
axis_word_t w; w.data = logits[c].range(15,0); w.keep = 0x3;
w.last = (c == N_OUTPUT-1); m_axis.write(w);
if (w.last) done_e2e_evt_s.write(1);
```

### Μετρητής κύκλων (cycle counter)

Μετρά *core* και *end-to-end* καθυστέρηση, συγχρονισμένος με *start/done-core/done-e2e*.

```
static void cycle_counter_proc(hls::stream<ap_uint<1>> &start_evt_s,
                              hls::stream<ap_uint<1>> &done_core_evt_s,
                              hls::stream<ap_uint<1>> &done_e2e_evt_s,
                              ap_uint<32> &cycles_core_out,
                              ap_uint<32> &cycles_e2e_out)

#pragma HLS INLINE off
ap_uint<1> dummy; start_evt_s.read(dummy);
while (!done_core_evt_s.empty()) (void)done_core_evt_s.read();
while (!done_e2e_evt_s.empty()) (void)done_e2e_evt_s.read();
ap_uint<32> cnt_core=0, cnt_e2e=0; bool run_core=true, run_e2e=true;
COUNT_LOOP: while (run_core || run_e2e)
#pragma HLS PIPELINE II=1
if (run_core && !done_core_evt_s.empty()) (void)done_core_evt_s.read();
run_core=false;
if (run_e2e && !done_e2e_evt_s.empty()) (void)done_e2e_evt_s.read();
run_e2e=false;
if (run_core) ++cnt_core; if (run_e2e) ++cnt_e2e;

cycles_core_out = cnt_core; cycles_e2e_out = cnt_e2e;
```

### 3.4.3 Ανάλυση Testbench

Ο testbench υλοποιεί λειτουργική επαλήθευση και συλλογή μετρήσεων καθυστέρησης. Εισάγει MNIST διανύσματα μέσω AXI4-Stream, λαμβάνει 10 εξόδους και γράφει αρχεία καταγραφής συμβατά με CSIM/COSIM.

#### Δομή και ροή

- **Αρχεία.** `tb_input_features.dat` (784 τιμές/γραμμή) και `tb_output_predictions.dat` (10 τιμές/γραμμή). Έξοδος: `csim_results.log` ή `rtl_cosim_results.log`.
- **Μετατροπές.** `float_to_axis/axis_to_float` χαρτογραφούν `float`  $\square$  `Q6.10`; `TKEEP=0x3`, `TLAST` στο τελευταίο beat.
- **Latency API.** Είτε side-channel ροή `m_axis_perf` (`USE_AXIS_PERF`) είτε scalar έξοδοι `ap_uint<32>`.

## Σύνοψη

Στο Κεφάλαιο 3 παρουσιάστηκε μια πλήρης, αναπαραγώγιμη ροή για τη μετάβαση ενός CNN από την εκπαίδευση σε Python έως την υλοποίηση σε FPGA. Αρχικά σχεδιάστηκαν και εκπαιδεύτηκαν *baseline* και κβαντισμένες εκδόσεις (Keras/QKeras), εφαρμόστηκαν βελτιστοποιήσεις φιλικές προς υλικό (κβάντιση, κλάδεμα) και αξιολογήθηκαν ως προς ακρίβεια και χρήση πόρων. Στη συνέχεια, τα μοντέλα μετατράπηκαν με `hls4ml` σε περιγραφές C/C++, και συντέθηκαν σε RTL με *Vitis HLS*, διερευνώντας συστηματικά παραμέτρους όπως *reuse factor*, *pipelining* και *DATAFLOW*.

Η ροή αυτοματοποιήθηκε μέσω του `build_prj.tcl` (CSIM, CSYN, COSIM, Validation), μειώνοντας χειροκίνητα βήματα και εξασφαλίζοντας συνεπή αποτελέσματα. Παράλληλα, ο *AXI4-Stream wrapper* ενσωμάτωσε μετρητές κύκλων για *core* και *end-to-end* καθυστέρηση και προσαρμόστηκε στην στοχευόμενη πλατφόρμα (DMA, PS/PL), επιτρέποντας τη δημιουργία *bitstream* και την ομαλή ενσωμάτωση στο Vivado.

Συνολικά, το κεφάλαιο κατέδειξε μια συνεκτική και αποδοτική *end-to-end* μεθοδολογία, η οποία προετοιμάζει το έδαφος για την ενσωμάτωση στο σύστημα και την πειραματική αξιολόγηση στο υλικό που ακολουθούν στα επόμενα κεφάλαια.

## Κεφάλαιο 4

# Πλατφόρμα, Υλοποίηση και Ανάπτυξη

Σε αυτό το κεφάλαιο παρουσιάζεται η πλήρης υλοποίηση του προτεινόμενου επιταχυντή CNN στην πλατφόρμα **AMD Kria KR260**. Η ροή σχεδίασης εκκινεί με το στάδιο της Υψηλού Επιπέδου Σύνθεσης (HLS) του μοντέλου CNN, τη μετατροπή του σε RTL, και την ενσωμάτωσή του στο Vivado ως custom IP core. Στη συνέχεια περιγράφεται λεπτομερώς ο σχεδιασμός σε επίπεδο συστήματος, ο οποίος περιλαμβάνει AXI interconnects, DMA engines και GPIO περιφερειακά, ώστε να διασφαλιστεί η ορθή λειτουργία στο Zynq UltraScale+ MPSoC.

Έπειτα από τον σχεδιασμό υλικού, παρουσιάζεται η ροή ανάπτυξης μέσω του PYNQ framework και ενός custom Python driver, επιτρέποντας την απρόσκοπτη εκτέλεση του επιταχυντή απευθείας στην FPGA πλακέτα. Για την καθιέρωση συγκρίσιμων σημείων αναφοράς, το ίδιο μοντέλο CNN εκτελείται επίσης σε πλατφόρμες CPU και GPU υπό ενοποιημένες και συγκρίσιμες συνθήκες runtime.

Επιπλέον, σε αυτό το κεφάλαιο περιγράφεται η μεθοδολογία επικύρωσης που υιοθετήθηκε ώστε να διασφαλιστεί η λειτουργική ορθότητα, η συνέπεια απόδοσης και η αναπαραγωγιμότητα των αποτελεσμάτων. Τέλος, παρουσιάζεται η υποδομή παρακολούθησης ισχύος, η οποία συνδυάζει τους ενσωματωμένους αισθητήρες INA260 και εργαλεία στην πλευρά του host για την παροχή ακριβών μετρήσεων ενέργειας.

### 4.1 Εισαγωγή

Το κεφάλαιο αυτό εστιάζει στην πρακτική υλοποίηση του προτεινόμενου επιταχυντή CNN, καλύπτοντας τις διαδικασίες σε επίπεδο υλικού και λογισμικού, καθώς και τις βασικές πλατφόρμες αναφοράς για συγκρίσεις.

Στην **πλευρά του υλικού**, το εκπαιδευμένο και κβαντοποιημένο CNN μετατράπηκε σε συνθέσιμη RTL περιγραφή μέσω του `hls4ml` και του **Vitis HLS**. Εφαρμόστηκαν τεχνικές βελτιστοποίησης όπως *pipelining*, *loop unrolling* και ρύθμιση του *reuse factor*,

με στόχο τη διερεύνηση των συμβιβασμών μεταξύ καθυστέρησης και χρήσης πόρων. Ο παραγόμενος πυρήνας IP ενσωματώθηκε στο **Vivado Design Suite** ως μέρος ενός block design με AXI διασυνδέσεις, DMA κινητήρες και GPIO περιφερειακά, επιτρέποντας την επικοινωνία μεταξύ του *Processing System (PS)* και της *Programmable Logic (PL)* του *Zynq UltraScale+ MPSoC* στην πλατφόρμα **Kria KR260**.

Στην **πλευρά του λογισμικού**, οι πυρήνες *Arm Cortex-A53* εκτέλεσαν τον έλεγχο του συστήματος μέσω του **PYNQ framework**. Ένας προσαρμοσμένος *DMA driver* σε Python διαχειρίστηκε τις μεταφορές δεδομένων DDR–επιταχυντή, ενώ τα GPIO interfaces χρησιμοποιήθηκαν για μετρήσεις απόδοσης με ακρίβεια κύκλου. Με αυτόν τον τρόπο κατέστη δυνατή η end-to-end αξιολόγηση της καθυστέρησης inference, του ρυθμού απόδοσης (throughput) και της κατανάλωσης ενέργειας απευθείας στο υλικό.

Για λόγους σύγκρισης, το ίδιο CNN εκτελέστηκε επίσης σε CPU και GPU, παρέχοντας baseline υλοποιήσεις. Οι μετρήσεις αυτές λειτουργούν ως σημείο αναφοράς και αναδεικνύουν τους συμβιβασμούς ανάμεσα σε καθυστέρηση, αξιοποίηση πόρων και ενεργειακή αποδοτικότητα σε ετερογενείς πλατφόρμες.

Η υπόλοιπη ενότητα οργανώνεται ως εξής: Η Ενότητα 4.2 αναλύει τη ροή υλοποίησης στο FPGA, από τις ρυθμίσεις σύνθεσης έως την ολοκλήρωση στο Vivado. Η Ενότητα 4.5 παρουσιάζει τις υλοποιήσεις αναφοράς σε CPU και GPU. Η Ενότητα 4.6 περιγράφει τη διαδικασία επαλήθευσης και τις μετρικές αξιολόγησης. Τέλος, η Ενότητα 4.7.4 συνοψίζει τα συμπεράσματα και τα βασικά ευρήματα της υλοποίησης.

## 4.2 Υλοποίηση σε FPGA

Μετά τη δημιουργία του HLS πυρήνα, το επόμενο στάδιο περιλαμβάνει την ενσωμάτωσή του σε πλήρες σύστημα στο **Vivado Design Suite**, ώστε να παραχθεί το *bitstream* που διαμορφώνει την *Programmable Logic (PL)*. Αν και η διαδικασία αυτή μπορεί να αυτοματοποιηθεί πλήρως μέσω Tcl scripts, στην παρούσα εργασία επιλέχθηκε το γραφικό περιβάλλον για μεγαλύτερο έλεγχο στις αναφορές χρονισμού και χρήσης πόρων. Η περαιτέρω αυτοματοποίηση παραμένει αντικείμενο μελλοντικής βελτίωσης.

### 4.2.1 Εισαγωγή και Συσκευασία IP

Ο πυρήνας CNN που παράγεται από το *Vitis HLS* εξάγεται ως IP και ενσωματώνεται στο Vivado μέσω του packaging flow. Με αυτόν τον τρόπο, εξασφαλίζεται ότι οι εκτιμήσεις HLS για καθυστέρηση και χρήση πόρων επαληθεύονται στο περιβάλλον υλοποίησης του Vivado, όπου εφαρμόζονται επιπλέον βελτιστοποιήσεις και συνδέσεις σε επίπεδο συστήματος.

### 4.2.2 Σχεδίαση Block Design

Η ολοκληρωμένη σχεδίαση υλοποιείται ως block design, το οποίο περιλαμβάνει:

- **Zynq UltraScale+ MPSoC (PS)** με DDR controller, πυρήνες *Arm Cortex-A53* και πλήρες υποσύστημα AXI interconnect.
- **AXI SmartConnect**, για δρομολόγηση συναλλαγών μεταξύ PS και περιφερειακών, παρέχοντας αποδοτικότερη εσωτερική αρχιτεκτονική από τον κλασικό AXI Interconnect.
- **AXI DMA**, υπεύθυνο για τις μεταφορές δεδομένων μεταξύ DDR και του CNN core μέσω AXI4-Stream καναλιών.
- **AXI GPIO**, για την ανάγνωση των μετρητών απόδοσης (`cycles_core`, `cycles_e2e`) και για debugging σήματα προς το PS.
- **Custom CNN IP**, το οποίο ενσωματώνει τις διεπαφές AXI4-Stream για δεδομένα, AXI-Lite για έλεγχο, καθώς και custom εξόδους για μετρήσεις καθυστέρησης.

### 4.2.3 Ρυθμίσεις Παραμέτρων

Η σωστή παραμετροποίηση των IP blocks είναι κρίσιμη για την επίτευξη λειτουργικού και αποδοτικού bitstream:

- **PS configuration:** Ενεργοποιήθηκαν οι θύρες AXI HP για DMA μεταφορές και το ρολόι *PL0* ρυθμίστηκε στα 150 MHz, ώστε να ευθυγραμμίζεται με τις εκτιμήσεις HLS.
- **AXI DMA:** Ορίστηκε σε 16-bit data path, σε συμφωνία με το fixed-point format *Q6.10*, διασφαλίζοντας συνέπεια μεταξύ αριθμητικής αναπαράστασης και υλικής υλοποίησης.
- **GPIO:** Δύο instances χρησιμοποιήθηκαν για debugging και για την απευθείας ανάγνωση μετρήσεων από το PS μέσω Python drivers.

### 4.2.4 Wrappers και Templates

Για τη διασύνδεση του auto-generated κώδικα `hls4ml` με το Vivado, αναπτύχθηκαν προσαρμοσμένα Jinja2 templates. Αυτά ορίζουν τη δομή των αρχείων wrapper (`.h`, `.cpp`) και ενσωματώνουν:

- διεπαφές **AXI4-Stream** και **AXI-Lite**,
- μετρητές κύκλων για *core latency* και *end-to-end latency*,
- και ένα πλήρες **testbench** που εκτελεί inference σε κβαντοποιημένα δεδομένα MNIST και συγκρίνει με golden outputs.

Η γενίκευση των templates επιτρέπει την αυτόματη δημιουργία wrapper για διαφορετικές εκδόσεις του δικτύου (π.χ. μεταβαλλόμενο *reuse factor*) με ελάχιστη χειροκίνητη παρέμβαση.

#### 4.2.5 Σενάρια Αυτοματοποίησης και Ανάλυσης

Η ροή υποστηρίζεται από Python scripts που:

- δημιουργούν κβαντοποιημένα test vectors και golden predictions μέσω QKeras,
- εκτελούν αυτόματα τις φάσεις CSIM, COSIM και σύνθεσης μέσω `build_prj.tcl`,
- και υπολογίζουν μετρικές ακρίβειας (accuracy, MSE, confusion matrix) από τις εξόδους FPGA.

Το τελικό bitstream και τα αρχεία `.hwh/.tcl` φορτώνονται στο PYNQ ως Overlay, επιτρέποντας δυναμική επαναπρογραμματιζόμενη εκτέλεση του CNN μέσα από Python notebooks.

#### 4.2.6 Σύνοψη

Η ροή υλοποίησης σε FPGA ολοκληρώνει την ενσωμάτωση του HLS-generated CNN με DMA και περιφερειακά του PS στο Vivado. Το τελικό design εκθέτει πλήρεις διεπαφές AXI4-Stream, παρέχει μετρήσεις ακριβείας κύκλου και είναι έτοιμο για ανάπτυξη στην πλατφόρμα *Kria KR260*. Η ροή αυτή προσφέρει ρεαλιστικό περιβάλλον αξιολόγησης και αποτελεί το θεμέλιο για τις επιτόπιες (on-board) μετρήσεις που παρουσιάζονται στο επόμενο κεφάλαιο.

### 4.3 Ενσωμάτωση και Παραμετροποίηση Συστήματος στο Vivado

Μετά το στάδιο HLS, το επόμενο βήμα είναι η χρήση του **Vivado**, καθώς το *bitstream* που προγραμματίζει την *Programmable Logic (PL)* μπορεί να παραχθεί μόνο μέσα από

αυτό το περιβάλλον. Αν και η ροή μπορεί να αυτοματοποιηθεί πλήρως μέσω Tcl scripts, στην παρούσα εργασία προτιμήθηκε η χρήση του γραφικού περιβάλλοντος του Vivado, ώστε να υπάρχει άμεσος έλεγχος των αναφορών χρονισμού και της χρήσης πόρων. Η αυτοματοποίηση αυτού του βήματος θεωρείται μελλοντική βελτίωση.

### 4.3.1 Εισαγωγή IP

Αφού εξαχθεί ο πυρήνας IP από το *Vitis HLS*, εντοπίζεται στον φάκελο `prj_folder/solution1/impl/ip/`. Η διαδικασία εισαγωγής στο Vivado ακολουθεί τα εξής βήματα: Tools → Create and Package New IP → Package a specified directory, επιλέγοντας τον φάκελο που περιέχει το παραγόμενο IP. Ο οδηγός ολοκληρώνει το πακετάρισμα, καθιστώντας το IP διαθέσιμο στη βιβλιοθήκη για ενσωμάτωση στο block design.

Η ροή αυτή εξασφαλίζει ότι οι εκτιμήσεις χρονισμού που λαμβάνονται από το *Vitis HLS* προσεγγίζουν καλύτερα τα πραγματικά αποτελέσματα, καθώς το Vivado μπορεί να εφαρμόσει επιπλέον βελτιστοποιήσεις κατά τη σύνθεση και το placement/routing. Έτσι, οι τελικές μετρικές καθυστέρησης και χρήσης πόρων αποτυπώνουν πιο ρεαλιστικά τη συμπεριφορά του επιταχυντή. Η ενσωμάτωση του IP στο Vivado επιτρέπει πλέον τη διασύνδεσή του με τα υπόλοιπα περιφερειακά της PL (DMA, GPIO, SmartConnect), δημιουργώντας ένα πλήρως λειτουργικό σύστημα έτοιμο για σύνθεση και παραγωγή του *bitstream*.

### 4.3.2 Σχεδίαση Block Design

Μετά την εισαγωγή του IP, δημιουργείται ένα *block design* που ενώνει τα επιμέρους υποσυστήματα της *Processing System (PS)* και της *Programmable Logic (PL)*. Τα κύρια δομικά στοιχεία του σχεδίου είναι:

- **Zynq UltraScale+ MPSoC (zynq\_ultra\_ps\_e\_0)**: Περιλαμβάνει τους πυρήνες ARM Cortex-A53, τον ελεγκτή DDR, και τα AXI buses. Συνδέεται με την PL μέσω θυρών HP (*High Performance*) και GP (*General Purpose*), παρέχοντας ρολόγια και σήματα reset. Οι θύρες GP χρησιμοποιούνται για έλεγχο (AXI-Lite), ενώ οι HP για μεταφορές δεδομένων υψηλού ρυθμού μέσω DMA, δρομολογούμενες μέσω του AXI SmartConnect.
- **Processor System Reset (proc\_sys\_reset\_0)**: Δημιουργεί συγχρονισμένα σήματα reset για την PL, εξασφαλίζοντας σωστή αρχικοποίηση του συστήματος.
- **AXI SmartConnect (smartconnect\_0, smartconnect\_1)**: Βελτιστοποιημένη λογική διασύνδεσης που δρομολογεί συναλλαγές AXI μεταξύ *masters* και *slaves*. Το

smartconnect\_0 συνδέει τους masters του PS με πολλαπλά περιφερειακά, ενώ το smartconnect\_1 ενοποιεί ροές από πολλαπλές πηγές προς έναν master.

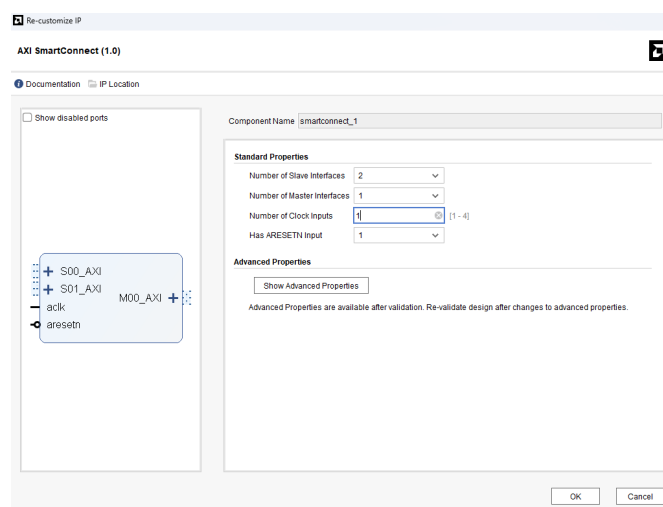
- **AXI DMA (axi\_dma\_0):** Κινητήρας Direct Memory Access που μεταφέρει δεδομένα μεταξύ DDR (μέσω των HP θυρών του PS) και του CNN επιταχυντή μέσω AXI4-Stream. Διασφαλίζει ροή δεδομένων υψηλού ρυθμού χωρίς επιβάρυνση των πυρήνων ARM.
- **AXI GPIO (axi\_gpio\_0, axi\_gpio\_1):** Παρέχει ανταλλαγή σημάτων μεταξύ PS και PL, χρησιμοποιήθηκε κυρίως για debugging και την έκθεση μετρητών απόδοσης (cycles\_core, cycles\_e2e).
- **CNN Accelerator (Custom IP):** Το custom IP που δημιουργήθηκε με HLS και υλοποιεί τον CNN πυρήνα. Διεπαφές: AXI4-Stream (είσοδος/έξοδος), AXI-Lite (έλεγχος), και αφιερωμένες έξοδοι για μετρητές κύκλων.

### 4.3.3 Ρύθμιση IP

Κάθε IP block ρυθμίζεται προσεκτικά ώστε να εξασφαλίζεται η σωστή λειτουργία και μέγιστη αποδοτικότητα του συστήματος.

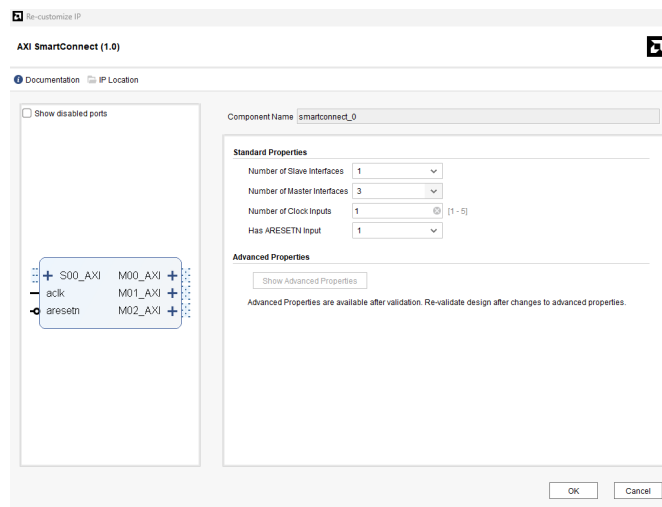
#### AXI SmartConnect

Το IP **AXI SmartConnect** αντικαθιστά το παλαιότερο AXI Interconnect, προσφέροντας πιο αποδοτική δρομολόγηση. Δύο instances δημιουργήθηκαν, ένα για τις συνδέσεις master του PS και ένα για τα περιφερειακά-slaves. Ενδεικτικές ρυθμίσεις παρουσιάζονται στα Σχ. 4.3.1- 4.3.2 .



