Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αξιολόγηση τηλεπικοινωνιακού συστήματος με συνδυασμό μηχανικής μάθησης και κλασικών τεχνικών επεξεργασίας σήματος

Διπλωματική Εργασία

της

Σιόκουρου Αιμιλία

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αξιολόγηση τηλεπικοινωνιακού συστήματος με συνδυασμό μηχανικής μάθησης και κλασικών τεχνικών επεξεργασίας σήματος

Διπλωματική Εργασία

της

Σιόκουρου Αιμιλία

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31$^{\eta}$ Μαρτίου 2023:

....	....	....
Δημήτριος Σούντρης	Διονύσης Ρείσης	Σωτήριος Ξύδης
Καθηγητής	Καθηγητής	Επίκουρος Καθηγητής
Ε.Μ.Π.	ΕΚΠΑ	Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

(Υπογραφή)

....

Σιόκουρου Αιμιλία
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών, Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια τα ασύρματα συστήματα 5G έχουν αρχίσει να εμφανίζονται για εμπορικούς σκοπούς. Αν και δεν έχουν εμφανιστεί ακόμη πλήρως συμβατά συστήματα, νέα χαρακτηριστικά ενσωματώνονται συνεχώς όχι μόνο στην τηλεπικοινωνιακή υποδομή αλλά και στον εξοπλισμό των χρηστών. Καθώς αυτή η ενσωμάτωση συνεχίζεται και οι ανάγκες αυτών των συστημάτων γίνονται ακόμη πιο απαιτητικές, είναι απαραίτητο να καλυφθούν με σταθερά, αξιόπιστα και έξυπνα συστήματα επικοινωνίας. Τα Field Programmable Gate Arrays (FPGAs) είναι μια εξαιρετική επιλογή για το σκοπό αυτό, καθώς παρέχουν καλό trade-off μεταξύ τιμής, ισχύος επεξεργασίας και απόδοσης.

Στην παρούσα διπλωματική, στοχεύουμε στη διόρθωση των I-Q imbalances σε Direct Conversion Receivers με έναν αλγόριθμο που υλοποιήθηκε στο πρόγραμμα Xilinx Vivado, χρησιμοποιώντας τη γλώσσα περιγραφής υλικού (VHDL). Η αξιολόγηση του αλγορίθμου πραγματοποιείται στο MATLAB συγκρίνοντας το σχετικό σφάλμα μεταξύ των αρχικών δεδομένων και των διορθωμένων αποτελεσμάτων τόσο στο MATLAB όσο και στο Vivado. Στη συνέχεια, χρησιμοποιήθηκε ένα RF dataset που περιλαμβάνει 24 τύπους ψηφιακών και αναλογικών διαμορφώσεων σε ποικίλα SNRs και εφαρμόστηκε σε αυτό I-Q imbalance. Ακολούθως, χρησιμοποιώντας αυτό το dataset ως test bench για τον αλγόριθμο διόρθωσης στο Vivado, επαναφέραμε τα imbalanced δεδομένα στην αρχική τους κατάσταση. Τέλος, με τη βοήθεια του περιβάλλοντος Vitis-AI πραγματοποιήθηκε αναγνώριση RF διαμορφώσεων χρησιμοποιώντας Deep Neural Networks για να ταξινομηθούν οι διάφορες διαμορφώσεις και να αξιολογηθεί η ακρίβεια της ταξινόμησης των αρχικών, των imbalanced και των διορθωμένων δεδομένων. Η απόδοση και η ακρίβεια, των κβαντισμένων και compiled μοντέλων, επαληθεύεται στην πλακέτα Zynq Ultrascale+ RFSoC ZCU111.