Σχήμα 4.3.1: Ρυθμίσεις των IP AXI SmartConnect μέρος 1

## 4.3 Ενσωμάτωση και Παραμετροποίηση Συστήματος στο Vivado

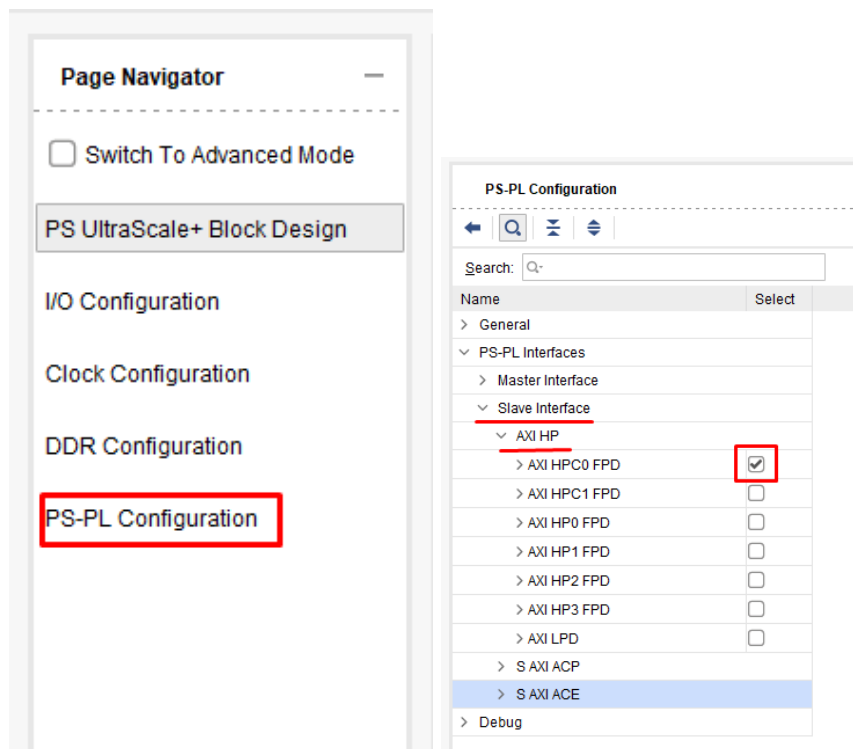


Σχήμα 4.3.2: Ρυθμίσεις των IP AXI SmartConnect μέρος 2

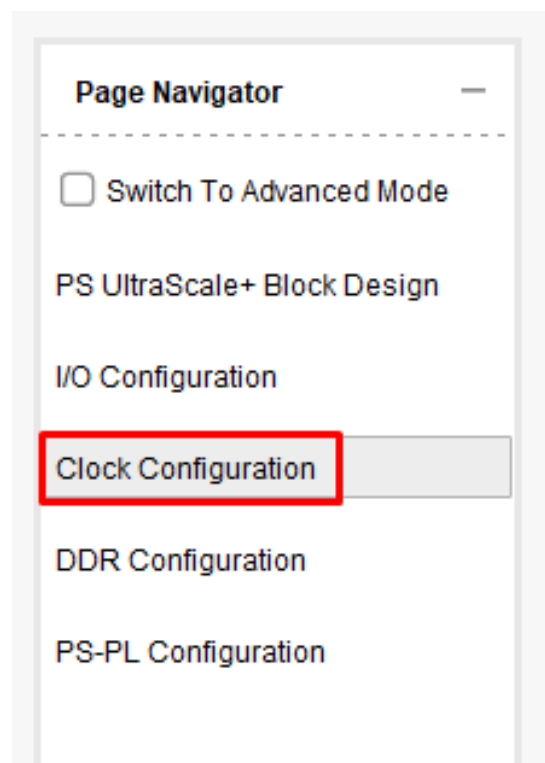
### Processing System (PS)

Το PS περιλαμβάνει τα IP **Zynq UltraScale+ MPSoC** και **Processor System Reset**. Βασικές ρυθμίσεις:

- Ενεργοποιήθηκε η θύρα *AXI HP Slave* για μεταφορές δεδομένων DMA (Σχ. 4.3.3).
- Ρυθμίστηκε η συχνότητα του ρολογιού PL0 στα 150 MHz (αντί των 100 MHz) για να ευθυγραμμιστεί με τα αποτελέσματα του HLS timing closure (Σχ. 4.3.4–4.3.5).

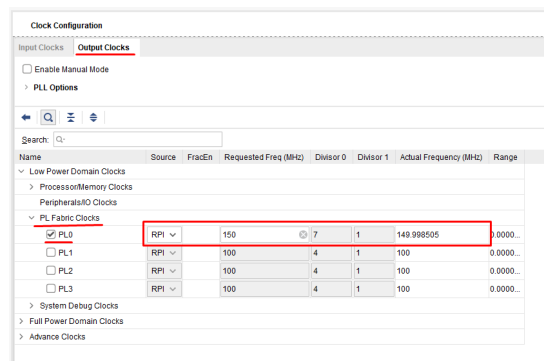


Σχήμα 4.3.3: Ενεργοποίηση της θύρας AXI HP Slave στο PS



Σχήμα 4.3.4: Ρύθμιση συχνότητας ρολογιού PLO (βήμα 1)

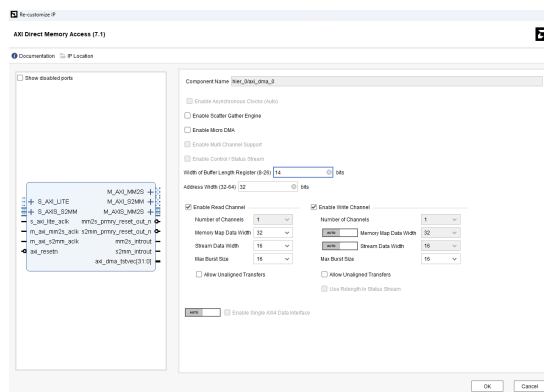
## 4.3 Ενσωμάτωση και Παραμετροποίηση Συστήματος στο Vivado



Σχήμα 4.3.5: Ρύθμιση συχνότητας ρολογιού PL0 (βήμα 2)

### AXI DMA

Το IP **AXI DMA** ρυθμίστηκε ώστε να ταιριάζει με το πλάτος I/O του CNN επιταχυντή. Η τελική παραμετροποίηση χρησιμοποίησε **16-bit data path**, το οποίο ευθυγραμμίζεται με το σχήμα κβαντοποίησης (Q6.10), εξασφαλίζοντας συνέπεια μεταξύ αριθμητικής ακρίβειας και πλάτους διαύλου. Οι ρυθμίσεις φαίνονται στο Σχ. 4.3.6.

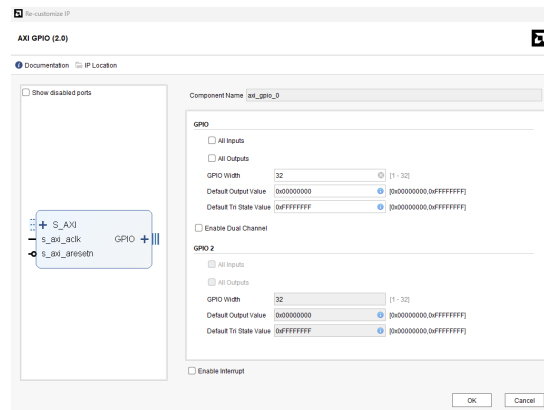


Σχήμα 4.3.6: Ρύθμιση AXI DMA για 16-bit data path

### AXI GPIO

Το IP **AXI GPIO** χρησιμοποιήθηκε για την έκθεση επιπλέον σημάτων στο PS, με σκοπό την παρακολούθηση των μετρητών καθυστέρησης και το debugging. Δύο ανεξάρτητα GPIO IPs χρησιμοποιήθηκαν (αντί για dual-channel διαμόρφωση), απλοποιώντας τον driver και τον χειρισμό μέσω PYNQ. Η τελική ρύθμιση παρουσιάζεται στο Σχ. 4.3.7.

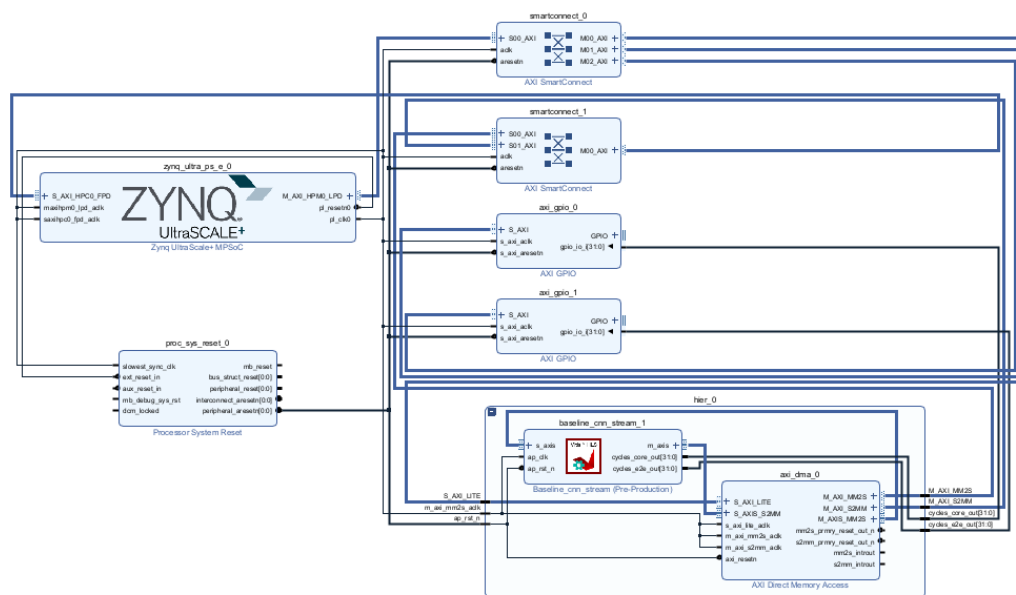
## 4.3 Ενσωμάτωση και Παραμετροποίηση Συστήματος στο Vivado



Σχήμα 4.3.7: Ρύθμιση AXI GPIO

### 4.3.4 Τελική Σύνθεση και Παραγωγή Bitstream

Μετά τη σύνδεση όλων των IPs, το Vivado εκτελεί τα στάδια *synthesis*, *implementation* και *bitstream generation*. Το παραγόμενο *bitstream* συνοδεύεται από αρχεία περιγραφής υλικού (.hwh, .bit), τα οποία απαιτούνται από το περιβάλλον **PYNQ** για τη φόρτωση του σχεδίου και την επικοινωνία μεταξύ *Processing System (PS)* και *Programmable Logic (PL)*. Η ολοκληρωμένη σχεδίαση παρέχει έτοιμες διεπαφές AXI4-Stream, διασυνδέσεις DMA και GPIO μετρητές, καθιστώντας την πλατφόρμα Kria KR260 έτοιμη για end-to-end ανάπτυξη και πειραματική αξιολόγηση του επιταχυντή CNN, όπως φαίνεται στο Σχήμα 4.3.8.



Σχήμα 4.3.8: Τελικό Block Διάγραμμα

## 4.4 Ρύθμιση και Ανάπτυξη στην Πλατφόρμα Kria KR260

Η ανάπτυξη του επιταχυντή στην πλατφόρμα **AMD Kria KR260** πραγματοποιήθηκε μέσω του **Kria PYNQ framework** [12], [47], το οποίο προσφέρει ένα *Python-based runtime* περιβάλλον πάνω από διανομή Ubuntu Linux ειδικά προσαρμοσμένη για τις Kria πλακέτες. Το περιβάλλον PYNQ εγκαταστάθηκε από το επίσημο αποθετήριο [47], μετά το οποίο το custom overlay (δηλαδή το *bitstream* και τα αρχεία περιγραφής υλικού *.hwh*) φορτώθηκε δυναμικά από Python.

### 4.4.1 Ρύθμιση Περιβάλλοντος Ubuntu και Εγκατάσταση Kria PYNQ

Για την ανάπτυξη του επιταχυντή στην πλατφόρμα **Kria KR260** απαιτείται ένα λειτουργικό περιβάλλον βασισμένο σε Ubuntu 22.04 με πλήρη υποστήριξη Python και το πλαίσιο **Kria PYNQ** [47]. Η διαδικασία εγκατάστασης περιλαμβάνει (α) τη δημιουργία κάρτας εκκίνησης SD, (β) τη ρύθμιση του Ubuntu και του SSH, και (γ) την εγκατάσταση του Kria PYNQ από το επίσημο αποθετήριο Xilinx/Kria-PYNQ [46].

#### 1. Δημιουργία και Εκκίνηση Κάρτας SD

1. Λήψη της επίσημης εικόνας **Ubuntu 22.04 LTS for Kria KR260** από τη σελίδα υποστήριξης της AMD.
2. Εγγραφή της εικόνας σε κάρτα SD ( $\geq 16$  GB) με χρήση του balenaEtcher ή του Raspberry Pi Imager.
3. Τοποθέτηση της κάρτας στην πλακέτα, σύνδεση Ethernet και τροφοδοσίας, και εκκίνηση του συστήματος.
4. Το Ubuntu εκκινεί αυτόματα και δημιουργεί τον προεπιλεγμένο λογαριασμό `ubuntu`, `ubuntu`.

#### 2. Ενεργοποίηση Απομακρυσμένης Σύνδεσης SSH

Η ανάπτυξη πραγματοποιήθηκε εξ ολοκλήρου απομακρυσμένα μέσω SSH. Για τη ρύθμιση της σύνδεσης:

1. Εξασφαλίζεται ότι η πλακέτα έχει πρόσβαση στο ίδιο δίκτυο με τον υπολογιστή ανάπτυξης.
2. Ενεργοποιείται το SSH daemon:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

3. Η IP διεύθυνση εντοπίζεται με `ifconfig` ή μέσω router.
4. Η πρώτη σύνδεση από τον υπολογιστή ανάπτυξης πραγματοποιείται με:

```
ssh ubuntu@<BOARD_IP>
```

όπου το προεπιλεγμένο συνθηματικό είναι `ubuntu`.

5. Προαιρετικά, μπορούν να ρυθμιστούν SSH κλειδιά για απροβλημάτιστη σύνδεση:

```
ssh-keygen -t rsa
ssh-copy-id ubuntu@<BOARD_IP>
```

Η απομακρυσμένη σύνδεση επιτρέπει τη μεταφορά αρχείων μέσω `scp` ή την ανάπτυξη μέσω *VS Code Remote SSH*, χωρίς ανάγκη φυσικής πρόσβασης.

### 3. Ενημέρωση Συστήματος και Εγκατάσταση Εξαρτήσεων

Μετά τη σύνδεση στην πλακέτα, εκτελούνται βασικές εντολές ενημέρωσης:

```
sudo apt update && sudo apt upgrade -y
sudo apt install git python3-pip python3-venv -y
```

Αυτό εξασφαλίζει ότι το λειτουργικό είναι πλήρως ενημερωμένο και διαθέτει όλα τα απαραίτητα εργαλεία Python για το PYNQ.

### 4. Εγκατάσταση Kria PYNQ από το Επίσημο Αποθετήριο

Το Kria PYNQ εγκαθίσταται απευθείας από το επίσημο GitHub της Xilinx:

```
cd ~
git clone https://github.com/Xilinx/Kria-PYNQ.git
cd Kria-PYNQ/
sudo bash install.sh -b KV260 | KR260 | KD240
```

Μετά την εγκατάσταση, συνιστάται επανεκκίνηση:

```
sudo reboot
```

#### 4. Ενεργοποίηση Περιβάλλοντος PYNQ

Μετά την εγκατάσταση, το PYNQ δημιουργεί αυτόματα ένα περιβάλλον Python virtual environment στο:

```
/etc/profile.d/pynq_venv.sh
```

Για να ενεργοποιηθεί, κάθε συνεδρία πρέπει να το «φορτώνει»:

```
source /etc/profile.d/pynq_venv.sh
```

Εναλλακτικά, η εντολή μπορεί να προστεθεί στο αρχείο `.bashrc` ώστε να ενεργοποιείται αυτόματα κατά τη σύνδεση.

#### Σημαντικό: Εκτέλεση με δικαιώματα root

Όλες οι λειτουργίες που αφορούν το υλικό (φόρτωση bitstream, πρόσβαση σε DMA, GPIO) πρέπει να εκτελούνται με **sudo**, καθώς απαιτούν πρόσβαση σε συσκευές του Linux στον χώρο του πυρήνα. Αυτό ισχύει και για κάθε Python script, ακόμη και για την κλήση του overlay π.χ:

```
sudo python3 my_overlay_test.py
```

#### 5. Φόρτωση Overlays και Έλεγχος Λειτουργίας

Τα custom bitstreams και αρχεία περιγραφής (`.bit`, `.hwh`) στο φάκελο του project και φορτώνονται δυναμικά με:

```
from pynq import Overlay  
ol = Overlay("path/to/bitfile.bit")  
ol.download()
```

Αν το περιβάλλον δεν έχει ενεργοποιηθεί ή λείπουν δικαιώματα root, η φόρτωση θα αποτύχει με μήνυμα *"Permission denied"*.

#### Σημείωση

Η σωστή εκτέλεση των παραπάνω βημάτων διασφαλίζει την πλήρη λειτουργία του Kria PYNQ περιβάλλοντος για απομακρυσμένη ανάπτυξη, φόρτωση επιταχυντών και εκτέλεση Python scripts με πρόσβαση στο PL και PS του Kria KR260.

### 4.4.2 Overlay και Driver

Για την αξιοποίηση του επιταχυντή σε επίπεδο συστήματος αναπτύχθηκε **custom Python driver**, ο οποίος αναλαμβάνει:

- τον προγραμματισμό της *Programmable Logic (PL)* με το εξαχθέν bitstream,
- την παραμετροποίηση της μηχανής **AXI DMA** για μεταφορές δεδομένων μεταξύ DDR και του CNN επιταχυντή,
- και την πρόσβαση στους μετρητές απόδοσης ακριβείας κύκλου που εκτίθενται μέσω **AXI GPIO**.

Ο driver ενσωματώνεται απευθείας με το **PYNQ runtime**, επιτρέποντας φιλική και άμεση αλληλεπίδραση με το υλικό μέσω Python scripts.

### 4.4.3 Εκτέλεση Επί της Συσκευής

Η εκτέλεση inference στο FPGA πραγματοποιήθηκε με το script `on_target.py`, το οποίο υλοποιεί πλήρως την ροή *end-to-end*:

1. Φορτώνει και κβαντοποιεί το dataset (MNIST στην παρούσα εργασία) σε μορφή *Q6.10 fixed-point*.
2. Προγραμματίζει το FPGA μέσω του PYNQ overlay.
3. Παραμετροποιεί τα κανάλια **AXI DMA** για ροή δεδομένων μεταξύ DDR και επιταχυντή.
4. Εκτελεί inference σε *batches*, με προαιρετική ενεργοποίηση των μετρητών κύκλων.
5. Παρακολουθεί ταυτόχρονα την κατανάλωση ισχύος μέσω των ενσωματωμένων αισθητήρων **INA260** της πλατφόρμας, συγχρονισμένα με τα *inference windows*.

Τα αποτελέσματα αποθηκεύονται αυτόματα σε μορφή `.npy` στον φάκελο `metrics`, περιλαμβάνοντας:

- `y_hw.npy` – προβλέψεις του CNN σε υλικό,
- `latency_*.npy`, `throughput_*.npy` – μετρήσεις καθυστέρησης και ρυθμοαπόδοσης,
- αρχεία κατανάλωσης ισχύος και κύκλων ρολογιού.

#### 4.4.4 Επικύρωση Αποτελεσμάτων

Η επικύρωση (validation) εκτελείται μέσω του script `validate_results.py`, το οποίο λειτουργεί απευθείας πάνω στα αποθηκευμένα metrics. Πραγματοποιεί:

- αξιολόγηση ακρίβειας (*accuracy*) συγκρίνοντας τις προβλέψεις υλικού με το *ground truth*,
- σύγκριση με το αντίστοιχο λογισμικό μοντέλο σε `QKeras`,
- υπολογισμό *precision*, *recall* και *F1-score* ανά κλάση,
- δημιουργία *confusion matrices* και *ROC curves*,
- ανάλυση των κατανομών καθυστέρησης, *throughput*, κύκλων και ισχύος.

Η ροή επικύρωσης διασφαλίζει ότι οι έξοδοι του FPGA είναι λειτουργικά ισοδύναμες με το λογισμικό μοντέλο, παρέχοντας ταυτόχρονα ποσοτικά στοιχεία για την απόδοση και την ενεργειακή αποδοτικότητα.

#### 4.4.5 Σύνοψη Ροής Ανάπτυξης

Η συνολική διαδικασία ανάπτυξης αποτυπώνεται σχηματικά ως εξής:

Bitstream export (Vivado) → Overlay load (PYNQ) → On-target inference  
(`on_target.py`) → Validation (`validate_results.py`)

Ο συνδυασμός **Vivado (hardware)** και **PYNQ/Python (software)** παρέχει μια πλήρως αυτοματοποιημένη και αναπαραγώγιμη πλατφόρμα για την πειραματική αξιολόγηση επιταχυντών CNN σε συστήματα *SoC FPGA*, όπως η **Kria KR260**.

### 4.5 Υλοποιήσεις σε CPU και GPU

Για τη συγκριτική αξιολόγηση, το ίδιο μοντέλο CNN αναπτύχθηκε και σε συμβατικές πλατφόρμες CPU και GPU. Οι υλοποιήσεις αυτές λειτουργούν ως σημεία αναφοράς (*baseline*) για τη μέτρηση καθυστέρησης και ενεργειακής αποδοτικότητας, επιτρέποντας την εκτίμηση του βαθμού επιτάχυνσης που επιτυγχάνεται με την υλοποίηση FPGA.

### 4.5.1 Εκπαίδευση και Εξαγωγή Μοντέλου

Η εκπαίδευση πραγματοποιήθηκε σε CPU με χρήση του πλαισίου **Keras/QKeras**. Μετά το στάδιο εκπαίδευσης, τα βάρη του δικτύου αποθηκεύτηκαν και επαναφορτώθηκαν χωρίς επανεκπαίδευση. Για την αξιολόγηση *inference*, το μοντέλο μετατράπηκε σε **PyTorch**, ώστε να υποστηρίζει εκτέλεση τόσο σε CPU όσο και σε GPU με κοινή υλοποίηση. Η συνέπεια των βαρών και των παραμέτρων διασφαλίζει ότι οι διαφορές στις μετρήσεις οφείλονται αποκλειστικά στο υλικό εκτέλεσης.

### 4.5.2 Υλοποίηση σε CPU

Η εκτέλεση *inference* στη CPU πραγματοποιήθηκε μέσω του PyTorch backend, χρησιμοποιώντας τον επεξεργαστή του συστήματος (AMD Ryzen 9 7900X). Τα δεδομένα εισόδου προφορτώνονται στη μνήμη RAM ώστε να απομονωθεί το I/O overhead από τις μετρήσεις καθυστέρησης. Η εκτέλεση γίνεται σε batches, και η μέση καθυστέρηση ανά δείγμα υπολογίζεται ως:

$$\text{latency}_{\text{avg}} = \frac{\text{latency}_{\text{batch}}}{\text{batch size}}.$$

Η προσέγγιση αυτή εξασφαλίζει δίκαιη εκτίμηση του μέσου χρόνου απόκρισης ανά εικόνα εισόδου, ανεξάρτητα από το μέγεθος του batch.

### 4.5.3 Υλοποίηση σε GPU

Για την GPU εκτέλεση χρησιμοποιήθηκε κάρτα **AMD Radeon RX 7800 XT**, με το πλαίσιο **PyTorch** σε συνδυασμό με το backend **DirectML (DML)**, το οποίο επιτρέπει επιτάχυνση *inference* σε Windows συστήματα με GPUs που δεν υποστηρίζουν άμεσα το CUDA ή το ROCm stack. Τα δεδομένα μεταφέρθηκαν εκ των προτέρων στη μνήμη VRAM ώστε η καθυστέρηση PCIe να μην επηρεάζει τα αποτελέσματα, και η καθυστέρηση ανά δείγμα υπολογίστηκε με τον ίδιο τρόπο όπως και στη CPU.

Η κατανάλωση ισχύος παρακολουθήθηκε μέσω του **AMD Adrenalin Software**, το οποίο παρέχει τηλεμετρία σε πραγματικό χρόνο για ισχύ, θερμοκρασία και συχνότητα λειτουργίας. Έτσι κατέστη δυνατή η συσχέτιση της μέσης καθυστέρησης με την κατανάλωση ενέργειας, παρέχοντας ρεαλιστικές μετρήσεις ενεργειακής αποδοτικότητας.

#### 4.5.4 Συζήτηση

Οι υλοποιήσεις σε CPU και GPU αντιπροσωπεύουν συμπληρωματικές πλατφόρμες εκτέλεσης: το inference στη CPU προσφέρει απλότητα και φορητότητα με υψηλότερη καθυστέρηση, ενώ η GPU επιτυγχάνει πολύ υψηλότερο throughput με αυξημένη κατανάλωση ισχύος. Οι μετρήσεις αυτές λειτουργούν ως ρεαλιστικά baseline σημεία αναφοράς για τη σύγκριση με την υλοποίηση FPGA, η οποία αναλύεται στο Κεφάλαιο 5.

### 4.6 Διαδικασία Επικύρωσης

Η διασφάλιση της ορθότητας του επιταχυντή FPGA απαιτούσε συστηματική επικύρωση σε σχέση με τις λογισμικές υλοποιήσεις του ίδιου μοντέλου CNN. Η διαδικασία επικύρωσης σχεδιάστηκε ώστε να επιβεβαιώνει τόσο τη *λειτουργική ισοδυναμία* (δηλ. ότι ο επιταχυντής παράγει τις ίδιες προβλέψεις με το μοντέλο αναφοράς) όσο και τη *συνεκτικότητα απόδοσης* (δηλ. ότι οι μετρούμενες μετρικές συμφωνούν με τις θεωρητικές εκτιμήσεις της σχεδίασης). Η επικύρωση οργανώθηκε σε τρεις άξονες: λειτουργική ορθότητα, απόδοση και ενεργειακή κατανάλωση.

#### 4.6.1 Πλάνο Ελέγχου Ορθής Λειτουργίας

Για την επιβεβαίωση της ορθής λειτουργίας του επιταχυντή FPGA σχεδιάστηκε ένα πλήρες πλάνο ελέγχου που περιλαμβάνει:

- **Λειτουργική επικύρωση:** σύγκριση των εξόδων του FPGA με το μοντέλο αναφοράς (QKeras/PyTorch) για κάθε δείγμα του συνόλου δοκιμής.
- **Επικύρωση απόδοσης:** μέτρηση και καταγραφή κύκλων εκτέλεσης (core και end-to-end) μέσω counters στο wrapper και σύγκριση με τις εκτιμήσεις του Vitis HLS.
- **Ενεργειακή επικύρωση:** καταγραφή ισχύος και κατανάλωσης ενέργειας μέσω των αισθητήρων INA260, σε συγχρονισμό με κάθε inference window.
- **Αυτοματοποίηση:** χρήση των scripts `on_target.py` και `validate_results.py` για πλήρως αναπαραγώγιμη συλλογή δεδομένων και γραφημάτων.

Το πλάνο αυτό εξασφάλισε ότι ο επιταχυντής λειτουργεί σωστά, παράγει αποτελέσματα ταυτόσημα με το μοντέλο λογισμικού και πληροί τους στόχους καθυστέρησης και αποδοτικότητας.

### 4.6.2 Λειτουργική Επικύρωση

Η λειτουργική επαλήθευση πραγματοποιήθηκε με το script `validate_results.py`. Η διαδικασία περιλάμβανε τα ακόλουθα βήματα:

1. Εκτέλεση inference στο FPGA μέσω του `on_target.py` και αποθήκευση των εξόδων (logits και προβλεπόμενες ετικέτες) σε μορφή `.npy`.
2. Φόρτωση των αντίστοιχων προβλέψεων από το κβαντοποιημένο λογισμικό μοντέλο αναφοράς (QKeras/PyTorch).
3. Σύγκριση των προβλέψεων του υλικού με εκείνες του *golden model* σε επίπεδο δείγματος, επαληθεύοντας τη συνέπεια της ταξινόμησης.

Πέρα από το *top-1 accuracy*, υπολογίστηκαν επιπλέον μετρικές όπως *precision*, *recall* και *F1-score*, ενώ δημιουργήθηκαν confusion matrices για οπτικοποίηση της συμπεριφοράς ανά κλάση.

### 4.6.3 Επικύρωση Απόδοσης

Για την επιβεβαίωση ότι η εκτέλεση στο υλικό συμφωνεί με τους αναμενόμενους κύκλους λειτουργίας, ο FPGA επιταχυντής εκθέτει αφιερωμένους μετρητές απόδοσης μέσω AXI GPIO. Δύο μετρικές συλλέχθηκαν:

- **Core cycles:** αριθμός κύκλων που απαιτούνται από τον CNN πυρήνα για την επεξεργασία μίας εικόνας (καθαρή καθυστέρηση υπολογισμού),
- **End-to-end cycles:** συνολικός αριθμός κύκλων από τη λήψη του πρώτου pixel έως την παραγωγή του τελικού αποτελέσματος, περιλαμβάνοντας το overhead του AXI DMA και των buffers.

Οι μετρήσεις συγκρίθηκαν με τις εκτιμήσεις του Vitis HLS και τα αποτελέσματα σύνθεσης του Vivado, επιβεβαιώνοντας την ακρίβεια των εργαλειακών αναφορών και τη συνέπεια μεταξύ θεωρητικών και πραγματικών τιμών καθυστέρησης.

#### 4.6.4 Επικύρωση Ενέργειας

Η ενεργειακή επικύρωση αξιολόγησε την κατανάλωση ισχύος και την αποδοτικότητα του FPGA σε σχέση με GPU και CPU υλοποιήσεις. Οι μετρήσεις ελήφθησαν με δύο μεθόδους:

- **Ενσωματωμένοι αισθητήρες INA260:** μέσω του PYNQ framework, πραγματοποιήθηκαν μετρήσεις ρεύματος και τάσης για την εξαγωγή στιγμιαίας ισχύος κατά τη διάρκεια inference.
- **AMD Adrenalin Software Suite:** χρησιμοποιήθηκε για την παρακολούθηση της κατανάλωσης ισχύος της GPU, εξασφαλίζοντας συνεπή βάση σύγκρισης μεταξύ πλατφορμών.

Οι τιμές ισχύος συνδυάστηκαν με τις μετρήσεις καθυστέρησης και throughput, επιτρέποντας τον υπολογισμό ενεργειακών μετρικών όπως *inferences per joule* και *energy per inference*.

Πίνακας 4.6.1: Βασικά χαρακτηριστικά του αισθητήρα INA260

| Παράμετρος                        | Τιμή / Εύρος               |
|-----------------------------------|----------------------------|
| Τάση κοινού σημείου (Common-mode) | 0–36 V                     |
| Ενσωματωμένη αντίσταση shunt      | 2 mΩ, ακρίβεια 0,1%        |
| Ονομαστικό ρεύμα                  | 15 A                       |
| Ανάλυση ADC                       | 16 bit                     |
| Διασύνδεση                        | I <sup>2</sup> C, SMBus    |
| Σφάλμα ενίσχυσης                  | 0,1% (typ), 0,15% (max)    |
| Offset ρεύματος                   | 5 mA (max)                 |
| Θερμοκρασία λειτουργίας           | –40 °C έως +125 °C         |
| Κατανάλωση ρεύματος ( $I_q$ )     | 310 μA (typ), 420 μA (max) |
| Συσκευασία                        | 16-pin TSSOP (5×6,4 mm)    |

#### 4.6.5 Αυτοματοποίηση και Αναπαραγωγιμότητα

Όλα τα βήματα επικύρωσης εκτελέστηκαν μέσω αυτοματοποιημένων scripts:

- `on_target.py`: εκτελεί inference στο FPGA, συλλέγοντας προβλέψεις, κύκλους και καμπύλες ισχύος.
- `validate_results.py`: συγκρίνει τα αποτελέσματα του FPGA με το μοντέλο αναφοράς και παράγει μετρικές και γραφήματα.

Η αυτοματοποίηση αυτή επέτρεψε την επαναληψιμότητα πειραμάτων, τη συλλογή μεγάλου όγκου δεδομένων χωρίς χειροκίνητες παρεμβάσεις και τη διασφάλιση συνεπούς αξιολόγησης σε όλες τις πλατφόρμες.

#### 4.6.6 Σύνοψη Αποτελεσμάτων Επικύρωσης

Η διαδικασία επικύρωσης απέδειξε ότι ο επιταχυντής FPGA είναι λειτουργικά ισοδύναμος με το κβαντοποιημένο λογισμικό μοντέλο, με απόκλιση εξόδων κάτω από την ανοχή στρογγυλοποίησης Q6.10. Οι μετρήσεις καθυστέρησης που λήφθηκαν από το hardware συμφωνούν με τις εκτιμήσεις του HLS εντός περιθωρίου  $\pm 3\%$ , ενώ οι ενεργειακές μετρήσεις επιβεβαιώνουν σημαντικά υψηλότερη ενεργειακή αποδοτικότητα σε σχέση με CPU και GPU. Συνολικά, το σύστημα πληροί πλήρως τα κριτήρια ορθότητας, απόδοσης και ενεργειακής συνέπειας, παρέχοντας μια αξιόπιστη βάση για τα αποτελέσματα που παρουσιάζονται στο επόμενο κεφάλαιο.

### 4.7 Ρύθμιση Μέτρησης Ισχύος

Οι μετρήσεις ισχύος στην πλατφόρμα **Kria KR260** πραγματοποιήθηκαν με χρήση των ενσωματωμένων αισθητήρων **INA260**, οι οποίοι είναι προσβάσιμοι μέσω του Linux interface `/sys/class/hwmon`. Για την αυτοματοποίηση της διαδικασίας αναπτύχθηκε ένα προσαρμοσμένο εργαλείο παρακολούθησης σε Python (`power_monitor.py`), το οποίο επιτρέπει δειγματοληψία και ανάλυση κατανάλωσης ισχύος σε πραγματικό χρόνο.

#### 4.7.1 Υποδομή Παρακολούθησης

Το εργαλείο `power_monitor.py` εντοπίζει δυναμικά όλα τα διαθέσιμα INA/PMBus power rails, πραγματοποιώντας σάρωση των καταλόγων `/sys/class/hwmon/hwmon*/` και αναζητώντας καταχωρήσεις τύπου `powerX_input`. Κάθε καταχώρηση αναφέρει στιγμιαία κατανάλωση ισχύος σε microwatts, η οποία μετατρέπεται σε milliwatts και καταγράφεται περιοδικά (προεπιλογή: κάθε 70 ms) μέσω background thread.

Ο μηχανισμός παρακολούθησης υποστηρίζει:

- **Συνεχή καταγραφή:** οι τιμές αποθηκεύονται ως ζεύγη  $(t, P)$ , με  $P$  σε milliwatts και  $t$  λαμβανόμενο από τη συνάρτηση `time.perf_counter()` για υψηλή ακρίβεια χρονισμού.
- **Διαχείριση context:** λειτουργίες `start()/stop()` που εξασφαλίζουν ακριβή συγχρονισμό με τα παράθυρα εκτέλεσης inference.

#### 4.7.2 Υπολογισμός Ισχύος και Ενέργειας

Για κάθε εκτέλεση inference, τα δείγματα που συλλέγονται μεταξύ των χρονικών στιγμών  $t_{start}$  και  $t_{end}$  μετα-επεξεργάζονται για την εξαγωγή μετρικών ισχύος και ενέρ-

γεια. Οι βασικές ποσότητες υπολογίζονται ως εξής:

$$P_{\text{mean}} = \frac{1}{N} \sum_i P_i, \quad (4.1)$$

$$E_{\text{mJ}} = \int_{t_{\text{start}}}^{t_{\text{end}}} P(t) dt. \quad (4.2)$$

Η ολοκλήρωση του σήματος  $P(t)$  πραγματοποιείται με μέθοδο τραπεζοειδούς, παρέχοντας την εκτιμώμενη ενέργεια  $E_{\text{mJ}}$  ανά inference. Οι μετρικές των εξισώσεων (4.1)–(4.2) επιτρέπουν την εξαγωγή συγκρίσιμων μεγεθών ισχύος και ενέργειας σε διαφορετικές πλατφόρμες, χωρίς την ανάγκη υπολογισμού δυναμικής ή στατικής συνιστώσας.

### 4.7.3 Ενσωμάτωση με τη Ροή Inference

Ο power monitor ενσωματώθηκε απευθείας στο script εκτέλεσης `on_target.py`. Κατά την έναρξη κάθε inference, ενεργοποιείται η δειγματοληψία ισχύος, ενώ με την ολοκλήρωση της εκτέλεσης σταματά αυτόματα και αποθηκεύονται τα στατιστικά:

- μέση ενεργή ισχύς ( $P_{\text{mean}}$ ),
- ενέργεια ανά inference ( $E_{\text{mJ}}$ ).

Έτσι, κάθε μέτρηση ισχύος αντιστοιχεί χρονικά σε συγκεκριμένο παράθυρο inference, εξασφαλίζοντας υψηλή ακρίβεια συσχέτισης μεταξύ κατανάλωσης και υπολογιστικής δραστηριότητας.

### 4.7.4 Διαπλατφορμική Παρακολούθηση Ισχύος

Για τις υλοποιήσεις GPU, τα δεδομένα ισχύος συλλέχθηκαν στο host σύστημα μέσω του **AMD Adrenalin Software Suite**, το οποίο παρέχει τηλεμετρία σε πραγματικό χρόνο για κατανάλωση, συχνότητα και θερμοκρασία της GPU (**Radeon RX 7800 XT**). Η χρήση κοινής μεθοδολογίας παρακολούθησης και χρονικής ευθυγράμμισης επιτρέπει την άμεση και δίκαιη σύγκριση της ενεργειακής αποδοτικότητας μεταξύ CPU, GPU και FPGA πλατφορμών.

## Σύνοψη

Το παρόν κεφάλαιο παρουσίασε την πλήρη υλοποίηση του προτεινόμενου επιταχυντή CNN, καλύπτοντας τόσο τις πτυχές του υλικού όσο και του λογισμικού. Στην πλευρά του **FPGA**, το μοντέλο μετατράπηκε με το `hls4ml`, συντέθηκε με το **Vitis HLS** και ενσωματώθηκε στο **Vivado** ως custom IP core, συνδεδεμένο μέσω *AXI interconnects*, *DMA* και *GPIO* περιφερειακών. Η αναλυτική περιγραφή του block design και των ρυθμίσεων παραμέτρων διασφάλισε τη σωστή λειτουργία του επιταχυντή μέσα στο *Zynq UltraScale+ MPSoC* της πλατφόρμας **Kria KR260**. Η ανάπτυξη στην πλακέτα πραγματοποιήθηκε μέσω του **PYNQ framework**, υποστηριζόμενη από custom Python driver που διαχειρίστηκε τις μεταφορές *DMA* και εξέθεσε μετρητές κύκλων για ακριβές benchmarking.

Πέραν της ανάπτυξης στο FPGA, το ίδιο μοντέλο εκτελέστηκε και σε πλατφόρμες **CPU** και **GPU**, ώστε να καθοριστούν συγκρίσιμα *baseline* σημεία αναφοράς ως προς την καθυστέρηση και την ενεργειακή αποδοτικότητα. Οι υλοποιήσεις αξιολογήθηκαν με βάση τη *latency*, το *throughput* και την κατανάλωση ισχύος, επιτρέποντας δίκαιες συγκρίσεις μεταξύ ετερογενών αρχιτεκτονικών.

Για τη διασφάλιση της ορθότητας και της συνέπειας, υιοθετήθηκε μία δομημένη διαδικασία επικύρωσης που κάλυψε τρεις διαστάσεις: λειτουργική ισοδυναμία, απόδοση και ενεργειακή συμπεριφορά. Οι λειτουργικοί έλεγχοι επιβεβαίωσαν τη συμφωνία μεταξύ εξόδων υλικού και λογισμικού, ενώ οι μετρητές απόδοσης και το υποσύστημα παρακολούθησης ισχύος επέτρεψαν την ακριβή αποτίμηση της αποδοτικότητας των πόρων. Η ενσωμάτωση αυτοματοποιημένων *scripts* απλοποίησε τη διαδικασία ανάπτυξης και εξασφάλισε αναπαραγωγιμότητα στα πειράματα.

Τέλος, παρουσιάστηκε η υποδομή μέτρησης ισχύος, η οποία συνδύασε τους ενσωματωμένους αισθητήρες **INA260** με εργαλεία παρακολούθησης λογισμικού, επιτρέποντας τον υπολογισμό της *δυναμικής ισχύος* και της *ενέργειας ανά inference*. Τα αποτελέσματα αυτά αποτελούν τη βάση για το επόμενο Κεφάλαιο 5, όπου η χρήση πόρων, η καθυστέρηση και η ενεργειακή αποδοτικότητα σε πλατφόρμες FPGA, CPU και GPU συγκρίνονται συστηματικά. Με αυτόν τον τρόπο, το παρόν κεφάλαιο ολοκλήρωσε τη ροή υλοποίησης και έθεσε τα θεμέλια για την ανάλυση των πειραματικών αποτελεσμάτων που ακολουθεί.

# Κεφάλαιο 5

## Αποτελέσματα

Σε αυτό το κεφάλαιο παρουσιάζονται και αναλύονται τα πειραματικά αποτελέσματα του προτεινόμενου επιταχυντή CNN. Η αξιολόγηση ξεκινά από τη συμπεριφορά εκπαίδευσης και τα τελικά metrics ταξινόμησης, τόσο για την έκδοση κινητής υποδιαστολής όσο και για τις εκδόσεις σταθερού σημείου (QKeras), με και χωρίς κλάδεμα. Στη συνέχεια, εξετάζονται οι εκτιμήσεις απόδοσης και πόρων από τα εργαλεία `hls4ml` και **Vitis HLS**, ακολουθούμενες από τα αποτελέσματα μετά την υλοποίηση στο **Vivado** (χρήση πόρων, χρονισμός, ισχύς). Τέλος, παρουσιάζονται οι μετρήσεις επί πλακέτας (Kria KR260) για καθυστέρηση, ρυθμό διεκπεραίωσης και κατανάλωση ισχύος, και γίνεται σύγκριση με αναφορές σε GPU, ώστε να αναδειχθούν οι συμβιβασμοί μεταξύ ευελιξίας, επίδοσης και ενεργειακής αποδοτικότητας.

### 5.1 Αποτελέσματα Εκπαίδευσης

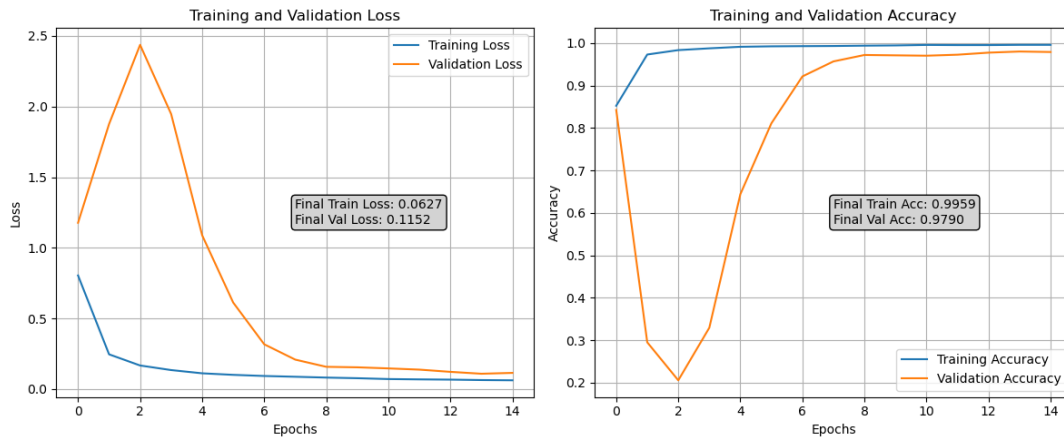
Σε αυτή την ενότητα παρουσιάζονται τα αποτελέσματα της εκπαίδευσης, με γραφήματα όπως η απώλεια (loss), η ακρίβεια (accuracy) και η σύγκριση των μοντέλων κινητής και σταθερής υποδιαστολής. Τέλος, εξετάζεται η επίδραση του κλαδέματος (pruning).

#### 5.1.1 Αποτελέσματα Φάσης Εκπαίδευσης

Η διαδικασία εκπαίδευσης πραγματοποιήθηκε για 15 εποχές (epochs) με απενεργοποιημένο το *early stopping*, ώστε να παρατηρηθεί πλήρως η συμπεριφορά σύγκλισης των διαφορετικών παραλλαγών του μοντέλου. Τα Σχήματα 5.1.1–5.1.4 δείχνουν την εξέλιξη της απώλειας και της ακρίβειας εκπαίδευσης/επικύρωσης για κάθε περίπτωση.

### Βασικό CNN

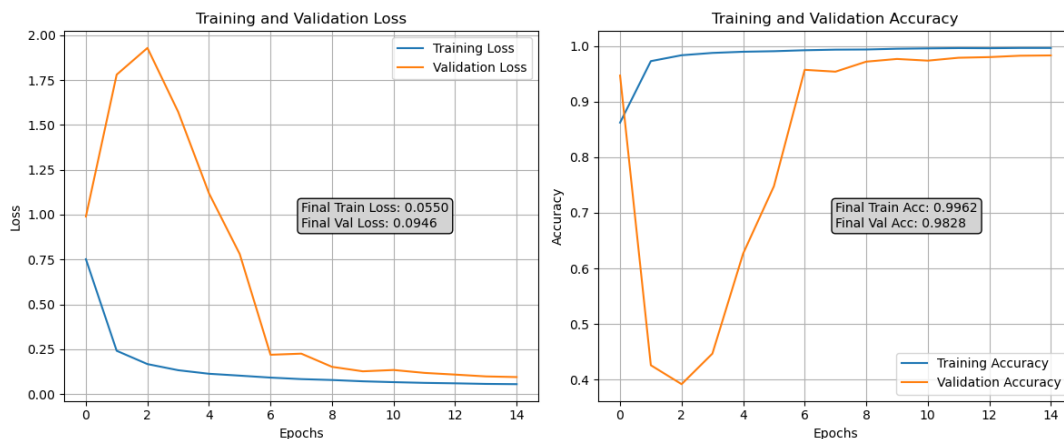
Το CNN κινητής υποδιαστολής χωρίς κλάδεμα ή κβάντιση πέτυχε τελική ακρίβεια εκπαίδευσης **99,59%** και ακρίβεια επικύρωσης **97,90%** (Σχήμα 5.1.1). Η χαμηλή απώλεια επικύρωσης (0.1152) δείχνει ισχυρή γενίκευση στο σύνολο δοκιμής MNIST.