Λέξεις Κλειδιά: Ψηφιακές Επικοινωνίες, Μηχανική Μάθηση, Νευρωνικά Δίκτυα, Αναγνώριση Διαμόρφωσης, Ακρίβεια, Απόδοση, Πόροι

Abstract

The past couple of years 5G wireless systems have started appearing for commercial purposes. Even thought, fully compatible systems are yet to appear, new features are continuously integrated not only on the telecommunication infrastructure but also to user equipment. As this integration continues and the needs of these systems become even more demanding, it is vital to cater them with stable, reliable and intelligent communication systems. Thus, they require high-speed digital interfaces which are capable to tackle these issues. Field Programmable Gate Arrays (FPGAs) are an excellent choice for this purpose since they provide a great trade-off of price, processing power, efficiency and parallelism.

In this diploma thesis, we target the correction of I-Q imbalances in Direct Conversion Receivers with an algorithm implemented in the software suite Xilinx Vivado, using hardware description language (VHDL). The evaluation of the algorithm is performed in MATLAB by comparing the relative error between the original data and the corrected results in both MATLAB and Vivado. Next, we use an RF Dataset which includes 24 digital and analog modulation types at varying signal-to-noise ratios (SNRs) and apply I-Q imbalance to its data. Then utilizing this dataset as a test bench for the correction algorithm in Vivado we restore the imbalanced data to their original state. Finally, with the assistance of the Vitis-AI environment we perform RF-modulation recognition using Deep Neural Networks to classify the different modulations and evaluate the accuracy of the classification from the original, the imbalanced and the corrected data. The performance and accuracy, of the quantized and compiled models, is verified on the Zynq Ultrascale+ RFSoC ZCU111 board.

Keywords: FPGA, VHDL, Digital Communication, I-Q imbalance correction, Resources, QAM Modulation, Machine Learning, Neural-Networks, Vitis-AI, Modulation Recognition, Accuracy, Performance

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Δημήτριο Σούντρη που μου έδωσε την ευκαιρία να εκπονήσω την διπλωματική μου στο Microlab με ένα θέμα άκρως ενδιαφέρον για μένα.

Ακόμη, θέλω να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα Ιωάννη Στρατάκο για την άμεση και αποτελεσματική βοήθεια που είχα κάθε φορά που αντιμετώπιζα κάποιο πρόβλημα. Επίσης, η εις βάθος γνώσεις του στα FPGA με σύστισε στις ατελείωτες δυνατότητες τους.

Επιπλέον, να ευχαριστήσω τους φίλους μου, τα αδέρφια μου και τους γονείς μου που με στήριξαν σε κάθε δυσκολία που μπορεί να αντιμετώπιζα όλα αυτά τα χρόνια.

Σιόκουρου Αιμιλία, Μάρτιος 2023

Περιεχόμενα

Ευρετήριο Εικόνων

Κατάλογος Πινάκων

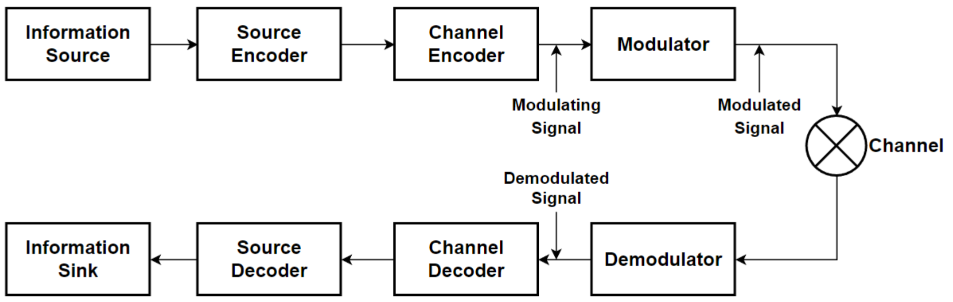
Εκτεταμένη Περίληψη

Εισαγωγή

Στην εποχή που ζούμε, είναι αδιαμφισβήτητο ότι, οι ψηφιακές επικοινωνίες υπερτερούν σε σχέση με τις αντίστοιχες αναλογικές. Αυτό συμβαίνει διότι τα πλεονεκτήματα των ψηφιακών επικοινωνιών είναι πολύ περισσότερα από αυτά των αναλογικών. Η τεράστια ζήτηση για μεταφορά δεδομένων, η μεγαλύτερη αξιοπιστία των ψηφιακών ηλεκτρονικών σε συνδυασμό με το χαμηλό τους κόστος για επεξεργασία σήματος, η ευκολία στον περιορισμό των επιπτώσεων του θορύβου και παρεμβολών όταν χρησιμοποιείται κωδικοποίηση των καναλιών, εξηγεί απόλυτα γιατί οι ψηφιακές επικοινωνίες είναι η ιδανική επιλογή για επικοινωνιακά συστήματα.