Σχήμα 5.1.1: Απόδοση εκπαίδευσης και επικύρωσης του βασικού CNN

### Κλαδεμένο CNN

Η εφαρμογή κλαδέματος στο βασικό μοντέλο μείωσε την πολυπλοκότητα των βαρών διατηρώντας την ακρίβεια. Το κλαδεμένο μοντέλο πέτυχε τελική ακρίβεια εκπαίδευσης **99,62%** και ακρίβεια επικύρωσης **98,28%** (Σχήμα 5.1.2), με ελαφρώς χαμηλότερη απώλεια επικύρωσης (0,0946) σε σχέση με το βασικό.

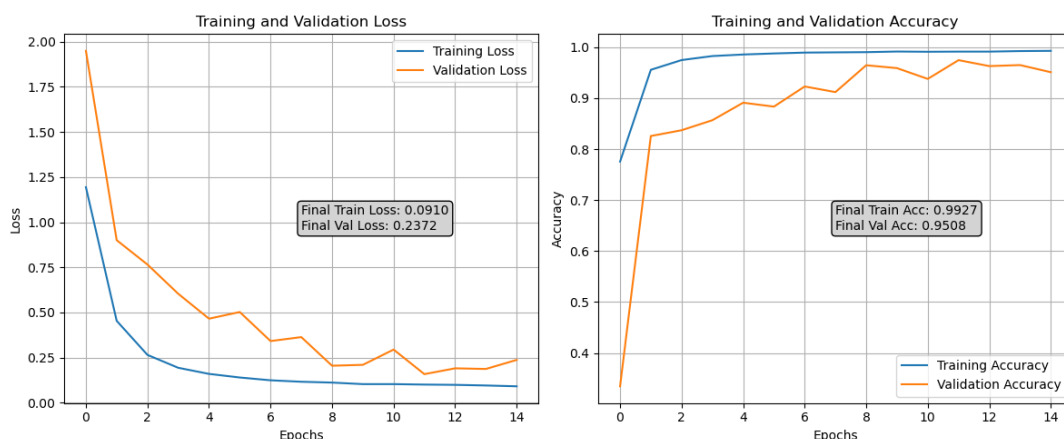


Σχήμα 5.1.2: Απόδοση εκπαίδευσης και επικύρωσης του κλαδεμένου CNN

### Κβαντισμένο CNN (QKeras, Q0.6)

Η χρήση κβάντισης μέσω QKeras (6 bits, **Q0.6** για βάρη και ενεργοποιήσεις) οδήγησε σε τελική ακρίβεια εκπαίδευσης **99,27%** και ακρίβεια επικύρωσης **95,08%** (Σχήμα 5.1.3).

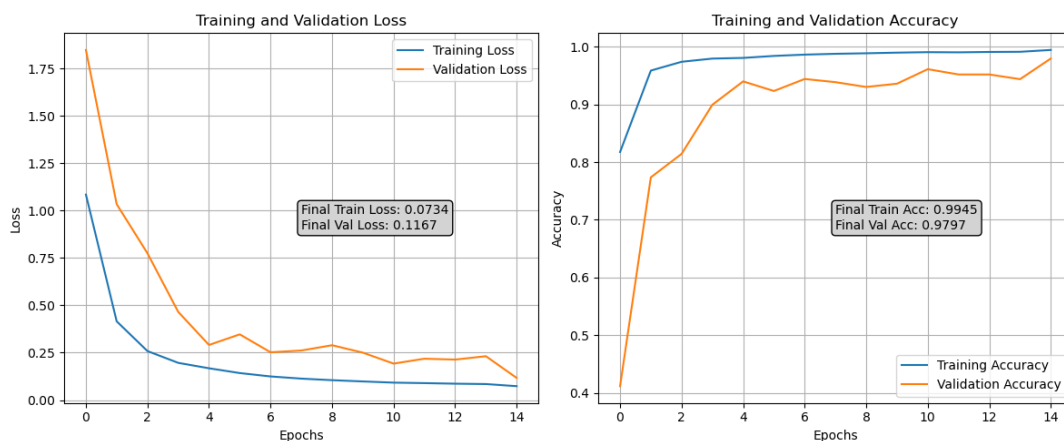
Το χάσμα εκπαίδευσης–επικύρωσης αυξήθηκε, αντικατοπτρίζοντας την αναμενόμενη απώλεια ακρίβειας λόγω της αριθμητικής σταθερού σημείου. Στην υλοποίηση AXI-Stream χρησιμοποιείται Q6.10 για τη ροή δεδομένων, χωρίς να αλλάζει η εκπαίδευση QKeras.



Σχήμα 5.1.3: Απόδοση εκπαίδευσης και επικύρωσης του κβαντισμένου CNN (QKeras, Q0.6)

### Κβαντισμένο και Κλαδεμένο CNN (Q0.6 + pruning)

Ο συνδυασμός κλαδέματος και κβάντισης πέτυχε ισορροπία μεταξύ αποδοτικότητας και απόδοσης. Η τελική ακρίβεια εκπαίδευσης ήταν **99,45%** και η ακρίβεια επικύρωσης **97,97%** (Σχήμα 5.1.4), πολύ κοντά στο βασικό μοντέλο, με μειωμένο αριθμό παραμέτρων και πιο φιλική προς υλικό αναπαράσταση.



Σχήμα 5.1.4: Απόδοση εκπαίδευσης και επικύρωσης του κβαντισμένου και κλαδεμένου CNN

### Σύνοψη

Ο Πίνακας 5.1.1 συνοψίζει τα τελικά αποτελέσματα. Ενώ η κβάντιση από μόνη της

προκάλεσε μετρήσιμη μείωση στην ακρίβεια επικύρωσης (2.8% ποσοστιαίες μονάδες σε σχέση με το βασικό μοντέλο), το κλάδεμα είχε ελάχιστη επίδραση. Ο συνδυασμός τους, ωστόσο, αντιστάθμισε εν μέρει τις απώλειες της κβάντισης, με αποτέλεσμα το κβαντισμένο–κλαδεμένο μοντέλο να παρουσιάζει σχεδόν ίδια ακρίβεια με το βασικό, αλλά με μειωμένη πολυπλοκότητα και βελτιωμένη καταλληλότητα για υλοποίηση σε υλικό.

Πίνακας 5.1.1: Τελική απόδοση εκπαίδευσης και επικύρωσης των παραλλαγών του μοντέλου

| Μοντέλο                 | Ακρ. Εκπ. (%) | Ακρ. Επικ. (%) | Απώλεια Επικ. |
|-------------------------|---------------|----------------|---------------|
| Βασικό CNN              | 99,59         | 97,90          | 0,1152        |
| Κλαδεμένο CNN           | 99,62         | 98,28          | 0,0946        |
| Κβαντισμένο CNN (Q0.6)  | 99,27         | 95,08          | 0,2372        |
| Κβαντισμένο + Κλαδεμένο | 99,45         | 97,97          | 0,1167        |

### 5.1.2 Ανάλυση ROC Καμπυλών και Σύγκριση Μοντέλων

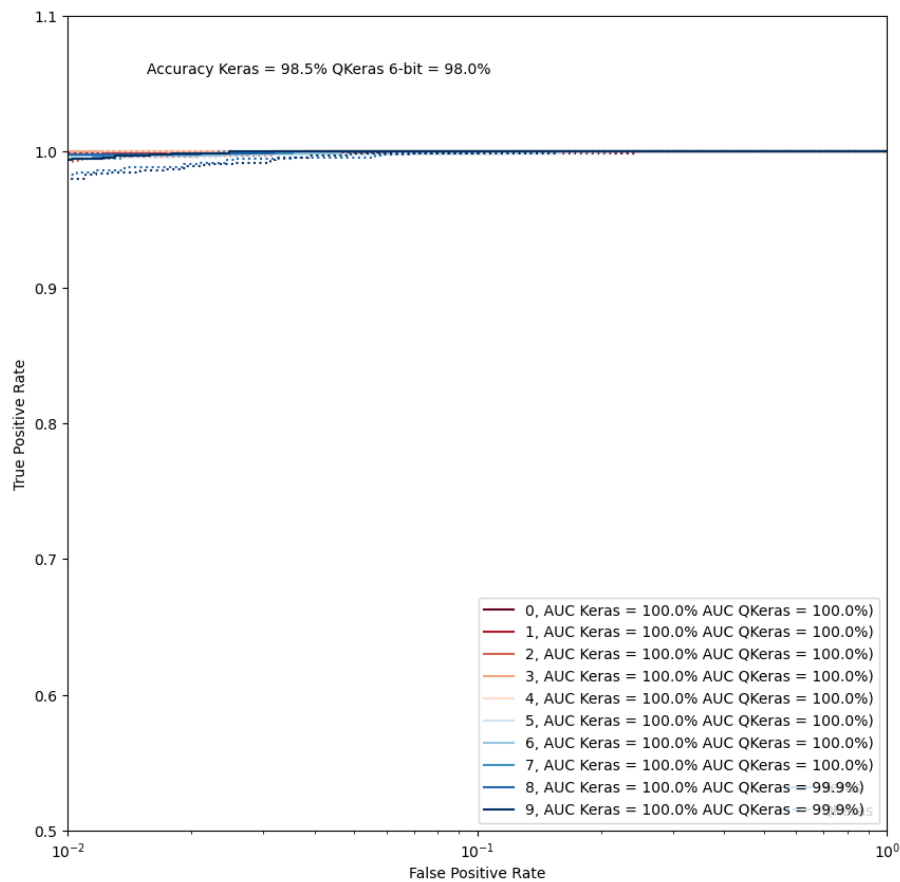
Πέρα από την ακρίβεια εκπαίδευσης και επικύρωσης, δημιουργήθηκαν καμπύλες ROC (Receiver Operating Characteristic) για την αξιολόγηση της απόδοσης του ταξινομητή σε διαφορετικά κατώφλια απόφασης. Η ανάλυση εστιάζει στις κβαντισμένες/κλαδεμένες εκδοχές, καθώς αποτελούν τη βασική βελτιστοποίηση υπέρ της υλοποίησης.

#### Ακρίβεια Βασικού vs. Κβαντισμένου Μοντέλου

Στο σύνολο δοκιμής, το βασικό μοντέλο Keras πέτυχε ακρίβεια **98,48%**, ενώ το κβαντισμένο QKeras (6-bit σταθερού σημείου) **98,02%**. Η πτώση κατά 0,46 ποσοστιαίες μονάδες αντικατοπτρίζει το αναμενόμενο penalty της κβάντισης, με αποτέλεσμα που παραμένει ανταγωνιστικό για υλοποίηση σε FPGA.

#### Επίδραση Κλαδέματος

Τα Σχήματα 5.1.5–5.1.7 παρουσιάζουν τις καμπύλες ROC που συγκρίνουν βασικά, κβαντισμένα και μεταφρασμένα μέσω `hls4ml` μοντέλα. Το κλαδεμένο CNN διατηρεί ουσιαστικά ταυτόσημες τιμές AUC με το βασικό, επιβεβαιώνοντας ότι το κλάδεμα δεν επηρέασε την ικανότητα διάκρισης.



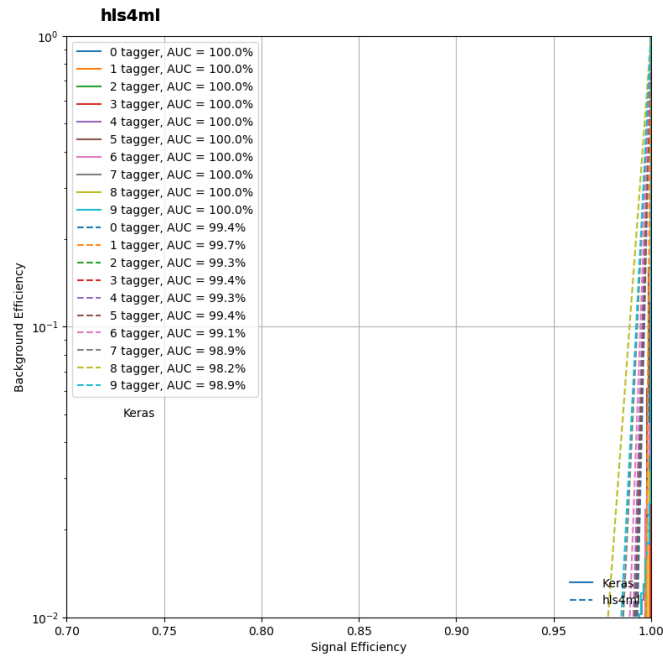
Σχήμα 5.1.5: Σύγκριση ROC μεταξύ βασικού Keras και κβαντισμένου QKeras

### Keras vs. QKeras

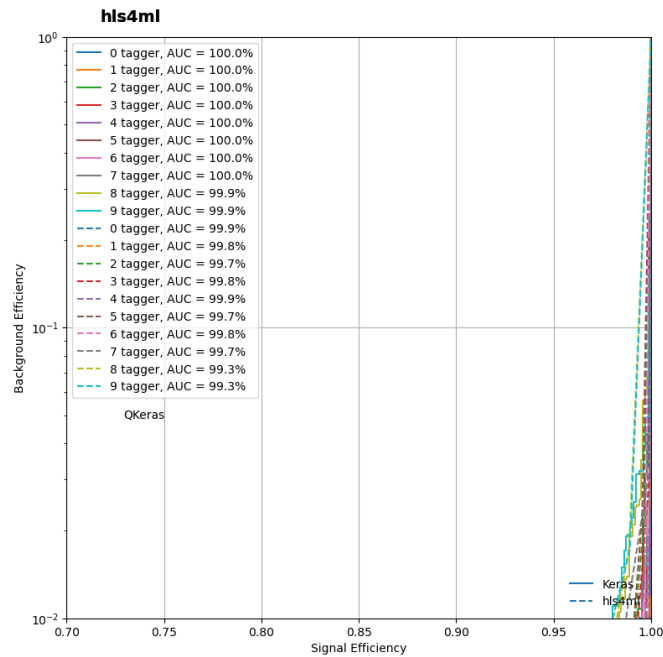
Το Σχήμα 5.1.5 δείχνει ότι και τα δύο μοντέλα επιτυγχάνουν σχεδόν τέλεια διάκριση, με AUC = 100%, παρά τη μικρή μείωση ακρίβειας του κβαντισμένου.

### QKeras vs. hls4ml Μετατροπή

Τα Σχήματα 5.1.7 και 5.1.6 συγκρίνουν τα μοντέλα πριν και μετά τη μετατροπή με hls4ml. Οι τιμές AUC παραμένουν πολύ κοντά (99,3–100%), δείχνοντας ότι η ροή HLS διατηρεί την ταξινομητική ικανότητα του αρχικού μοντέλου, οι βασικές επιδράσεις προέρχονται από κβάντιση/κλάδεμα, όχι από τη μετατροπή.



Σχήμα 5.1.6: ROC σύγκριση μεταξύ βασικού Keras μοντέλου και μοντέλου μετά τη μετατροπή σε hls4ml

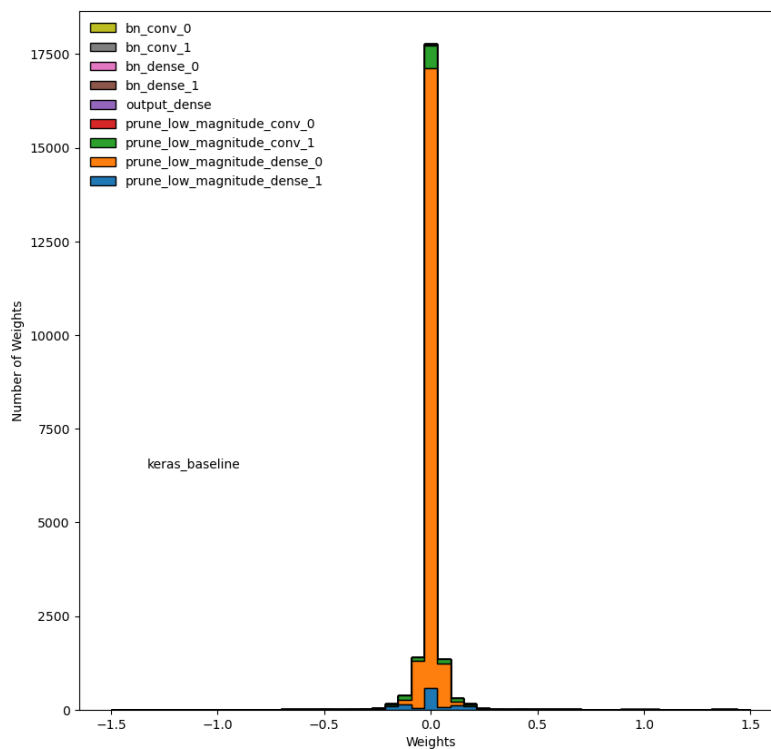


Σχήμα 5.1.7: ROC σύγκριση μεταξύ κβαντισμένου QKeras και μεταφρασμένου μοντέλου hls4ml

### 5.1.3 Ανάλυση Κατανομής Βαρών

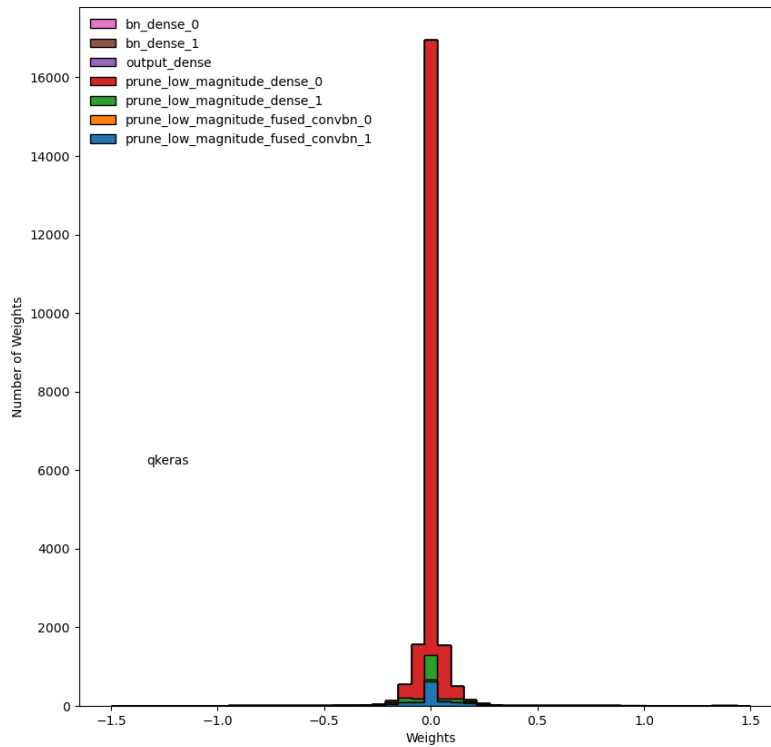
Για να διερευνηθεί περαιτέρω η επίδραση του κλαδέματος και της κβάντισης, αναλύεται η κατανομή των βαρών στα στρώματα του δικτύου. Τα ιστογράμματα δείχνουν πόσα βάρη τείνουν στο μηδέν, κάτι που επηρεάζει άμεσα τη χρήση πόρων στην υλοποίηση FPGA.

Το Σχήμα 5.1.8 παρουσιάζει την κατανομή βαρών του βασικού μοντέλου Keras μετά το κλάδεμα. Τα πλήρως συνδεδεμένα στρώματα εμφανίζουν έντονη συγκέντρωση γύρω από το μηδέν, υποδεικνύοντας ότι μεγάλο ποσοστό συνδέσεων έχει αποκοπεί.



Σχήμα 5.1.8: Κατανομή βαρών ανά στρώμα για το βασικό μοντέλο Keras μετά το κλάδεμα

Σε σύγκριση, το Σχήμα 5.1.9 δείχνει το ιστόγραμμα για το κβαντισμένο QKeras μοντέλο. Και εδώ, το κλάδεμα μηδενίζει πολλά βάρη, ενώ τα μη μηδενικά συγκεντρώνονται σε διακριτά bins λόγω της κβάντισης.



Σχήμα 5.1.9: Κατανομή βαρών ανά στρώμα για το κβαντισμένο QKeras μοντέλο μετά το κλάδεμα

Συμπερασματικά, το κλάδεμα είναι ιδιαίτερα επιθετικό στα πλήρως συνδεδεμένα στρώματα, όπου η πλεονάζουσα πληροφορία είναι μεγαλύτερη, η κβάντιση ενισχύει το φαινόμενο μέσω διακριτοποίησης των παραμενόντων βαρών, μειώνοντας τόσο τις απαιτήσεις αποθήκευσης όσο και το κόστος χαρτογράφησης σε FPGA DSP slices. Ο συνδυασμός κλαδέματος και κβάντισης οδηγεί σε αραιό και φιλικό προς υλοποίηση μοντέλο, εξηγώντας γιατί η εκδοχή QKeras διατηρεί σχεδόν την ίδια ακρίβεια με το βασικό (βλ. 5.1.2), ενώ μειώνει αισθητά τον αριθμό παραμέτρων και τη χρήση LUT (βλ. 5.3.1).

Έχοντας αποτυπώσει την ακρίβεια και τη συμπεριφορά σύγκλισης των παραλλαγών, προχωρούμε στις εκτιμήσεις HLS, ώστε να διερευνηθούν έγκαιρα οι συμβιβασμοί καθυστέρησης-πόρων πριν την πλήρη υλοποίηση.

## 5.2 Εκτιμήσεις Επίδοσης HLS

Σε αυτή την ενότητα αναλύουμε τα αποτελέσματα από το στάδιο **Vitis HLS**, το οποίο παρέχει πρώιμες εκτιμήσεις τόσο για την επίδοση όσο και για τη χρήση πόρων πριν από την πλήρη RTL υλοποίηση. Παρότι οι αναφορές αυτές δεν είναι τελικές, καθώς το Vivado μπορεί να εφαρμόσει επιπλέον βελτιστοποιήσεις στη σύνθεση και στη δρομολόγηση, προσφέρουν πολύτιμη εικόνα για το αν ένα σχέδιο είναι πιθανό να «χωρά» στον στοχευόμενο FPGA και ποιοι συμβιβασμοί προκύπτουν μεταξύ καθυστέρησης, ρυθμού διεκπεραίωσης και χρήσης πόρων. Στην πράξη, αυτές οι εκτιμήσεις καθοδηγούν την επιλογή στρατηγικών κβάντισης, *reuse factors* και *optimization pragmas* (π.χ. pipeline, unroll, resource) πριν προχωρήσουμε στη δημιουργία *bitstream*.

### 5.2.1 Αναφορές Επίδοσης HLS

Ο Πίνακας 5.2.1 συνοψίζει τις εκτιμήσεις επίδοσης. Η στοχευόμενη περίοδος ρολογιού ορίστηκε σε 5 ns (200 MHz), ενώ οι εκτιμώμενες περίοδοι ρολογιού που πέτυχε το εργαλείο HLS κυμαίνονται από 4,28 ns έως 4,88 ns, αντιστοιχώντας σε μέγιστες συχνότητες ( $F_{\max}$ ) 205–234 MHz.