Ψηφιακό Σύστημα Επικοινωνιών

Στο σχήμα 0.0.1 παρουσιάζεται το μπλοκ διάγραμμα ενός ψηφιακού επικοινωνιακού συστήματος, συμπεριλαμβανομένου του πομπού και δέκτη.



Σχήμα 0.0.1: Μπλοκ διάγραμμα ψηφιακών συστημάτων επικοινωνιών

Πιο αναλυτικά, ο κωδικοποιητής πηγής είναι υπεύθυνος για την αναπαράσταση

της μεταδιδόμενης πληροφορίας σε ψηφιακή μορφή.
Ο κωδικοποιητής καναλιού προσθέτει ελεγχόμενο πλεονασμό στην πληροφορία έτσι ώστε να είναι εφικτό να αντιμετωπίσει σε επόμενα στάδια πιθανά σφάλματα λόγω ατελειών του καναλιού και θορύβου.
Ο διαμορφωτής, μεταφράζει τα διακριτά σύμβολα που είναι η έξοδος του κωδικοποιητή καναλιού σε αναλογική κυματομορφή έτσι ώστε να μπορεί να μεταδοθεί μέσω φυσικού καναλιού.

Η μορφή του καναλιού που χρησιμοποιείται σε ένα ψηφιακό σύστημα επικοινωνίας ποικίλλει ανάλογα ανάλογα με την κάθε εφαρμογή. Για παράδειγμα, στην ενσύρματη επικοινωνία χρειάζεται ένα γραμμικό χρονικά αμετάβλητο που σε πολλές περιπτώσεις περιλαμβάνει ανατροφοδότηση απο τον δέκτη στον πομπό σχετικά με την διαμόρφωση που εφαρμόστηκε στο προηγούμενο στάδιο. Αντιθέτως, στην ασύρματη κινητή επικοινωνία δεν υπάρχει ανάδραση προς τον πομπό. Επομένως, το κανάλι πρέπει να εκτιμηθεί ή να χρησιμοποιήσει μεθόδους που δεν απαιτούν ακριβή εκτίμηση καναλιού.

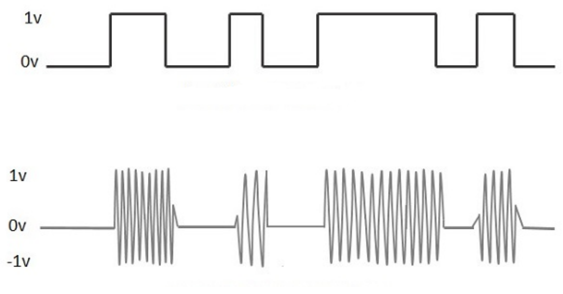
Ο αποδιαμορφωτής επεξεργάζεται την αναλογική κυματομορφή που έχει λάβει μέσω του καναλιού, η οποία έχει αλλοιωθεί λόγω θορύβου. Συγκεκριμένα, πρέπει να πάρει αποφάσεις σχετικά με τα μεταδιδόμενα σύμβολα όσο αφορά τις μετατοπίσεις φάσης, συχνότητας και χρόνου και την εξισορρόπηση ή αντιστάθμιση του καναλιού.
Αυτές οι αποφάσεις εισάγονται στον αποκωδικοποιητή καναλιού, ο οποίος χρησιμοποιεί τον πλεονασμό για να αναπαραστήσει την εκτίμηση της ακολουθίας συμβόλων που παράχθηκαν από τον κωδικοποιητή πηγής.
Ο αποκωδικοποιητής πηγής, μετασχηματίζει τα εκτιμώμενα σύμβολα σε μορφή που μπορεί να κατανοήσει ο δέκτης.

Τεχνικές Ψηφιακής Διαμόρφωσης

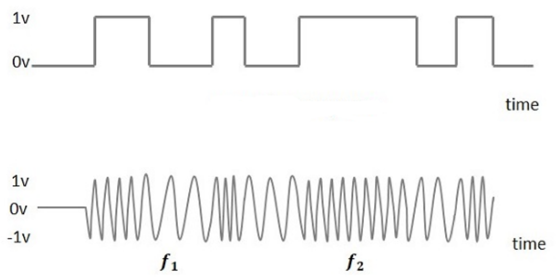
Τα είδη ψηφιακής διαμόρφωσης μπορούν να ταξινομηθούν σε τρεις μεγάλες κατηγορίες:

- Μεταλλαγή Μετατόπισης Πλάτους (Amplitude Shift Keying), όπου το πλάτος του φέροντος σήματος μετατοπίζεται μεταξύ δύο διαφορετικών επιπέδων πλάτους για να αναπαραστήσει τα ψηφιακά δεδομένα (0.0.2).

- Μεταλλαγή Μετατόπισης Συχνότητας (Frequency Shift Keying), όπου το κύμα εξόδου μετατοπίζεται μεταξύ διαφορετικών συχνοτήτων για να αναπαραστήσει τα ψηφιακά σύμβολα(Σχήμα 0.0.3).

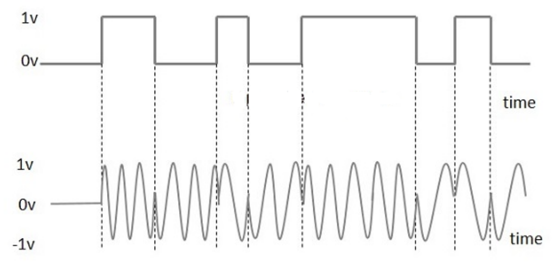


Σχήμα 0.0.2: ASK Διαμορφωμένο Κύμα Εξόδου



Σχήμα 0.0.3: FSK Διαμορφωμένο Κύμα Εξόδου

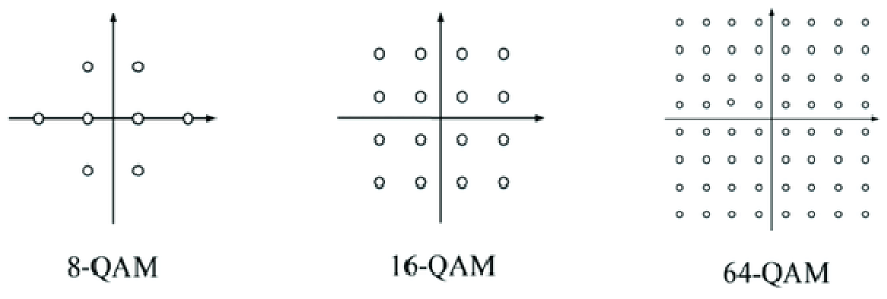
- Μεταλλαγή Μετατόπισης Φάσης (Phase Shift Keying), όπου η φάση ενός ημιτονοειδούς κύματος φορέα σταθερής συχνότητας αλλάζει κάθε φορά που υπάρχει αλλαγή στην κατάσταση του σήματος εισόδου (Σχήμα 0.0.4).