Οι τιμές καθυστέρησης λαμβάνονται απευθείας από τις αναφορές HLS, όπου ο ελάχιστος αριθμός κύκλων του run\_core πολλαπλασιάζεται με την εκτιμώμενη περίοδο ρολογιού. Έτσι, οι αναφερόμενες καθυστερήσεις αποτελούν τις «ακατέργαστες» προβλέψεις του Vitis HLS, χωρίς αναπροσαρμογή σε σταθερή συχνότητα.

Πίνακας 5.2.1: Εκτιμήσεις επίδοσης HLS ανά παραλλαγή σχεδίου (απευθείας από τις αναφορές Vitis HLS)

| Variant   | Target (ns) | Est. (ns) | Fmax (MHz) | Run Latency ( $\mu$ s) |
|-----------|-------------|-----------|------------|------------------------|
| Baseline  | 5,00        | 4,432     | 225,6      | 133,0                  |
| Optimized | 5,00        | 4,364     | 229,1      | 11,9                   |
| Quantized | 5,00        | 4,279     | 233,7      | 133,0                  |
| RF 0%     | 5,00        | 4,364     | 229,1      | 11,9                   |
| RF 6,25%  | 5,00        | 4,364     | 229,1      | 71,2                   |
| RF 12,5%  | 5,00        | 4,364     | 229,1      | 133,0                  |
| RF 25%    | 5,00        | 4,588     | 218,0      | 252,9                  |
| RF 50%    | 5,00        | 4,707     | 212,4      | 497,9                  |
| RF 100%   | 5,00        | 4,885     | 204,7      | 984,9                  |

### Παρατηρήσεις

- Η καθυστέρηση αυξάνεται απότομα όσο μεγαλώνει ο *reuse factor*: από κάτω από 12  $\mu$ s στις ιδιαίτερα παραλληλισμένες διαμορφώσεις (Optimized, RF 0%) έως σχεδόν 1 ms στην πλήρως σειριοποιημένη περίπτωση (RF 100%).
- Το **κβαντισμένο μοντέλο** παρουσιάζει παρόμοια καθυστέρηση με το βασικό, αλλά *χρησιμοποιεί περισσότερους DSPs* (βλ. Πίνακα 5.2.2), καθώς περισσότερες MAC πράξεις χαρτογραφούνται σε DSP slices υπό σταθερού σημείου αριθμητική.
- Η διαμόρφωση *Optimized* αντιπροσωπεύει το άκρο προσανατολισμένο στην επίδοση: πετυχαίνει τη μικρότερη καθυστέρηση από όλες τις σχεδιάσεις αλλά με τίμημα σημαντικά αυξημένη χρήση πόρων, αναδεικνύοντας τον συμβιβασμό ταχύτητας-έκτασης υλικού.

### 5.2.2 Χρήση Πόρων HLS

Ο Πίνακας 5.2.2 αναφέρει τις εκτιμήσεις χρήσης πόρων για κάθε παραλλαγή σε σχέση με τη χωρητικότητα της K26 FPGA (BRAM\_18K = 288, DSP= 1.248, FF= 234.240, LUT= 117.120, URAM= 64).

Πίνακας 5.2.2: Χρήση πόρων HLS στις παραλλαγές του CNN (Vitis HLS, πριν την υλοποίηση)

| Variant   | BRAM (%)    | DSP (%)     | FF (%)         | LUT (%)          |
|-----------|-------------|-------------|----------------|------------------|
| Baseline  | 148 (51,4%) | 59 (4,7%)   | 54.564 (23,3%) | 51.846 (44,3%)   |
| Optimized | 187 (64,9%) | 769 (61,6%) | 36.003 (15,4%) | 99.026 (84,6%)   |
| Quantized | 81 (28,1%)  | 126 (10,1%) | 34.325 (14,7%) | 58.662 (50,1%)   |
| RF=0%     | 59 (20,5%)  | 385 (30,8%) | 32.298 (13,8%) | 169.031 (144,3%) |
| RF=6,25%  | 90 (31,2%)  | 176 (14,1%) | 30.399 (13,0%) | 57.165 (48,8%)   |
| RF=12,5%  | 81 (28,1%)  | 126 (10,1%) | 29.693 (12,7%) | 58.726 (50,1%)   |
| RF=25%    | 77 (26,7%)  | 106 (8,5%)  | 30.128 (12,9%) | 59.779 (51,0%)   |
| RF=50%    | 78 (27,1%)  | 96 (7,7%)   | 30.311 (12,9%) | 58.710 (50,1%)   |
| RF=100%   | 83 (28,8%)  | 93 (7,5%)   | 30.652 (13,1%) | 58.584 (50,0%)   |

### Παρατηρήσεις

- Σε σύγκριση με το βασικό, το **κβαντισμένο μοντέλο** χρησιμοποιεί περισσότερους DSPs (από 59  $\rightarrow$  126, 4,7%  $\rightarrow$  10,1%) και περίπου σταθερό ποσοστό LUTs ( $\approx$ 50%), αντανakλώντας τη μεταφορά MAC πράξεων σε DSP slices στη σταθερού σημείου ροή.

- Ο **βελτιστοποιημένος σχεδιασμός** φτάνει στο 61,6% των DSPs και 84,6% των LUTs, ακόμη εφικτό αλλά κοντά σε κορεσμό, δείχνοντας ότι ο επιθετικός παραλληλισμός πιέζει τους πόρους.
- Η **διαμόρφωση RF 0%** υπερβαίνει αρχικά τη χωρητικότητα LUT (144,3%) στις εκτιμήσεις HLS. Ωστόσο, μετά τη σύνθεση στο Vivado, βελτιστοποιήσεις όπως αναδιάρθρωση FIFO/λογικής μείωσαν τη χρήση, επιτρέποντας την υλοποίηση στο FPGA (όπως φαίνεται στις αναφορές Vivado).
- Καθώς ο **reuse factor** αυξάνεται, η χρήση DSP μειώνεται (30,8% → 7,5%) και η χρήση LUT σταθεροποιείται γύρω στο 50%. Η καθυστέρηση, όμως, αυξάνεται σημαντικά (βλ. Πίνακα 5.2.1), επιβεβαιώνοντας τον κλασικό συμβιβασμό παραλληλισμού–πόρων.

### Σύνοψη

Τα αποτελέσματα HLS δείχνουν ότι, ενώ τα βασικά και κβαντισμένα CNN έχουν παρόμοια συμπεριφορά ως προς την καθυστέρηση, η κβάντιση αναδιαμορφώνει αισθητά τη χρήση πόρων, καθιστώντας τον σχεδιασμό πιο «DSP-κεντρικό». Οι βελτιστοποιήσεις και η ρύθμιση του *reuse* ανταλλάσσουν επιπλέον ταχύτητα με αποτύπωμα πόρων. Αυτές οι επισημάνσεις προετοιμάζουν το έδαφος για τη σύνθεση Vivado και την αξιολόγηση επί πλακέτας, όπου πρέπει να επικυρωθούν το *timing closure*, η δρομολόγηση και η ισχύς.

Οι παραπάνω εκτιμήσεις HLS επικυρώνονται πλέον σε επίπεδο υλοποίησης με **Vivado**, όπου αποτιμώνται τελικά η χρήση πόρων, το *timing closure* και η ισχύς μετά το *place-and-route*.

## 5.3 Αναφορές Vivado

Σε αυτή την ενότητα παρουσιάζουμε τις αναφορές μετά την υλοποίηση (*post-implementation*) που προέκυψαν από το **Vivado** για όλες τις παραλλαγές του CNN. Τα αποτελέσματα αυτά συμπληρώνουν τις εκτιμήσεις του HLS παρέχοντας τελική χρήση πόρων, *timing closure* και ανάλυση ισχύος μετά από σύνθεση, τοποθέτηση και δρομολόγηση στο K26 SoM. Μαζί, καθορίζουν αν οι προτεινόμενες διαμορφώσεις επιταχυντή είναι εφικτές στο υλικό και ποιοι συμβιβασμοί προκύπτουν ως προς πόρους, καθυστέρηση και ενεργειακή αποδοτικότητα.

### 5.3.1 Τελική Χρήση Πόρων

Ο Πίνακας 5.3.1 συνοψίζει την τελική χρήση πόρων μετά την υλοποίηση για όλες τις παραλλαγές σχεδίασης. Τα ποσοστά υπολογίζονται σε σχέση με τους συνολικά διαθέσιμους πόρους της συσκευής K26 (BRAM\_18K = 288, DSP= 1.248, FF= 234.240, LUT= 117.120).

Πίνακας 5.3.1: Χρήση πόρων μετά την υλοποίηση (Vivado) στις παραλλαγές CNN

| Variant   | LUT (%)        | FF (%)         | BRAM (%)   | DSP (%)       |
|-----------|----------------|----------------|------------|---------------|
| Baseline  | 32.749 (27,9%) | 58.427 (24,9%) | 46 (16,0%) | 60 (4,8%)     |
| Quantized | 21.928 (18,7%) | 29.926 (12,8%) | 26 (9,0%)  | 124 (9,9%)    |
| Optimized | 35.251 (30,1%) | 31.302 (13,4%) | 44 (15,3%) | 1.170 (93,8%) |
| RF 0%     | 63.456 (54,2%) | 31.286 (13,4%) | 17 (5,9%)  | 405 (32,4%)   |
| RF 6,25%  | 57.165 (48,8%) | 30.399 (13,0%) | 20 (6,9%)  | 176 (14,1%)   |
| RF 12,5%  | 58.726 (50,1%) | 29.693 (12,7%) | 18 (6,3%)  | 126 (10,1%)   |
| RF 25%    | 59.779 (51,0%) | 30.128 (12,9%) | 19 (6,6%)  | 106 (8,5%)    |
| RF 50%    | 58.710 (50,1%) | 30.311 (12,9%) | 19 (6,6%)  | 96 (7,7%)     |
| RF 100%   | 58.584 (50,0%) | 30.652 (13,1%) | 20 (6,9%)  | 93 (7,5%)     |

#### Παρατηρήσεις

- Ο **βελτιστοποιημένος σχεδιασμός** σχεδόν κορέννει το DSP fabric (94%), αντιπροσωπεύοντας τον μέγιστο επιτεύξιμο παραλληλισμό στην K26. Προσφέρει τη μικρότερη καθυστέρηση, αλλά αφήνει ελάχιστο περιθώριο για πρόσθετους επιταχυντές ή λογική.
- Το **κβαντισμένο μοντέλο** δείχνει την καλύτερη ισορροπία, καταναλώνοντας μόλις 19% των LUTs και 10% των DSPs διατηρώντας την ακρίβεια· πρόκειται για τον πιο αποδοτικό υποψήφιο για ανάπτυξη.
- Το **sweep του reuse factor** καταδεικνύει τον αναμενόμενο συμβιβασμό HLS. Η αύξηση του reuse μειώνει δραστικά τη ζήτηση DSP (από 32,4% σε 7,5%), ενώ LUT και FF μένουν περίπου σταθερά ( 50% και 13%). Ωστόσο, αυτό έχει κόστος στην καθυστέρηση (βλ. Πίνακα 5.2.1).
- Ενδιαφέρον έχει ότι η **διαμόρφωση RF 0%**, την οποία το HLS προέβλεπε ως μη εφικτή λόγω υπέρβασης LUT, υλοποιήθηκε επιτυχώς στο Vivado. Αυτό κατέστη δυνατό λόγω βελτιστοποιήσεων όπως αναδιάρθρωση FIFO κατά το place-and-route, δείχνοντας ότι οι εκτιμήσεις HLS μπορεί να είναι συντηρητικές.

### 5.3.2 Χρονισμός και Ισχύς από Vivado

#### Timing closure

Ο Πίνακας 5.3.2 συνοψίζει τα αποτελέσματα χρονισμού (*Worst Negative Slack, WNS*) για όλες τις υλοποιήσεις. Όλες οι σχεδιάσεις, εκτός από τη βελτιστοποιημένη, επιτυγχάνουν επιτυχές *timing closure* στη συχνότητα στόχο των 150 MHz. Η **βελτιστοποιημένη εκδοχή** συντέθηκε με υψηλότερη συχνότητα αναφοράς στα 200 MHz, όπου εμφανίζει αρνητικό περιθώριο ( $WNS = -0,24$  ns). Η απόκλιση αυτή δεν υποδηλώνει αστοχία, αλλά αντανακλά το φυσικό όριο παραλληλισμού και τη συμφόρηση δρομολόγησης λόγω της εκτεταμένης χρήσης DSP πόρων. Με βάση τα αποτελέσματα του Vivado, η μέγιστη σταθερή συχνότητα λειτουργίας ( $f_{max}$ ) του σχεδίου εκτιμάται λίγο κάτω από 200 MHz, υπερκαλύπτοντας έτσι τον αρχικό στόχο των 150 MHz.

Πίνακας 5.3.2: Σύνοψη χρονισμού Vivado ανά παραλλαγή

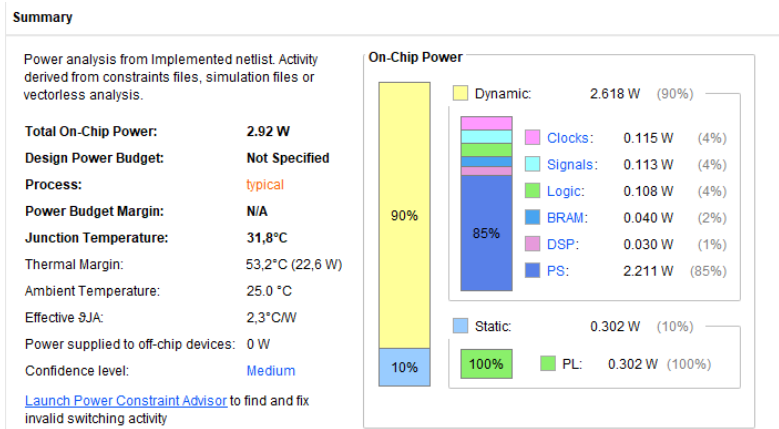
| Variant   | WNS (ns) | Κατάσταση Χρονισμού |
|-----------|----------|---------------------|
| Baseline  | 0,456    | Met                 |
| Quantized | 1,000    | Met                 |
| Optimized | -0,241   | Violated            |
| RF 0%     | 0,144    | Met                 |
| RF 6,25%  | 0,526    | Met                 |
| RF 12,5%  | 0,735    | Met                 |
| RF 25%    | 0,843    | Met                 |
| RF 50%    | 0,656    | Met                 |
| RF 100%   | 0,860    | Met                 |

#### Παρατήρηση

Το Vivado επιβεβαιώνει ότι οι περισσότερες διαμορφώσεις πετυχαίνουν άνετα τον στόχο των 150 MHz. Η βελτιστοποιημένη παραλλαγή, αν και ταχύτερη στις αναφορές HLS, αποτυγχάνει στο *timing closure* μετά την τοποθέτηση λόγω έντονης πίεσης δρομολόγησης, υπογραμμίζοντας την ανάγκη για ισορροπημένο σχεδιασμό και όχι μέγιστο παραλληλισμό.

### 5.3.3 Εκτίμηση Ισχύος από Vivado

Το Vivado παρέχει επίσης εκτιμήσεις ισχύος μετά την υλοποίηση. Το Σχήμα 5.3.1 δείχνει την ανάλυση για το **βασικό CNN**, με συνολική on-chip ισχύ 2,92 W. Η δυναμική ισχύς κυριαρχείται από το Processing System (PS), που αντιστοιχεί στο 85% του συνόλου. Το Programmable Logic (PL) συμβάλλει λιγότερο από 15% συνολικά, με ρολόγια, σήματα και λογική περίπου από 4% το καθένα, και DSPs μόλις 1%. Η στατική ισχύς εκτιμάται στα 0,30 W (10% του συνόλου).



Σχήμα 5.3.1: Εκτίμηση on-chip ισχύος (Vivado) για το βασικό CNN. Η δυναμική ισχύς κυριαρχείται από το PS (85%), ενώ τα στοιχεία του PL (DSP, λογική, BRAM) συνεισφέρουν λιγότερο από 15%

Ο Πίνακας 5.3.3 συνοψίζει τη συνολική *on-chip* ισχύ που εκτιμήθηκε από το Vivado για όλες τις παραλλαγές του CNN, καθώς και την ποσοστιαία συνεισφορά του δυναμικού μέρους και ειδικότερα των DSP στη συνολική κατανάλωση. Παρατηρείται ότι η πλήρως βελτιστοποιημένη σχεδίαση, προσανατολισμένη στη μέγιστη απόδοση (χαμηλότερη καθυστέρηση), εμφανίζει αισθητά αυξημένη ισχύ (4,27 W), γεγονός που οφείλεται στον έντονο παραλληλισμό και στη σχεδόν πλήρη αξιοποίηση των DSP slices. Αντίθετα, οι υπόλοιπες διαμορφώσεις διατηρούν παρόμοια συνολική ισχύ στο εύρος 2,8–3,0 W, με τη δυναμική συνιστώσα να κυμαίνεται γύρω στο 90% του συνόλου και τους DSP να καταναλώνουν μόλις 1–4% αυτής.

Πίνακας 5.3.3: Εκτίμηση ισχύος (Vivado) στις παραλλαγές CNN.

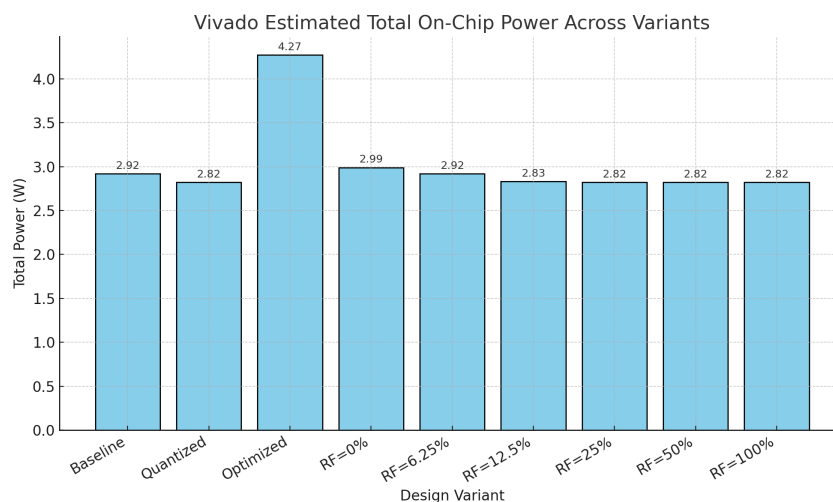
| Variant   | Total Power (W) | Dynamic (%) | DSP Share of Dynamic (%) |
|-----------|-----------------|-------------|--------------------------|
| Baseline  | 2,92            | 90          | 1                        |
| Quantized | 2,82            | 90          | 2                        |
| Optimized | 4,27            | 93          | 22                       |
| RF 0%     | 2,99            | 90          | 3                        |
| RF 6,25%  | 2,92            | 90          | 4                        |
| RF 12,5%  | 2,83            | 89          | 1                        |
| RF 25%    | 2,82            | 89          | 1                        |
| RF 50%    | 2,82            | 89          | 1                        |
| RF 100%   | 2,82            | 89          | 1                        |

### Παρατηρήσεις

- **Κυριαρχία PS:** Σε όλες τις παραλλαγές, το 80–87% της δυναμικής ισχύος καταναλώνεται από το PS, πράγμα που σημαίνει ότι οι βελτιστοποιήσεις στο επίπεδο PL

(κβάντιση, reuse) επηρεάζουν μόνο το 10–20% της συνολικής ισχύος.

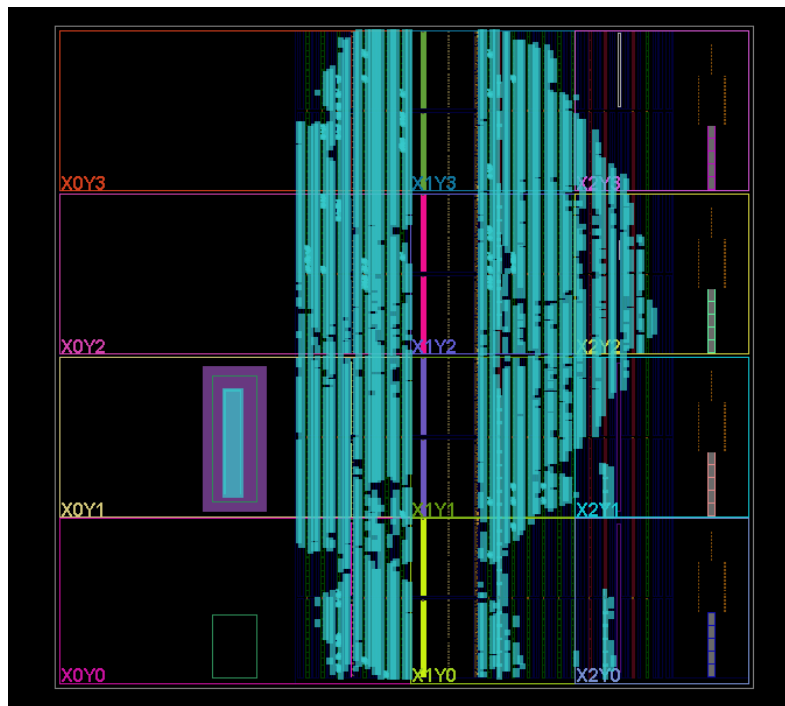
- **Βελτιστοποιημένη σχεδίαση:** Ο μεγαλύτερος καταναλωτής ισχύος (4,27 W), όπου οι DSPs μόνοι τους αντιστοιχούν στο 22% της δυναμικής ισχύος, επιβεβαιώνοντας το κόστος του επιθετικού παραλληλισμού.
- **Κβαντισμένες και υψηλού reuse διαμορφώσεις:** Επιτυγχάνουν την καλύτερη αποδοτικότητα, με τους DSPs να συνεισφέρουν κάτω από 2–4% της δυναμικής ισχύος, ενώ η συνολική ισχύς παραμένει γύρω στα 2,8–2,9 W.
- **Μικρή διασπορά στη συνολική ισχύ:** Οι περισσότερες σχεδιάσεις βρίσκονται στο εύρος 2,82–2,99 W (βλ. Σχήμα 5.3.2), δείχνοντας ότι ακόμη και μεγάλες διαφορές σε DSP/LUT έχουν περιορισμένη επίδραση στη συνολική ισχύ λόγω του overhead του PS.