Σχήμα 0.0.4: PSK Διαμορφωμένο Κύμα Εξόδου

Ο συνδυασμός δύο ή περισσοτέρων από τις παραπάνω τεχνικές οδηγούν σε άλλες γνωστές τεχνικές διαμόρφωσης όπως:

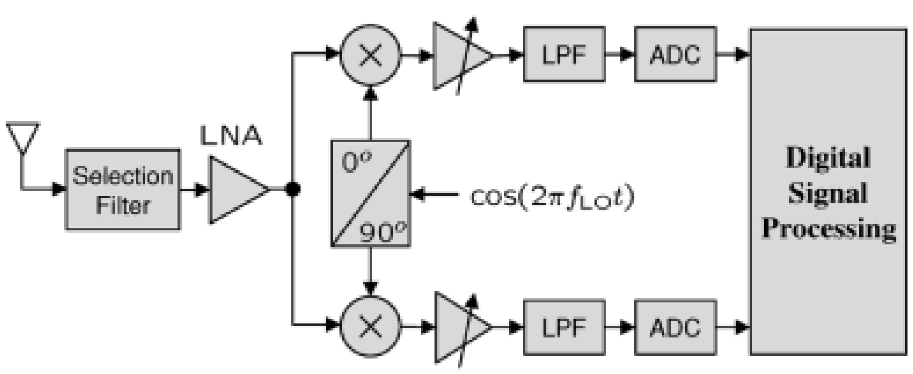
- Τετραγωνική Διαμόρφωση Πλάτους (Quadrature Modulation Technique, όπου δύο ροές σημάτων έχουν ίδια συχνότητα αλλά έχουν διαφορά φάσης 90 μοίρες (τετραγωνισμός) (Σχήμα 0.0.5).



Σχήμα 0.0.5: Διάφορες QAM Διαμορφώσεις

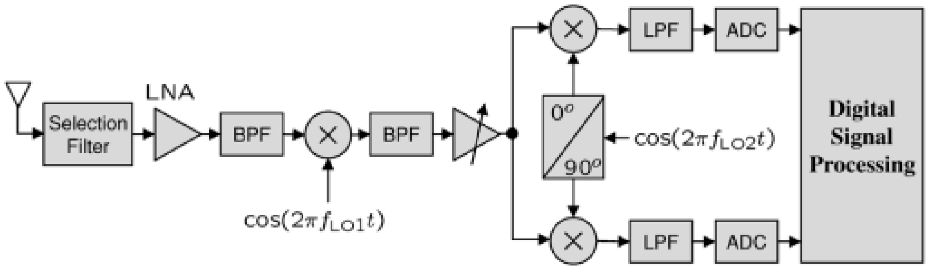
I-Q Imbalance

Imbalances μεταξύ των I-Q συμβόλων (I-Q Imbalances) μπορούν να βρεθούν σε δέκτες άμεσης μετατροπής (direct conversion receivers) όπου δεν υπάρχει ενδιάμεση συχνότητα f_{IF}, με αποτέλεσμα το λαμβανόμενο σήμα RF ή το ζωνοπερατό σήμα να μεταφράζεται απευθείας από την φέρουσα συχνότητα f_C στο σήμα ζώνης με ένα στάδιο ανάμειξης (Σχήμα 0.0.6) . Ο δέκτης άμεσης μετατροπής εκτελεί την λεγόμενη μετατροπη τετραγωνισμού προς τα κάτω (quadrature down-conversion) με την χρήση δύο τετραγωνικών ημιτονοειδών σημάτων. Σε αυτή την διαδικασία, μετατοπίζεται το σήμα από τον τοπικό ταλαντωτή (LO) κατά 90°για να παραχθεί ένα τετραγωνικό ημιτονοειδές στοιχείο και ένα ταιριαστό ζεύγος από mixers μετατρέπει το ίδιο σήμα εισόδου με τις δύο εκδόσεις του LO. Οι αναντιστοιχίες μεταξύ των δύο σημάτων του LO ή κατά μήκος των δύο διακλαδώσεων των down-conversion mixers και τυχόν ακόλουθων ενισχυτών και low pass φίλτρων, προκαλούν αλλοίωση στο τετραγωνικό baseband σήμα. Αν η διαφορά φάσης μεταξύ των τετραγωνικών κυματομορφών δεν είναι ακριβώς 90°ή αν τα πλάτη τους δεν είναι ίσα, τότε υπάρχει I-Q imbalance.



Σχήμα 0.0.6: Αρχιτεκτονική ενός Zero intermediate frequency (ZIF) receiver

Αυτό είναι το κύριο μειονέκτημα των δεκτών άμεσης μετατροπής σε σχέση με τους heterodyne receivers (Σχήμα 0.0.7), όπου η φέρουσα συχνότητα πολλαπλασιάζεται με τα τοπικά ταλαντευόμενα σήματα για να μετατραπούν σε ενδιάμεσες συχνότητες f_{IF}, οι οποίες είναι κατάλληλες για περαιτέρω ενίσχυση και επεξεργασία.

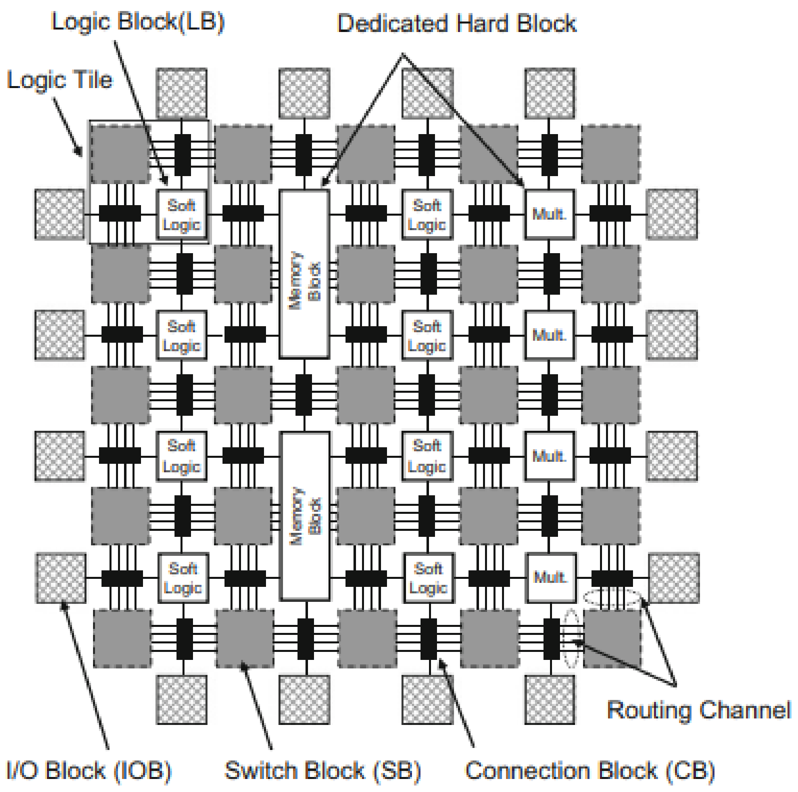


Σχήμα 0.0.7: Αρχιτεκτονική ενός Heterodyne receiver with intermediate frequency (IF)

Field-Programmable Gate Arrays (FPGAs)

Τα FPGAs είναι επαναπρογραμματιζόμενα τσιπ υλικού που χρησιμοποιούνται για ψηφιακή λογική. Είναι μια σειρά από λογικές πύλες που μπορούν να ρυθμιστούν για να δημιουργούν αυθαίρετα ψηφιακά κυκλώματα. Η προδιαγραφή αυτών των κυκλωμάτων γίνεται χρησιμοποιώντας είτε σχηματικά κυκλώματα είτε με γλώσσες περιγραφής υλικού, Verilog ή VHDL. Η αρχιτεκτονική ενός

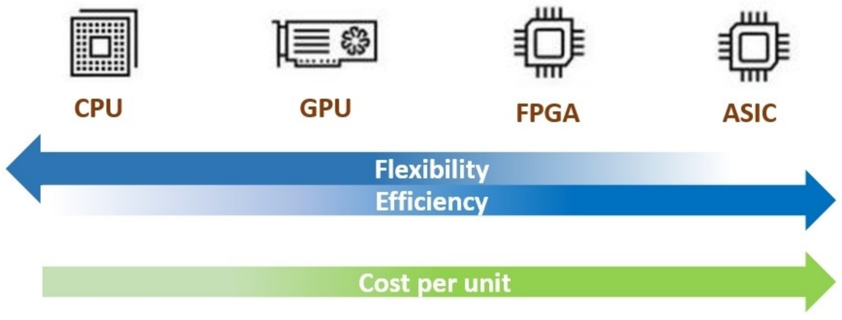
FPGA παρουσιάζεται στο Σχήμα 0.0.8.