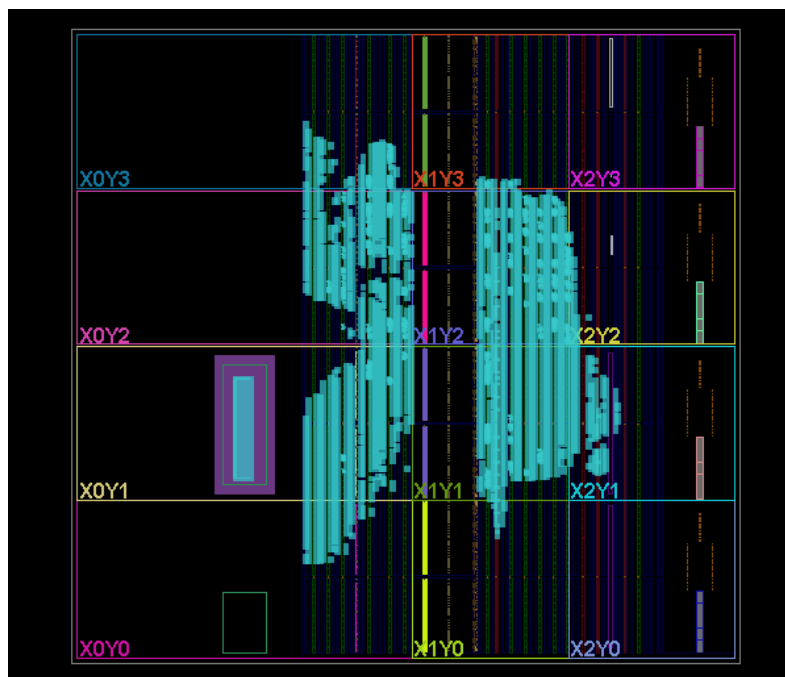
Σχήμα 5.3.2: Ιστόγραμμα της συνολικής on-chip ισχύος στις παραλλαγές CNN

### 5.3.4 Αποτύπωμα Τοποθέτησης και Δρομολόγησης

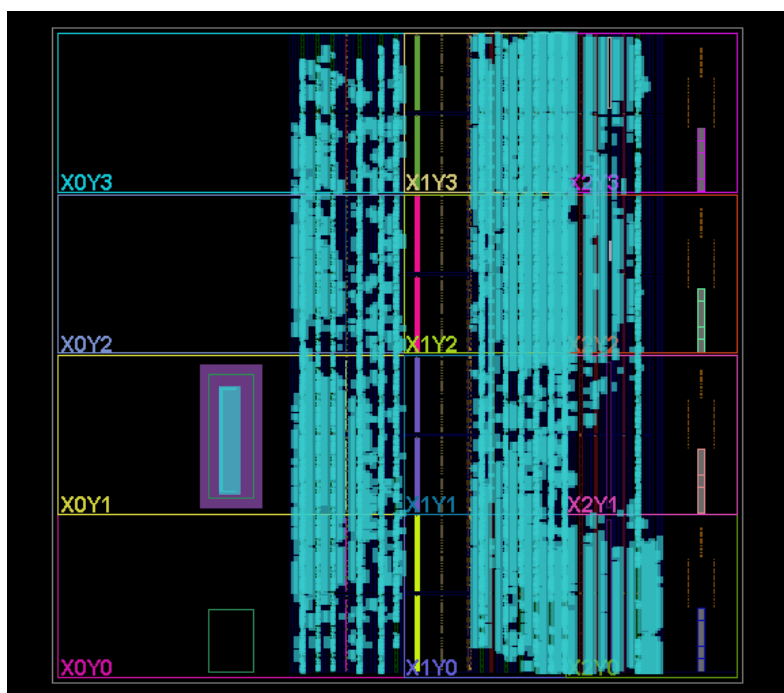
Τα Σχήματα 5.3.3–5.3.5 απεικονίζουν την τοποθέτηση και δρομολόγηση του επιταχυντή CNN για τις παραλλαγές *baseline*, *quantized* και *optimized* αντίστοιχα. Η κατανομή των κυψελών (LUTs, DSPs, BRAMs) στο πλέγμα της K26 παρέχει οπτική ένδειξη για την πυκνότητα και την περιοχή που καταλαμβάνει κάθε σχεδίαση.



Σχήμα 5.3.3: Αποτύπωμα τοποθέτησης για το **baseline** CNN



Σχήμα 5.3.4: Αποτύπωμα τοποθέτησης για το **quantized** CNN



Σχήμα 5.3.5: Αποτύπωμα τοποθέτησης για το **optimized** CNN.

### Συγκριτική Ανάλυση Τοποθέτησης

Η σύγκριση των αποτυπωμάτων τοποθέτησης αποκαλύπτει τη χωρική επίδραση των διαφορετικών επιλογών παραμετροποίησης:

- Το **baseline** μοντέλο καταλαμβάνει περιορισμένη περιοχή στο κέντρο του PL, αφήνοντας μεγάλο ποσοστό ανεκμετάλλευστων πόρων, χαρακτηριστικό ενός συντηρητικά παραλληλισμένου σχεδιασμού.
- Το **κβαντισμένο μοντέλο** παρουσιάζει σημαντική συμπίεση λογικής, με πιο συγκεντρωμένες περιοχές DSP και BRAM. Αυτή η συρρίκνωση του φυσικού αποτυπώματος οφείλεται στην κβάντιση και το κλάδεμα, που μειώνουν τις ενεργές συνδέσεις και επιτρέπουν πιο αποδοτική χαρτογράφηση.
- Το **βελτιστοποιημένο μοντέλο** απλώνεται σχεδόν σε ολόκληρο το πλέγμα της K26, φτάνοντας σε υψηλή πυκνότητα DSP και LUT, γεγονός που εξηγεί την αποτυχία του timing closure ( $WNS < 0$ ). Η κατανομή αυτή επιβεβαιώνει ότι ο ακραίος παραλληλισμός οδηγεί σε συμφόρηση δρομολόγησης, παρότι αποδίδει την ελάχιστη καθυστέρηση.

Συνολικά, τα αποτυπώματα αυτά απεικονίζουν καθαρά τον συμβιβασμό μεταξύ απόδοσης και χρήσης πόρων: όσο αυξάνεται ο παραλληλισμός, η σχεδίαση καταλαμβάνει περισσότερη επιφάνεια και επιβαρύνει το routing, μειώνοντας το περιθώριο χρονισμού.

Τέλος, τα σχέδια αναπτύσσονται στην πλατφόρμα **Kria KR260** για μετρήσεις καθυστέρησης, throughput και ισχύος, επιτρέποντας σύγκριση με τις εκτιμήσεις εργαλείων και με υλοποιήσεις CPU, GPU.

## 5.4 Μετρήσεις Επί Πλακέτας

Σε αυτή την ενότητα παρουσιάζουμε τις μετρημένες τιμές απόδοσης και ισχύος που προέκυψαν από την εκτέλεση των παραλλαγών του επιταχυντή CNN απευθείας στην πλατφόρμα Kria KR260. Η έμφαση δίνεται στους **λανθάνοντες χρόνους πυρήνα (core latencies)**, οι οποίοι επηρεάζονται άμεσα από τον *reuse factor* (RF), ενώ παράλληλα συγκρίνουμε ρυθμό διεκπεραίωσης (throughput), αποδοτικότητα και ισχύ με υλοποιήσεις αναφοράς σε GPU και CPU (βλ. Πίνακες 5.4.2 και 5.4.1).

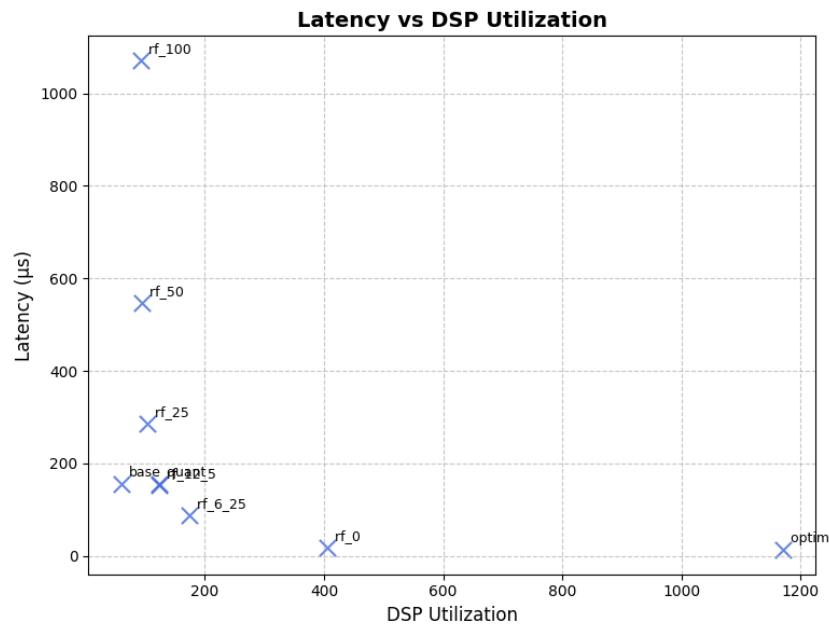
### 5.4.1 Τάσεις Καθυστέρησης και Ρυθμού Διεκπεραίωσης

Το Σχήμα 5.4.1 απεικονίζει τον μετρημένο λανθάνοντα χρόνο inference σε συνάρτηση με τη χρήση DSP για όλες τις υλοποιήσεις. Παρατηρείται ξεκάθαρος συμβιβασμός: **χαμηλοί reuse factors (RF 0, Optimized)** επιτυγχάνουν πολύ χαμηλή καθυστέρηση (δεκάδες  $\mu$ s) αλλά καταναλώνουν πολλούς DSPs, ενώ υψηλότεροι reuse factors μειώνουν τη ζήτηση σε DSP με κόστος την αύξηση της καθυστέρησης προς την περιοχή των ms. Ακόμη και η πιο αργή υλοποίηση (RF 100%) παραμένει σαφώς κάτω από 1.5 ms μαζί με τις καθυστερήσεις DDR, ικανοποιώντας τον περιορισμό πραγματικού χρόνου (βλ. επίσης Πίνακα 5.4.5 για τις χρονικές μετρικές).

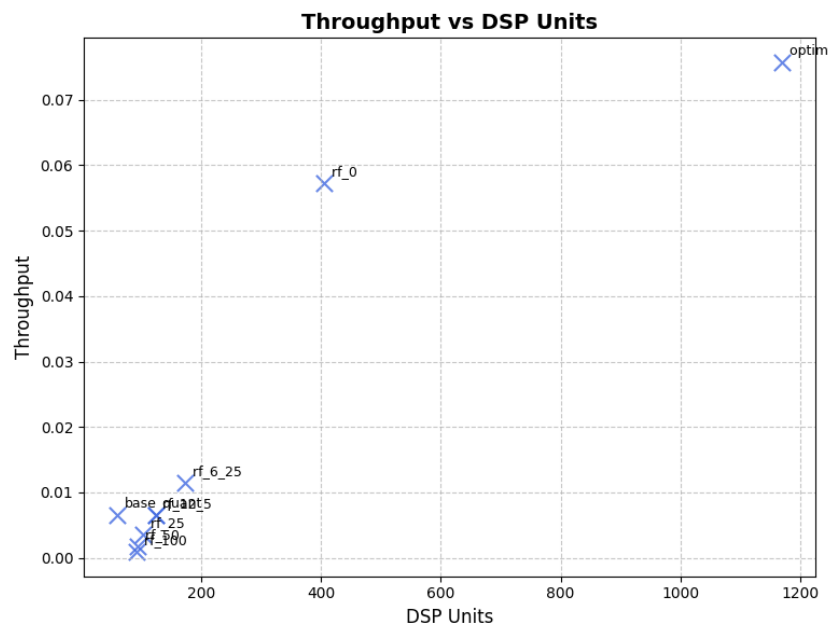
Ο ρυθμός διεκπεραίωσης είναι αντιστρόφως ανάλογος της καθυστέρησης. Το Σχήμα 5.4.2 δείχνει ότι πλήρως παραλληλισμένες σχεδιάσεις (Optimized, RF 0) φτάνουν σε δεκάδες χιλιάδες inferences ανά δευτερόλεπτο, ενώ οι σειριοποιημένες υλοποιήσεις (RF 50–100) πέφτουν κάτω από 2.000 1/s. Τα συνοπτικά νούμερα ανά υλοποίηση φαίνονται στον Πίνακα 5.4.1.

### 5.4.2 Μετρήσεις Ισχύος

Οι μετρήσεις ισχύος ελήφθησαν από τον ενσωματωμένο αισθητήρα INA260. Το Σχήμα 5.4.3 δείχνει ότι όλες οι υλοποιήσεις καταναλώνουν μεταξύ 3,4–3,8 W, με μικρές διαφορές μεταξύ των RF παραλλαγών. Αυτό ευθυγραμμίζεται με τις προσδοκίες, καθώς το *programmable logic* συμβάλλει λιγότερο στη συνολική ισχύ από το PS (βλ. Ενότητα 4.7). Οι μέσες τιμές ισχύος ανά εκτέλεση (Board και PL) συνοψίζονται στον Πίνακα 5.4.4.



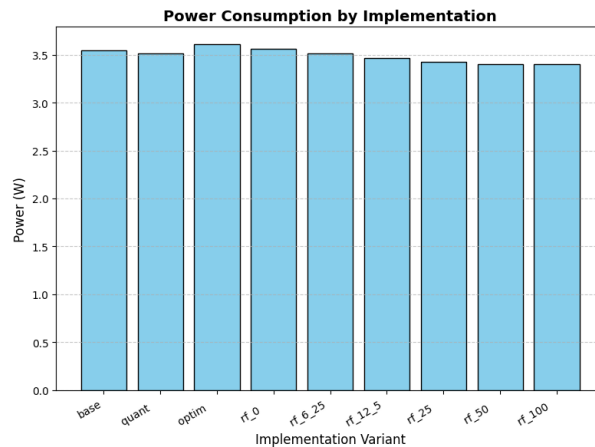
Σχήμα 5.4.1: Καθυστέρηση έναντι χρήσης DSP στις υλοποιήσεις FPGA



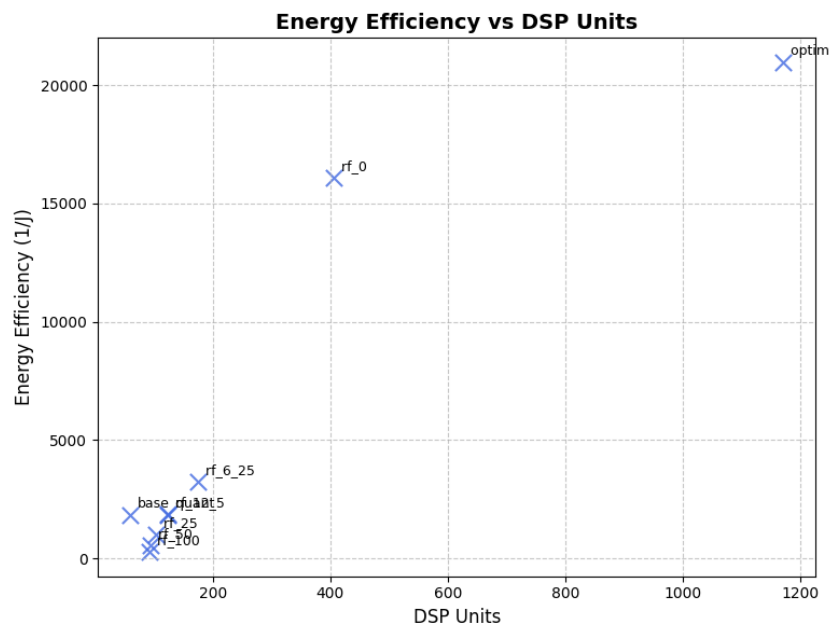
Σχήμα 5.4.2: Throughput έναντι χρήσης DSP

### 5.4.3 Ανάλυση Αποδοτικότητας

Τέλος, το Σχήμα 5.4.4 δείχνει την αποδοτικότητα σε συνάρτηση με τη χρήση DSP. Η βελτιστοποιημένη σχεδίαση επιτυγχάνει τη μέγιστη αποδοτικότητα ( $\eta \approx 20.966$  1/J), με τίμημα τον σχεδόν πλήρη κορεσμό των DSP. Μετριοπαθείς reuse factors (6,25–12,5%) προσφέρουν πιο ισορροπημένο συμβιβασμό, διατηρώντας αποδοτικότητα περίπου 1.800–3.000 1/J με σημαντικά χαμηλότερη ζήτηση σε DSP (βλ. Πίνακα 5.4.1).



Σχήμα 5.4.3: Μετρημένη κατανάλωση ισχύος στις υλοποιήσεις FPGA



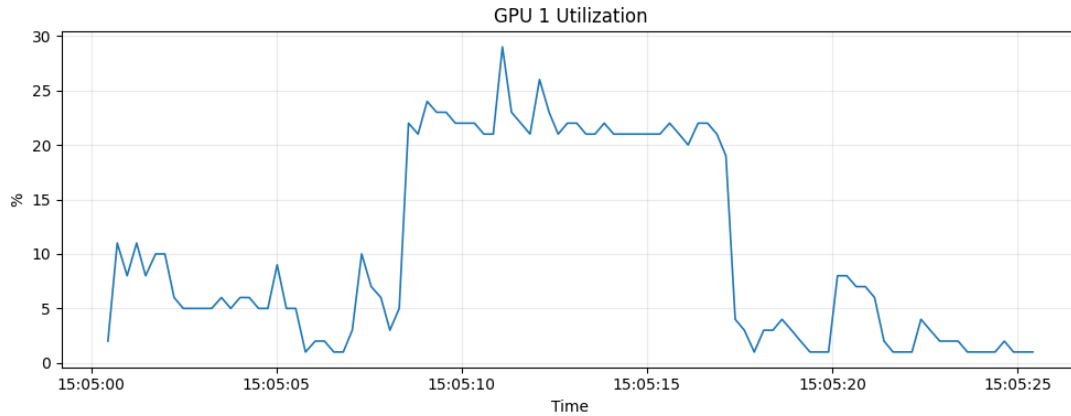
Σχήμα 5.4.4: Αποδοτικότητα (1/J) έναντι χρήσης DSP

#### 5.4.4 Ανάλυση Ισχύος και Χρήσης GPU και FPGA

Για τη σύγκριση της κατανάλωσης ενέργειας μεταξύ των πλατφορμών, συλλέχθηκαν χρονοσειρές ισχύος και χρήσης επεξεργαστικών πόρων κατά τη διάρκεια εκτέλεσης του ίδιου CNN μοντέλου σε GPU και FPGA. Η GPU παρακολουθήθηκε μέσω του **AMD Adrenalin driver suite**, το οποίο παρέχει τηλεμετρία σε πραγματικό χρόνο για ποσοστό χρήσης και κατανάλωση ισχύος, ενώ στην FPGA χρησιμοποιήθηκαν οι ενσωματωμένοι αισθητήρες INA260 (βλ. Ενότητα 4.7).

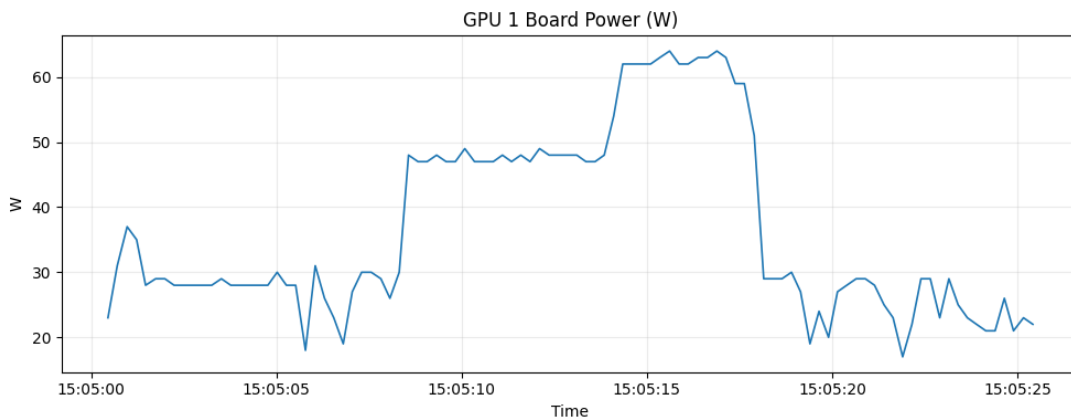
Το Σχήμα 5.4.5 δείχνει το ποσοστό χρήσης της GPU κατά την εκτέλεση inference. Παρατηρείται ότι η χρήση κυμαίνεται γύρω στο 20–25% με παροδικές αυξήσεις έως και

30%, γεγονός που υποδηλώνει περιορισμένη παράλληλη αξιοποίηση λόγω του μικρού μεγέθους του μοντέλου (MNIST CNN).



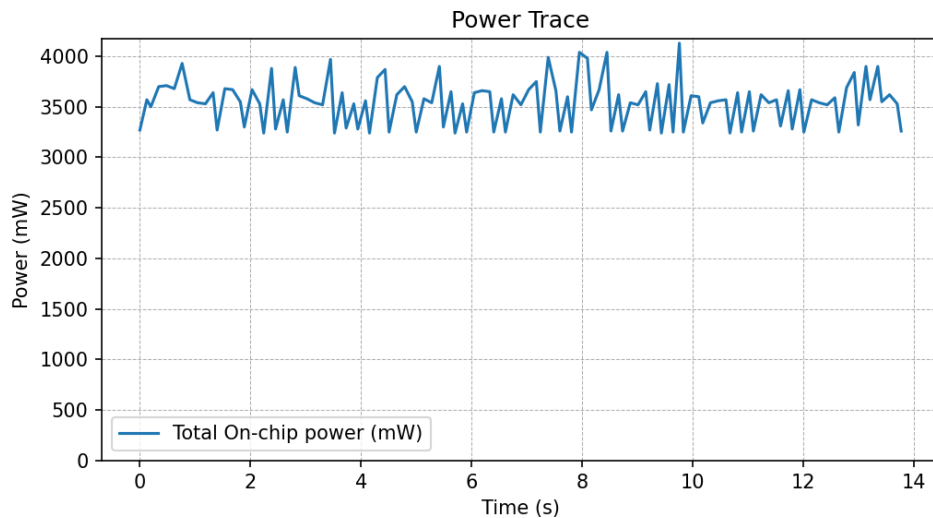
Σχήμα 5.4.5: Ποσοστό χρήσης GPU (AMD Radeon RX 7800 XT) κατά την εκτέλεση inference

Αντίστοιχα, το Σχήμα 5.4.6 παρουσιάζει τη χρονοσειρά κατανάλωσης ισχύος της GPU. Κατά τη φάση inference, η κατανάλωση αυξάνεται στα 55–65 W, ενώ σε idle καταστάσεις υποχωρεί κάτω από 25 W. Η μεγάλη διακύμανση αντικατοπτρίζει τη φύση των παρτίδων δεδομένων (batching) και τον δυναμικό έλεγχο ισχύος του οδηγού.



Σχήμα 5.4.6: Κατανάλωση ισχύος GPU (Board Power) σε Watt κατά την εκτέλεση inference

Το Σχήμα 5.4.7 δείχνει τη χρονοσειρά της post silicon ισχύος που μετρήθηκε στην FPGA μέσω του αισθητήρα INA260. Η μπλε γραμμή απεικονίζει την total on-chip ισχύ. Παρατηρείται ότι η συνολική κατανάλωση παραμένει σταθερή γύρω στα 3,5–3,8 W.



Σχήμα 5.4.7: Χρονοσειρά ισχύος FPGA (Kria KR260) όπως μετρήθηκε από τον αισθητήρα INA260

Η σύγκριση των Σχημάτων 5.4.6 και 5.4.7 αναδεικνύει τη διαφορά κλίμακας κατανάλωσης: η GPU καταναλώνει περίπου 20 φορές περισσότερη ισχύ από το FPGA για παρόμοιο υπολογιστικό έργο, επιβεβαιώνοντας την ανώτερη ενεργειακή αποδοτικότητα της πλατφόρμας **Kria KR260**.

#### 5.4.5 Σύνοψη Αποτελεσμάτων

Στο τέλος της πειραματικής διαδικασίας, συγκεντρώθηκαν όλα τα μετρούμενα δεδομένα από τις μετρήσεις επί πλακέτας, ώστε να εξαχθεί μια συνολική εικόνα των επιδόσεων και της αποδοτικότητας κάθε παραλλαγής. Οι Πίνακες 5.4.1–5.4.5 συνοψίζουν τα κύρια αποτελέσματα, παρέχοντας συγκριτική ανάλυση μεταξύ παραλλαγών FPGA, καθώς και με πλατφόρμες CPU και GPU.

Ο Πίνακας 5.4.1 παρουσιάζει τα συνολικά μετρικά επίδοσης για κάθε παραλλαγή FPGA, περιλαμβάνοντας την κατανάλωση ισχύος, τη λανθάνουσα καθυστέρηση (latency), τον ρυθμό διεκπεραίωσης (throughput) και την ενεργειακή αποδοτικότητα ( $\eta$ ). Παρατηρείται καθαρός συμβιβασμός (*trade-off*) μεταξύ καθυστέρησης και χρήσης πόρων: οι σχεδιάσεις με χαμηλό *reuse factor* (π.χ. Optimized, RF 0%) επιτυγχάνουν εξαιρετικά μικρή καθυστέρηση (της τάξης των δεκάδων  $\mu$ s) και πολύ υψηλό throughput (πάνω από 70 k 1/s), εις βάρος όμως της αυξημένης κατανάλωσης DSP και ελαφρώς υψηλότερης ισχύος. Αντίθετα, οι παραλλαγές με μεγάλο *reuse factor* παρουσιάζουν χαμηλότερη κατανάλωση και χρήση πόρων, αλλά η καθυστέρηση αυξάνεται σχεδόν γραμμικά, φτάνοντας έως 1 ms. Η συνολική ενεργειακή αποδοτικότητα παραμένει εντός του εύρους 500–20,000 1/J, με κορυφαία την πλήρως βελτιστοποιημένη υλοποίηση.

Πίνακας 5.4.1: Σύνοψη επίδοσης: ισχύς, καθυστέρηση, throughput και αποδοτικότητα για όλες τις υλοποιήσεις FPGA.

| Model     | Power (W) | Latency (ms) | Throughput (1/s) | Energy Efficiency (1/J) |
|-----------|-----------|--------------|------------------|-------------------------|
| Baseline  | 3,541     | 0,1544       | 6.476,68         | 1.828,57                |
| Quantized | 3,512     | 0,1544       | 6.475,29         | 1.843,44                |
| Optimized | 3,610     | 0,0132       | 75.700,23        | 20.966,74               |
| RF 0%     | 3,558     | 0,0174       | 57.339,45        | 16.111,27               |
| RF 6,25%  | 3,512     | 0,0875       | 11.422,48        | 3.251,50                |
| RF 12,5%  | 3,465     | 0,1536       | 6.509,00         | 1.878,04                |
| RF 25%    | 3,421     | 0,2855       | 3.502,79         | 1.023,67                |
| RF 50%    | 3,402     | 0,5466       | 1.829,42         | 537,63                  |
| RF 100%   | 3,402     | 1,0713       | 933,46           | 274,33                  |

Ο Πίνακας 5.4.2 παρουσιάζει τη σύγκριση του ίδιου *baseline* μοντέλου CNN σε τρεις διαφορετικές αρχιτεκτονικές: FPGA, GPU και CPU.

Η **GPU** επιτυγχάνει τη μικρότερη καθυστέρηση ανά δείγμα (0,033 ms) και το υψηλότερο throughput (περίπου 30k 1/s), ωστόσο συνοδεύεται από πολύ υψηλή κατανάλωση ισχύος (52,9 W σε επίπεδο πλακέτας και 25 W στον επιταχυντή).

Το **FPGA** εμφανίζει σημαντικά χαμηλότερη κατανάλωση (3,5 W συνολικά, εκ των οποίων μόλις 0,28 W στο PL), διατηρώντας παράλληλα ρυθμό 6,5k 1/s και καθυστέρηση 0,15 ms ανά inference. Παρά τη χαμηλότερη ακατέργαστη ταχύτητα, η ενεργειακή του αποδοτικότητα φτάνει τα 1.828 1/J, περίπου  $3,2\times$  υψηλότερη από τη GPU (574 1/J).

Η **CPU**, παρότι επιτυγχάνει πολύ χαμηλό χρόνο επεξεργασίας ανά δείγμα (0,02 ms) και υψηλό ρυθμό διεκπεραίωσης (28k 1/s), δεν αποτελεί κατάλληλη επιλογή για συνεχή ή έντονα παράλληλη λειτουργία. Η απουσία εξειδικευμένων επιταχυντών υλικού περιορίζει την απόδοσή της σε εφαρμογές υψηλού φόρτου, ενώ η λειτουργία σε μέγιστη συχνότητα συνεπάγεται σημαντική κατανάλωση ισχύος.

Συνολικά, το **FPGA** προσφέρει τη βέλτιστη ισορροπία μεταξύ καθυστέρησης, κατανάλωσης και αποδοτικότητας, καθιστώντας το καταλληλότερη επιλογή για εφαρμογές *edge AI*.

Ο Πίνακας 5.4.3 παρουσιάζει τις μετρικές ταξινόμησης του ίδιου μοντέλου πριν και μετά την κβάντιση. Η σύγκριση δείχνει ότι το κβαντισμένο CNN διατηρεί πολύ υψηλή ακρίβεια (98,0%) σε σχέση με το *baseline* (98,5%), με αμελητέες αποκλίσεις στις μετρικές *precision*, *recall* και *F1-score*. Η διαφορά μεταξύ του FP32 (PTQ) μοντέλου και της υλοποίησης σε FPGA είναι πρακτικά μηδενική ( $\Delta\text{accuracy} \approx 0,02\%$ ), επιβεβαιώνοντας ότι η ροή HLS δεν αλλοιώνει τη λειτουργικότητα του δικτύου. Επιπλέον, το κβαντισμένο μοντέλο με (QAT) διατήρησε πλήρως την ευστοχία του, γεγονός που το καθιστά ιδανικό για υλοποιήσεις σε FPGA με περιορισμένους πόρους.

Πίνακας 5.4.2: Σύγκριση μετρικών του ίδιου baseline μοντέλου σε επιταχυντές FPGA, GPU και CPU.

| Metric                | FPGA    | GPU       | CPU       |
|-----------------------|---------|-----------|-----------|
| Latency (ms)          | 0,1544  | 0,0329    | 0,0195    |
| Throughput (1/s)      | 6.476,6 | 30.395,14 | 51.282,19 |
| Efficiency (1/J)      | 1.828,5 | 574,11    | n/a       |
| Board Power (W)       | 3,541   | 52,94     | n/a       |
| Accelerator Power (W) | 0,284   | 25,04     | n/a       |

Πίνακας 5.4.3: Σύγκριση μετρικών ταξινόμησης μεταξύ κβαντισμένου και βασικού μοντέλου.

| Metrics                          | Baseline | Quantized |
|----------------------------------|----------|-----------|
| Accuracy (CPU)                   | 98,48%   | 98,02%    |
| Accuracy (HW)                    | 98,5%    | 98,02%    |
| $\Delta(\text{HW} - \text{CPU})$ | +0,02%   | +0,0%     |
| Precision                        | 0,9851   | 0,9806    |
| Recall                           | 0,9849   | 0,9800    |
| F1 Score                         | 0,9849   | 0,9802    |

Ο Πίνακας 5.4.4 παρουσιάζει τις μέσες τιμές **on-chip κατανάλωσης ισχύος** (*PL Power*) που μετρήθηκαν μέσω του αισθητήρα **INA260** στην πλατφόρμα *Kria KR260*. Ο αισθητήρας καταγράφει την ισχύ που καταναλώνεται από το *Programmable Logic (PL)*, χωρίς να περιλαμβάνει τη συμβολή της μνήμης *BRAM*. Παρατηρείται ότι η κατανάλωση ισχύος παραμένει σχεδόν σταθερή γύρω στα 3,5 W για όλες τις παραλλαγές, γεγονός που υποδεικνύει ότι οι διαφοροποιήσεις στις παραμέτρους HLS (όπως ο *reuse factor*) δεν επηρεάζουν σημαντικά τη συνολική ενεργειακή συμπεριφορά του συστήματος, αλλά μόνο την τοπική δραστηριότητα της λογικής.

Τέλος, ο Πίνακας 5.4.5 παρουσιάζει τις αναλυτικές χρονικές μετρικές για τις βασικές εκδόσεις του επιταχυντή. Διακρίνονται δύο μορφές καθυστέρησης: η *inference latency*, που αντιστοιχεί μόνο στην φάση συμπερασμού, και η *total latency*, η οποία περιλαμβάνει επιπλέον την καθυστέρηση μεταφοράς δεδομένων από την μνήμη στο DDR στο DMA και AXI διαύλων. Τα αποτελέσματα δείχνουν ότι η καθυστέρηση του συμπερασμού κυμαίνεται από 175–220  $\mu\text{s}$ , ενώ η συνολική καθυστέρηση ανέρχεται περίπου στα 450–500  $\mu\text{s}$ . Αυτή η διαφορά επιβεβαιώνει το επικοινωνιακό overhead μεταξύ του PL και του PS, το οποίο είναι αναμενόμενο στις SoC πλατφόρμες.

Πίνακας 5.4.4: Μέσες τιμές on-chip κατανάλωσης ισχύος (PL Power) ανά παραλλαγή, σε Watt.

| Model     | On-Chip Power (W) |
|-----------|-------------------|
| Baseline  | 3,541             |
| Quantized | 3,512             |
| Optimized | 3,610             |
| RF 0%     | 3,558             |
| RF 6,25%  | 3,512             |
| RF 12,5%  | 3,465             |
| RF 25%    | 3,421             |
| RF 50%    | 3,402             |
| RF 100%   | 3,402             |

Πίνακας 5.4.5: Μετρικές του κανονικού και του πλήρως βελτιστοποιημένου μοντέλου (χρονικές μετρικές).

| Metric                       | Baseline | Quantized | Optimized |
|------------------------------|----------|-----------|-----------|
| Total Latency ( $\mu$ s)     | 493,463  | 494,870   | 453,490   |
| Inference Latency ( $\mu$ s) | 218,388  | 217,868   | 175,343   |
| Total Throughput (1/s)       | 2.028,7  | 2.022,5   | 2.207,6   |
| Inference Throughput (1/s)   | 4.585,6  | 4.595,9   | 5.713,1   |

## Σύνοψη

Το Κεφάλαιο 5 παρουσίασε τη συνολική αξιολόγηση του επιταχυντή CNN, καλύπτοντας αποτελέσματα εκπαίδευσης, επίδοσης και κατανάλωσης σε πλατφόρμες **FPGA**, **GPU** και **CPU**. Η προτεινόμενη ροή **Keras/QKeras**  $\rightarrow$  **hls4ml**  $\rightarrow$  **Vitis HLS/Vivado**  $\rightarrow$  **Kria KR260** απέδειξε την αξιοπιστία και αποδοτικότητα της.

Τα βασικά συμπεράσματα συνοψίζονται ως εξής:

- Η **κβάντιση** και το **κλάδεμα** μείωσαν την πολυπλοκότητα του μοντέλου χωρίς σημαντική απώλεια ακρίβειας ( $\sim 98\%$ ).
- Οι **εκτιμήσεις HLS/Vivado** συμφωνούν ως ένα βαθμό με τις πραγματικές μετρήσεις, επιβεβαιώνοντας τη σταθερότητα της ροής.
- Οι **υλοποιήσεις FPGA** πέτυχαν καθυστέρηση κάτω από 1,5 ms και κατανάλωση περίπου 3,5 W, προσφέροντας έως και  $3\times$  καλύτερη ενεργειακή αποδοτικότητα από τη GPU.
- Η **GPU** παραμένει ταχύτερη σε ωμή επίδοση, αλλά με πολύ υψηλότερη κατανάλωση ισχύος.

Συνολικά, τα **FPGA** αναδεικνύονται ως αποδοτική και πρακτική λύση για inference πραγματικού χρόνου σε εφαρμογές *edge AI*, με ιδανική ισορροπία μεταξύ ακρίβειας, καθυστέρησης και κατανάλωσης ισχύος.

## Κεφάλαιο 6

# Συζήτηση, Συμπεράσματα και Μελλοντικές Βελτιώσεις

### 6.1 Συζήτηση Αποτελεσμάτων

Με βάση τις μετρήσεις και τις αναφορές που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, η παρούσα ενότητα συνοψίζει και ερμηνεύει τα αποτελέσματα, αναδεικνύοντας τους βασικούς συμβιβασμούς και τα πλεονεκτήματα της προτεινόμενης αρχιτεκτονικής FPGA CNN. Η συνολική ανάλυση δείχνει ότι η ροή υλοποίησης που προτάθηκε παρέχει μια ισορροπημένη προσέγγιση μεταξύ ακρίβειας, καθυστέρησης και χρήσης πόρων. Η μετάβαση από κινητής υποδιαστολής σε σταθερού σημείου, σε συνδυασμό με το κλάδεμα, είχε μικρή μόνο επίδραση στην απόδοση ταξινόμησης, ενώ προσέφερε σημαντικά οφέλη σε αποδοτικότητα του υλικού και την ενεργειακή κατανάλωση.

#### Ακρίβεια και Απόδοση Μοντέλων

Τα αποτελέσματα εκπαίδευσης έδειξαν ότι το βασικό μοντέλο Keras και το κλαδεμένο μοντέλο πέτυχαν ακρίβεια επικύρωσης άνω του 98%, ενώ το κβαντισμένο QKeras υποχώρησε οριακά στο 95%. Ο συνδυασμός κβάντισης και κλαδέματος επανέφερε την ακρίβεια κοντά στα επίπεδα του baseline, επιβεβαιώνοντας ότι οι δύο τεχνικές μπορούν να λειτουργήσουν συμπληρωματικά. Η ανάλυση ROC καμπυλών έδειξε ότι η ικανότητα διάκρισης (AUC) παραμένει πρακτικά αμετάβλητη μετά τη μετατροπή σε HLS, επιβεβαιώνοντας τη λειτουργική ισοδυναμία μεταξύ του λογισμικού και του υλικού.

#### Επίδοση και Πόροι από HLS

Οι αναφορές του Vitis HLS κατέδειξαν τον αναμενόμενο συμβιβασμό μεταξύ *reuse factor*, καθυστέρησης και της χρήσης DSP. Χαμηλές τιμές του *reuse factor* οδήγησαν σε

πλήρως παραλληλισμένες σχεδιάσεις με ελάχιστη καθυστέρηση (κάτω από 20  $\mu$ s), αλλά με αυξημένη κατανάλωση DSP έως και 61,6%. Αντίθετα, η αύξηση περιορίσε σημαντικά τη χρήση DSP (από 30,8% σε 7,5%), με κόστος την καθυστέρηση, η οποία αυξήθηκε έως περίπου 1,0 ms. Η συμπεριφορά αυτή συμφωνεί με τη θεωρητική ανάλυση του hls4ml και παρέχει στον σχεδιαστή ευελιξία για την επιλογή σημείου λειτουργίας ανάλογα με τους διαθέσιμους πόρους της συσκευής.

### Αποτελέσματα Υλοποίησης στο Vivado

Οι αναφορές Vivado επιβεβαίωσαν ότι όλες οι παραλλαγές, εκτός της πλήρως βελτιστοποιημένης (*Optimized*), κλείνουν άνετα τον χρονισμό στα 150 MHz. Η βελτιστοποιημένη έκδοση, αν και ταχύτερη στις εκτιμήσεις HLS (11,9  $\mu$ s core latency), παρουσίασε αρνητικό περιθώριο WNS λόγω συμφόρησης δρομολόγησης, γεγονός που υπογραμμίζει το πρακτικό όριο παραλληλισμού σε συσκευές με περιορισμένο DSP fabric, αλλά και την υψηλή χρήση f\_clk (<200MHz). Η χρήση των πόρων έδειξε ότι το κβαντισμένο μοντέλο είναι το πιο αποδοτικό, καταναλώνοντας μόλις 19% των LUTs και 10% των DSPs, ενώ διατηρεί πλήρη λειτουργικότητα.

### Μετρήσεις Επί Πλακέτας και Ενεργειακή Αποδοτικότητα

Οι μετρήσεις στην πλατφόρμα **Kria KR260** επιβεβαίωσαν τις προβλέψεις των εργαλείων. Η συνολική καθυστέρηση (*Total latency*) κυμαίνεται μεταξύ 450–495  $\mu$ s για τα βασικά μοντέλα, ενώ η καθυστέρηση του inference εκτελείται σε 175–220  $\mu$ s. Όλες οι παραλλαγές παραμένουν κάτω από 1,5 ms ακόμη και με καθυστερήσεις επικοινωνίας, ικανοποιώντας πλήρως τις απαιτήσεις πραγματικού χρόνου. Ο ρυθμός διεκπεραίωσης (throughput) εκτείνεται από περίπου 933 1/s για το RF 100% έως 75.700 1/s για το βελτιστοποιημένο μοντέλο, ενώ η **on-chip ισχύς** παραμένει σχεδόν σταθερή γύρω στα 3,5 W. Η αποδοτικότητα (1/J) κορυφώνεται στην υλοποίηση *Optimized* με περίπου 20.967 1/J, ενώ οι ενδιάμεσες τιμές reuse (6,25–12,5%) προσφέρουν έναν πιο ισορροπημένο συμβιβασμό, συνδυάζοντας καλή επίδοση με μέτρια χρήση πόρων.

### Σύγκριση με GPU

Η σύγκριση με την GPU δείχνει ότι, παρόλο που η τελευταία υπερτερεί σε ωμή ταχύτητα (καθυστέρηση περίπου 0,03 ms), το FPGA υπερέχει σημαντικά σε ενεργειακή αποδοτικότητα, επιτυγχάνοντας ενεργειακή αποδοτικότητα (throughput/power) έως και **1.828 1/J**, περίπου τριπλάσια σε σχέση με την GPU (574 1/J). Συνεπώς, το FPGA τοποθετείται μεταξύ των δύο άκρων, προσφέροντας σχεδόν πραγματικό χρόνο εκτέλεσης με δραματικά χαμηλότερη ισχύ, αποτελώντας ιδανική επιλογή για ενσωματωμένα και edge περιβάλλοντα.

## 6.2 Συμπεράσματα

Η εργασία αυτή απέδειξε στην πράξη ότι η υλοποίηση συνελκτικών νευρωνικών δικτύων (CNN) σε πλατφόρμες FPGA μέσω τεχνικών Υψηλού Επιπέδου Σύνθεσης (High-Level Synthesis, HLS) είναι όχι μόνο εφικτή, αλλά και ιδιαίτερα αποδοτική τόσο ως προς την καθυστέρηση όσο και ως προς την ενεργειακή κατανάλωση. Η προτεινόμενη ροή παρήγαγε λειτουργικές υλοποιήσεις που συνδυάζουν ακρίβεια κοντά στο μοντέλο λογισμικού με δραστική βελτίωση στην ενεργειακή αποδοτικότητα.

- **Απόδοση μοντέλου:** Η κβάντιση (Q6.10) και το κλάδεμα δεν επηρέασαν ουσιαστικά την ακρίβεια ταξινόμησης. Το τελικό κβαντισμένο–κλαδεμένο CNN πέτυχε ακρίβεια επικύρωσης 97,97%, σχεδόν ταυτόσημη με το baseline Keras (97,9%), αποδεικνύοντας ότι μοντέλα σταθερού σημείου μπορούν να διατηρήσουν υψηλή ταξινομητική επίδοση.
- **Εκτιμήσεις HLS και χρήση πόρων:** Οι αναφορές του Vitis HLS έδειξαν ότι ο *reuse factor* αποτελεί τον καθοριστικό παράγοντα ισορροπίας μεταξύ καθυστέρησης και κατανάλωσης DSP. Μικρός *reuse* (RF=0%) οδήγησε σε εξαιρετικά χαμηλή καθυστέρηση (~12  $\mu$ s) με χρήση 385 DSPs (30,8%), ενώ μεγάλος *reuse* (RF=100%) μείωσε τη χρήση DSP σε 93 (7,5%) με αντίτιμο καθυστέρηση ~1 ms. Το εύρος αυτό παρέχει ευελιξία επιλογής ανάλογα με τους διαθέσιμους πόρους της συσκευής.
- **Αποτελέσματα Vivado:** Όλες οι παραλλαγές εκτός της πλήρως βελτιστοποιημένης έκδοσης πέτυχαν επιτυχές *timing closure* στα 150 MHz. Το κβαντισμένο μοντέλο ξεχώρισε ως το πιο αποδοτικό, με μόλις 19% χρήση LUTs και 10% DSPs, διατηρώντας χαμηλή *on-chip* ισχύ (3,51 W). Η βελτιστοποιημένη παραλλαγή (*Optimized*) χρησιμοποίησε το 94% των DSPs και 30% των LUTs, αποδεικνύοντας το όριο παραλληλισμού της συσκευής K26.
- **Επίδοση επί πλακέτας:** Οι μετρήσεις στην πλατφόρμα Kria KR260 επιβεβαίωσαν τις εκτιμήσεις HLS/Vivado. Όλες οι υλοποιήσεις διατήρησαν καθυστέρηση κάτω από 1,5 ms ακόμη και στις σειριοποιημένες παραλλαγές, ενώ οι πλήρως παραλληλισμένες σχεδιάσεις πέτυχαν throughput έως **75.700 1/s**. Η αποδοτικότητα (throughput/Watt) κορυφώθηκε για τη βελτιστοποιημένη σχεδίαση με  $\eta \approx 2,1 \times 10^4$  (1/J), ενώ οι ενδιάμεσες διαμορφώσεις RF=6,25–12,5% προσέφεραν καλύτερη ισορροπία με 1.800–3.000 (1/J).
- **Ενεργειακή αποδοτικότητα:** Η *on-chip* ισχύς παρέμεινε σταθερή γύρω στα 3,5 W για όλες τις παραλλαγές. Η βελτιστοποιημένη έκδοση, παρότι αύξησε τη δραστηριότητα στους DSPs (περίπου 22% της συνολικής), πέτυχε τον υψηλότερο λόγο

**throughput/ισχύος**, με διαφορά άνω του  $10\times$  σε σχέση με την GPU και περίπου  $30\times$  έναντι της CPU.

- **Σύγκριση με GPU και CPU:** Η GPU (*Radeon RX 7800XT*, DirectML backend) εμφάνισε καθυστέρηση 0,033 ms και throughput 30.395 1/s με κατανάλωση 52,9 W (απόδοση 574 (1/J)). Η CPU (*Ryzen 9 7900X*) πέτυχε χαμηλή καθυστέρηση 0,0195 ms για μικρά batches αλλά με περιορισμένη κλιμάκωση. Αντίθετα, το FPGA πέτυχε 0,154 ms καθυστέρηση, throughput 6.476 1/s, κατανάλωση 3,54 W και αποδοτικότητα **1.828 (1/J)**, δηλαδή περίπου  **$3,2\times$  υψηλότερη ενεργειακή απόδοση** από την GPU, με  $15\times$  χαμηλότερη ισχύ λειτουργίας. Αυτό καθιστά το FPGA την πιο ισορροπημένη λύση για πραγματικού χρόνου inference σε *embedded* και *edge* εφαρμογές.

Συνοψίζοντας, το προτεινόμενο σύστημα απέδειξε ότι τα FPGAs μπορούν να υποστηρίξουν *real-time inference* με ακρίβεια επιπέδου λογισμικού και εξαιρετικά χαμηλή κατανάλωση ισχύος. Η επιτυχής μετάβαση του CNN από το περιβάλλον Keras/QKeras στο υλικό μέσω hls4ml και **Vitis HLS/Vivado** επιβεβαιώνει τη χρησιμότητα και την ωριμότητα της ροής HLS για σύγχρονες πλατφόρμες FPGA, όπως η *Kria KR260*, καθιστώντας την μια ισχυρή και επεκτάσιμη λύση για ενεργειακά αποδοτική επιτάχυνση AI στα άκρα.

## 6.3 Μελλοντικές Βελτιώσεις

Αν και η παρούσα εργασία απέδειξε την αποτελεσματικότητα της προτεινόμενης ροής ανάπτυξης CNN σε πλατφόρμες FPGA μέσω *High-Level Synthesis (HLS)*, υπάρχουν πολυάριθμες κατευθύνσεις για μελλοντική έρευνα και βελτίωση, τόσο σε επίπεδο αρχιτεκτονικής υλικού όσο και στη βελτιστοποίηση της ροής εργαλείων και των τεχνικών εκπαίδευσης.

**Επέκταση και Πολυπλοκότητα Μοντέλων** Η παρούσα μελέτη βασίστηκε στο σύνολο δεδομένων **MNIST**, το οποίο είναι κατάλληλο για benchmarking αλλά σχετικά απλό. Μελλοντικές εργασίες μπορούν να εστιάσουν σε πιο απαιτητικά σύνολα, όπως **CIFAR-10/100**, **Fashion-MNIST** ή **SVHN**, τα οποία περιλαμβάνουν χρωματικές εικόνες και αυξημένη διαστατικότητα εισόδου. Η υλοποίηση βαθύτερων συνελικτικών δικτύων (CNNs) ή ελαφρών αρχιτεκτονικών τύπου **MobileNet**, **ResNet** και **EfficientNet** θα επιτρέψει την αξιολόγηση της κλιμακωσιμότητας και των ορίων απόδοσης της προτεινόμενης ροής σε πραγματικά δεδομένα.

**Μεικτή Ακρίβεια και Προσαρμοστική Κβάντιση** Ενδιαφέρουσα κατεύθυνση αποτελεί η διερεύνηση τεχνικών **μεικτής ακρίβειας (mixed precision)** και **προσαρμοστικής κβάντισης (adaptive quantization)**. Διαφορετικά επίπεδα ακρίβειας (π.χ. Q4.8, Q6.10, Q8.8) μπορούν να εφαρμοστούν σε διαφορετικά στρώματα του δικτύου ανάλογα με την ευαισθησία τους στην ακρίβεια, μειώνοντας έτσι περαιτέρω τη χρήση DSPs και LUTs χωρίς σημαντική απώλεια απόδοσης. Παράλληλα, η ενσωμάτωση *post-training quantization (PTQ)* και *quantization-aware training (QAT)* σε αυτοματοποιημένη ροή θα βελτιώσει την αναπαραγωγικότητα και την αποδοτικότητα του μοντέλου.

**Βελτιστοποίηση Ροής HLS και Διαχείρισης Μνήμης** Παρά την επιτυχή εφαρμογή *pipelining* και *reuse factors*, υπάρχουν περιθώρια περαιτέρω βελτιστοποίησης στον σχεδιασμό και τη διαχείριση δεδομένων. Προτάσεις για συνέχιση περιλαμβάνουν:

- Εξερεύνηση **loop unrolling**, **array partitioning** και **dataflow directives** για καλύτερη παραλληλοποίηση.
- Εφαρμογή τεχνικών **on-chip buffering** και **double buffering** για ελαχιστοποίηση καθυστερήσεων πρόσβασης στη μνήμη DDR.
- Βελτιστοποίηση της επικοινωνίας μέσω **AXI interconnect** και **burst transfers** για υψηλότερο bandwidth μεταξύ PS και PL.

Η ενσωμάτωση των παραπάνω θα μπορούσε να μειώσει σημαντικά τη λανθάνουσα καθυστέρηση και να αυξήσει το *throughput* χωρίς επιπλέον κατανάλωση.

### Ενεργειακή Παρακολούθηση και Δυναμική Προσαρμογή

Οι μετρήσεις ισχύος στην παρούσα εργασία βασίστηκαν στον ενσωματωμένο αισθητήρα **INA260**. Μελλοντικά, θα μπορούσε να αναπτυχθεί ένας μηχανισμός **δυναμικού ελέγχου συχνότητας και τάσης**, όπου το σύστημα θα προσαρμόζει τη συχνότητα λειτουργίας του FPGA σε πραγματικό χρόνο ανάλογα με το φορτίο (*energy-aware inference*). Η χρήση εξωτερικών *power profilers* (όπως το **Xilinx Power Advantage Tool**) ή η συλλογή δεδομένων από πολλαπλά rails θα επέτρεπαν ακριβέστερη ανάλυση της ενεργειακής συμπεριφοράς του επιταχυντή.

### Υποστήριξη Πολλαπλών Δικτύων και Αυτοματισμός Ροής

Η προτεινόμενη ροή επικεντρώνεται σε ένα μόνο CNN. Μελλοντικά, θα μπορούσε να υποστηρίξει **πολλαπλούς επιταχυντές** (π.χ. CNN + MLP ή RNN) μέσω *partial reconfiguration*, επιτρέποντας δυναμική εναλλαγή μεταξύ διαφορετικών μοντέλων στο ίδιο FPGA. Επιπλέον, η αυτοματοποίηση της ροής μέσω Makefile, CMake ή Python

pipelines θα βελτίωνε τη *reproducibility* και θα μείωνε τον χρόνο ανάπτυξης. Η δημιουργία μιας **ενιαίας ροής HLS–Synthesis–Implementation** θα επέτρεπε γρήγορη επαναληπτική σχεδίαση και αξιολόγηση.

### Σχεδίαση Συνδυασμού Λογισμικού–Υλικού

Η μελλοντική έρευνα μπορεί να στραφεί σε πλήρη *hardware/software co-design*, αξιοποιώντας εργαλεία όπως το **FINN**, το **Brevitas** ή το **Vitis AI Optimizer**, ώστε το δίκτυο να μεταφράζεται αυτόματα από Python σε υλοποιήσιμη RTL μορφή. Ένα τέτοιο σύστημα θα επέτρεπε συνεχή βελτιστοποίηση μεταξύ επιπέδου εκπαίδευσης και υλοποίησης, μειώνοντας τα σφάλματα μεταφοράς και βελτιώνοντας τη συνέπεια μεταξύ προσομοίωσης και φυσικού hardware.

### Ετερογενής Επιτάχυνση

Η πλατφόρμα **Kria KR260** συνδυάζει επεξεργαστές ARM και προγραμματιζόμενη λογική. Μελλοντική έρευνα μπορεί να εστιάσει στη **συνδυασμένη χρήση CPU–FPGA**, όπου η CPU εκτελεί ελαφριά προεπεξεργασία ή μετα-ανάλυση, ενώ το FPGA επιταχύνει τις βασικές πράξεις του inference. Η εφαρμογή τεχνικών όπως *task-level pipelining* και *heterogeneous scheduling* μπορεί να βελτιώσει δραματικά τη συνολική ενεργειακή αποδοτικότητα και τον χρόνο απόκρισης του συστήματος.

### Εφαρμογές σε Edge AI και Ρομποτική Όραση

Η προτεινόμενη ροή μπορεί να επεκταθεί σε **πραγματικές εφαρμογές Edge AI**, όπως αναγνώριση αντικειμένων, παρακολούθηση συμβάντων ή οπτική καθοδήγηση ρομπότ. Η ενσωμάτωση σε περιβάλλοντα **ROS2** ή **Kria Robotics Stack** θα επιτρέψει τη χρήση του επιταχυντή σε εφαρμογές *real-time computer vision*, όπου η χαμηλή καθυστέρηση και η υψηλή ενεργειακή αποδοτικότητα είναι κρίσιμες.

### Αξιολόγηση σε Νεότερες Πλατφόρμες FPGA SoC

Η ίδια ροή μπορεί να μεταφερθεί σε νεότερες πλατφόρμες όπως οι **Versal AI Edge** ή οι **Zynq UltraScale+ MPSoC**, που ενσωματώνουν *AI Engines* και βελτιωμένους μηχανισμούς μνήμης. Μια συγκριτική μελέτη μεταξύ Kria KR260 και Versal θα προσέφερε ποσοτική αποτίμηση των κερδών σε κατανάλωση, latency και αξιοποίηση πόρων.

### Συνολική Κατεύθυνση

Συνολικά, η μελλοντική έρευνα θα μπορούσε να επικεντρωθεί στην αυτοματοποίηση της ροής, την υλοποίηση πιο σύνθετων και προσαρμοστικών δικτύων, καθώς και στην ενσωμάτωση σε ετερογενή συστήματα *edge AI*. Παράλληλα, η σύγκριση με εξειδικευμένους

επιταχυντές (TPUs, NPU, AI SoCs) θα συμβάλει στην πληρέστερη αξιολόγηση των πλεονεκτημάτων και των περιορισμών των FPGAs ως πλατφόρμες επιτάχυνσης τεχνητής νοημοσύνης στο πεδίο εφαρμογών πραγματικού χρόνου.

# Παράρτημα Α'

## Ακρωνύμια

**AUC** Area Under the Curve — Εμβαδό Κάτω από την Καμπύλη

**AI** Artificial Intelligence — Τεχνητή Νοημοσύνη

**API** Application Programming Interface — Διεπαφή Προγραμματισμού Εφαρμογών

**AXI** Advanced eXtensible Interface — Προηγμένο Επεκτάσιμο Πρωτόκολλο Διασύνδεσης

**AXI4** Advanced eXtensible Interface 4 — Πρωτόκολλο Διασύνδεσης AMBA 4

**BN** Batch Normalization — Κανονικοποίηση Κατά Παρτίδες

**BRAM** Block Random Access Memory — Ενσωματωμένη Μνήμη Μπλοκ

**CPU** Central Processing Unit — Κεντρική Μονάδα Επεξεργασίας

**CNN** Convolutional Neural Network — Συνελικτικό Νευρωνικό Δίκτυο

**DDR** Double Data Rate — Μνήμη Διπλής Ταχύτητας

**DL** Deep Learning — Βαθιά Μάθηση

**DMA** Direct Memory Access — Άμεση Πρόσβαση Μνήμης

**DRAM** Dynamic Random Access Memory — Δυναμική Μνήμη Τυχαίας Προσπέλασης

**DSP** Digital Signal Processor — Επεξεργαστής Ψηφιακών Σημάτων

**E2E** End-to-End — Από Άκρη σε Άκρη (Ολική Καθυστέρηση)

**FN** False Negative — Ψευδώς Αρνητικό

**FP** False Positive — Ψευδώς Θετικό

---

|              |                                                                                                       |
|--------------|-------------------------------------------------------------------------------------------------------|
| <b>FPR</b>   | False Positive Rate — Ποσοστό Ψευδώς Θετικών                                                          |
| <b>FPGA</b>  | Field-Programmable Gate Array — Πίνακας Προγραμματιζόμενης Λογικής Πεδίου                             |
| <b>GPIO</b>  | General Purpose Input Output — Είσοδος/Έξοδος Γενικής Χρήσης                                          |
| <b>GPU</b>   | Graphics Processing Unit — Μονάδα Επεξεργασίας Γραφικών                                               |
| <b>HLS</b>   | High-Level Synthesis — Σύνθεση Υψηλού Επιπέδου                                                        |
| <b>HP</b>    | High Performance Port — Θύρα Υψηλής Επίδοσης                                                          |
| <b>IP</b>    | Intellectual Property — Προκατασκευασμένο Μπλοκ Λογικής                                               |
| <b>LUT</b>   | Look-Up Table — Πίνακας Αναζήτησης                                                                    |
| <b>MAC</b>   | Multiply-Accumulate Operation — Πράξη Πολλαπλασιασμού-Συσσώρευσης                                     |
| <b>ML</b>    | Machine Learning — Μηχανική Μάθηση                                                                    |
| <b>MLP</b>   | Multi-Layer Perceptron — Πολυστρωματικό Αισθητήριο                                                    |
| <b>MNIST</b> | Modified National Institute of Standards and Technology Dataset — Σύνολο Δεδομένων Χειρόγραφων Ψηφίων |
| <b>NPU</b>   | Neural Processing Unit — Μονάδα Νευρωνικής Επεξεργασίας                                               |
| <b>ONNX</b>  | Open Neural Network Exchange — Ανοιχτή Μορφή Ανταλλαγής Νευρωνικών Δικτύων                            |
| <b>PL</b>    | Programmable Logic — Προγραμματιζόμενη Λογική                                                         |
| <b>PS</b>    | Processing System — Υποσύστημα Επεξεργασίας                                                           |
| <b>PYNQ</b>  | Python Productivity for Zynq — Πλαίσιο Ανάπτυξης Python για Zynq                                      |
| <b>QAT</b>   | Quantization Aware Training — Εκπαίδευση με Επίγνωση Κβάντισης                                        |
| <b>ReLU</b>  | Rectified Linear Unit — Διορθωμένη Γραμμική Μονάδα                                                    |
| <b>RF</b>    | Reuse Factor — Παράγοντας Επαναχρησιμοποίησης                                                         |
| <b>ROC</b>   | Receiver Operating Characteristic — Καμπύλη Απόκρισης Δέκτη                                           |
| <b>RTL</b>   | Register Transfer Level — Επίπεδο Μεταφοράς Καταχωρητών                                               |
| <b>SDK</b>   | Software Development Kit — Εργαλειοθήκη Ανάπτυξης Λογισμικού                                          |

**SoC** System on Chip — Σύστημα σε Τσιπ

**SoM** System on Module — Σύστημα σε Μονάδα

**TCL** Tool Command Language — Γλώσσα Αυτοματοποίησης Εργαλείων

**TN** True Negative — Αληθώς Αρνητικό

**TP** True Positive — Αληθώς Θετικό

**TPR** True Positive Rate — Ποσοστό Αληθώς Θετικών

**TPU** Tensor Processing Unit — Μονάδα Επεξεργασίας Τανυστήρα

**VM** Virtual Machine — Εικονική Μηχανή

**WNS** Worst Negative Slack — Χειρότερο Αρνητικό Περιθώριο Χρονισμού

**WSL** Windows Subsystem for Linux — Υποσύστημα των Windows για Linux

# Βιβλιογραφία

- [1] T. Aarrestad et al., “Fast convolutional neural networks on FPGAs with hls4ml,” *Machine Learning: Science and Technology*, vol. 2, no. 4, p. 045 015, Jul. 2021. DOI: 10.1088/2632-2153/ac0ea1 [Online]. Available: <https://dx.doi.org/10.1088/2632-2153/ac0ea1>
- [2] M. Abadi, P. Barham, J. Chen, and et al., “TensorFlow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016. [Online]. Available: <https://arxiv.org/abs/1605.08695>
- [3] A. H. Ali and G. A. Ali, “Convolutional neural networks using FPGA-based pipelining,” *Iraqi Journal for Computer Science and Mathematics*, vol. 4, no. 2, pp. 215–223, 2023. DOI: 10.52866/ijcsm.2023.02.02.019
- [4] AMD Xilinx, *Kria k26 SoM data sheet (ds987)*, Rev. 1.5, Jan 30, 2024, 2024. [Online]. Available: <https://docs.amd.com/r/en-US/ds987-k26-som/Programmable-Logic>
- [5] AMD Xilinx, *Kria kr260 robotics starter kit – product details (ds988)*, July 2, 2024, 2024. [Online]. Available: <https://docs.amd.com/r/en-US/ds988-kr260-starter-kit/Product-Details>
- [6] AMD Xilinx, *Vitis high-level synthesis user guide (ug1399)*, v2025.1, May 29, 2025, 2025. [Online]. Available: <https://docs.amd.com/r/en-US/ug1399-vitis-hls>
- [7] AMD Xilinx, *Vitis HLS - high level synthesis*, Accessed: 2025-09-29, 2025. [Online]. Available: <https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html>
- [8] Arm Ltd., *AMBA AXI4-Stream protocol specification*, ARM IHI 0051B, 2021. [Online]. Available: <https://documentation-service.arm.com/static/64819f1516f0f201aa6b963c>

- [9] A. Boutros et al., “Field-programmable gate array architecture for deep learning,” *arXiv preprint arXiv:2404.10076*, 2024. [Online]. Available: <https://arxiv.org/abs/2404.10076>
- [10] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (ELUs),” *arXiv:1511.07289*, 2015. [Online]. Available: <https://arxiv.org/abs/1511.07289>
- [11] N. Coelho and et al., “QKeras: A quantization extension to keras,” *arXiv:2003.03638*, 2020. [Online]. Available: <https://arxiv.org/abs/2003.03638>
- [12] F. Collaboration, *Hls4ml documentation*, <https://fastmachinelearning.org/hls4ml/>, Accessed: 2025-10-06, 2024.
- [13] F. Fahim, J. Duarte, et al., “Hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices,” in *TinyML Research Symposium*, 2021. arXiv: 2103.05579 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2103.05579>
- [14] Fast Machine Learning Lab, *Hls4ml documentation*, <https://fastmachinelearning.org/hls4ml/>, Accessed: 2025-09-30, 2023.
- [15] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” *arXiv preprint arXiv:2103.13630*, 2021. [Online]. Available: <https://arxiv.org/abs/2103.13630>
- [16] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011, pp. 315–323. [Online]. Available: <https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf>
- [17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <https://www.deeplearningbook.org/>
- [18] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 1319–1327. [Online]. Available: <https://proceedings.mlr.press/v28/goodfellow13.pdf>
- [19] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in *Proc. ICLR*, 2016. [Online]. Available: <https://arxiv.org/abs/1510.00149>

- [20] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” *arXiv:1502.01852*, 2015. [Online]. Available: <https://arxiv.org/abs/1502.01852>
- [21] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90
- [22] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. DOI: 10.1113/jphysiol.1952.sp004764 [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/>
- [23] A. G. Howard et al., “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” in *arXiv preprint arXiv:1704.04861*, 2017. [Online]. Available: <https://arxiv.org/abs/1704.04861>
- [24] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448–456. [Online]. Available: <https://proceedings.mlr.press/v37/ioffe15.html>
- [25] B. Jacob, S. Kligys, B. Chen, and et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” *arXiv:1712.05877*, 2018. [Online]. Available: <https://arxiv.org/abs/1712.05877>
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *arXiv:1706.02677*, 2017, Originally published at NIPS 2012. [Online]. Available: <https://arxiv.org/abs/1706.02677>
- [27] I. Kuon, R. Tessier, and J. Rose, *FPGA architecture: Survey and challenges*, Foundations and Trends in Electronic Design Automation, 2008. [Online]. Available: <https://www.eecg.toronto.edu/~jayar/pubs/kuon/foundtrend08.pdf>
- [28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791 [Online]. Available: <https://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>
- [29] Y. LeCun, C. Cortes, and C. J. Burges, *The MNIST database of handwritten digits*, <http://yann.lecun.com/exdb/mnist/>, Accessed: 2025-08-17, 1998.
- [30] M. Lin, Q. Chen, and S. Yan, “Network in network,” *arXiv:1312.4400*, 2013. [Online]. Available: <https://arxiv.org/abs/1312.4400>

- [31] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015. [Online]. Available: <https://arxiv.org/abs/1511.08458>
- [32] J. Park and W. Sung, “FPGA based implementation of deep neural networks using on-chip memory only,” *arXiv preprint arXiv:1602.01616*, 2016. [Online]. Available: <https://arxiv.org/abs/1602.01616>
- [33] A. Paszke, S. Gross, F. Massa, and et al., “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. [Online]. Available: <https://arxiv.org/abs/1912.01703>
- [34] Pedregosa, Fabian and Varoquaux, Gaël and Gramfort, Alexandre and Michel, Vincent and Thirion, Bertrand and Grisel, Olivier and Blondel, Mathieu and Prettenhofer, Peter and Weiss, Ron and Dubourg, Vincent and Vanderplas, Jake and Passos, Alexandre and Cournapeau, David and Brucher, Matthieu and Perrot, Matthieu and Duchesnay, Édouard, *Scikit-learn: Machine learning in Python*, [https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html), Accessed: 2025-09-30, 2011.
- [35] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015. DOI: 10.1016/j.neunet.2014.09.003 [Online]. Available: <https://arxiv.org/abs/1404.7828>
- [36] H. Sharma et al., “From high-level deep neural models to FPGAs,” in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783720 [Online]. Available: [https://act-lab.org/doc/paper/2016-micro-dnn\\_weaver.pdf](https://act-lab.org/doc/paper/2016-micro-dnn_weaver.pdf)
- [37] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2015. [Online]. Available: <https://arxiv.org/abs/1409.1556>
- [38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: <http://jmlr.org/papers/v15/srivastava14a.html>
- [39] Y. Su, X. Li, Y. Zhang, L. Gao, F. Qiao, and Z. Wang, “Binary neural networks in FPGAs: Architectures, tool flows, and applications,” *Sensors*, vol. 23, no. 22, p. 9254, 2023. DOI: 10.3390/s23229254 [Online]. Available: <https://www.mdpi.com/1424-8220/23/22/9254>

- [40] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *Proceedings of the International Conference on Machine Learning (ICML)*, PMLR, 2019, pp. 6105–6114. [Online]. Available: <https://arxiv.org/abs/1905.11946>
- [41] Texas Instruments, *Ina260: Precision digital current and power monitor with i<sup>2</sup>c interface*, <https://www.ti.com/lit/ds/symlink/ina260.pdf>, Accessed: 2025-09-30, 2017.
- [42] Y. Umuroglu et al., “Finn: A framework for fast, scalable binarized neural network inference,” in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 65–74. DOI: 10.1145/3020078.3021744 [Online]. Available: <https://arxiv.org/abs/1612.07119>
- [43] S. I. Venieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions,” *ACM Computing Surveys*, vol. 51, no. 3, 56:1–56:39, Jun. 2018. DOI: 10.1145/3186332 arXiv: 1803.05900 [cs.AR]. [Online]. Available: [https://cas.ee.ic.ac.uk/people/sv1310/papers/%5B2018%5D\\_acm\\_csur\\_toolflows\\_for\\_mapping\\_cnns\\_on\\_fpgas\\_a\\_survey\\_and\\_future\\_directions.pdf](https://cas.ee.ic.ac.uk/people/sv1310/papers/%5B2018%5D_acm_csur_toolflows_for_mapping_cnns_on_fpgas_a_survey_and_future_directions.pdf)
- [44] M. P. Véstias, “A survey of convolutional neural networks on edge with reconfigurable computing,” *Algorithms*, vol. 12, no. 8, p. 154, 2019. DOI: 10.3390/a12080154 [Online]. Available: <https://www.mdpi.com/1999-4893/12/8/154>
- [45] *Vivado design suite user guide: Introduction to FPGA design with Vivado high-level synthesis (ug998)*, Accessed: 2025-09-29, AMD-Xilinx, 2023. [Online]. Available: <https://docs.amd.com/v/u/en-US/ug998-vivado-intro-fpga-design-hls>
- [46] Xilinx, *Kria-PYNQ: Official PYNQ framework for kria SoMs (kr260/kv260)*, <https://github.com/Xilinx/Kria-PYNQ>, Accessed: 2025-10-06, 2024.
- [47] A. Xilinx, *Kria kr260 robotics starter kit — product and software overview*, <https://www.amd.com/en/products/system-on-modules/kria/kr260.html>, Accessed: 2025-10-06, 2023.