Σχήμα 0.0.8: Αρχιτεκτονική ενός FPGA

Τα τελευταία χρόνια τα FPGAs έχουν γίνει πολύ δημοφιλή στην βιομηχανία και χρησιμοποιούνται και ως επιταχυντές(accelerators) για AI εφαρμογές και για επιτάχυνση τεχνητών νευρωνικών δικτύων. Οι αιτίες που οδήγησαν σε αυτή την δημοτικότητα των FPGAs έναντι άλλων μονάδων επεξεργασίας (π.χ CPUs, GPUs, ASICs είναι:
Τα FPGAs έχουν την δυνατότητα να επαναδιαμορφωθούν εκ νέου μετά την εγκατάσταση, ενώ τα ASICs δεν μπορούν να το κάνουν αυτό. Επίσης, μπορούν να επεξεργαστούν παράλληλα μεγάλο όγκο δεδομένων με αποτέλεσμα να έχουν χαμηλό latency. Οι CPUs και GPUs λειτουργούν σειριακά, επομένως δεν μπορούν να επεξεργαστούν πληροφοριές ταυτόχρονα. Ακόμη, λόγω του χαμηλού latency τους μπορούν να χρησιμοποιηθούν σε εφαρμογές με απαιτητικούς χρόνους απόκρισης, π.χ ιατρικές συσκευές. Παρόλο, που τα ASICs

μπορούν να έχουν ακόμα χαμηλότερο latency, δεν μπορούν να χρησιμοποιηθούν για πάνω από μία εφαρμογή. Τέλος, τα FPGAs έχουν καλύτερη απόδοση σε σχέση με τις CPUs και GPUs και έχουν πολύ χαμηλότερο κόστος κατασκευής σε σχέση με τα ASICs που μπορούν να ξεπεράσουν την απόδοση των FPGAs. (Σχήμα 0.0.9)



Σχήμα 0.0.9: CPU vs GPU vs FPGA vs ASIC

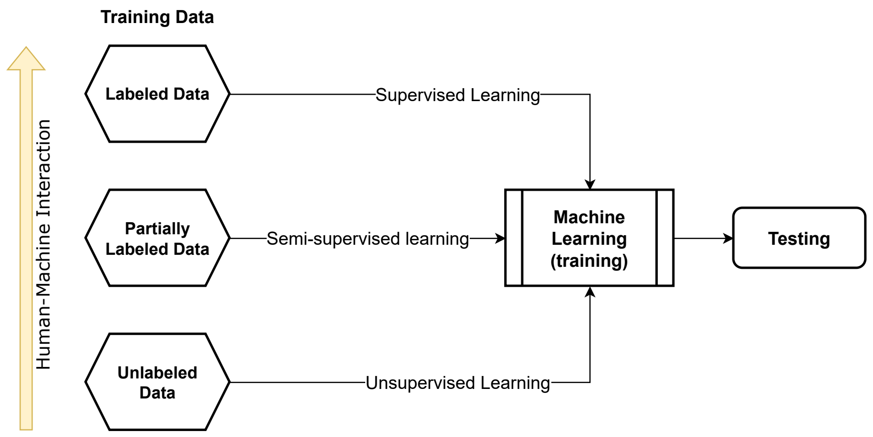
Μηχανική Μάθηση

Η Μηχανική Μάθηση χρησιμοποιεί υπολογιστικές μεθόδους για την βελτίωση της απόδοσης ενός συστήματος μέσω της ¨εμπειρίας¨. Μιμείται την ανθρώπινη νοημοσύνη μαθαίνοντας από το περιβάλλον του. Ο κύριος στόχος της μηχανικής μάθησης είναι η δημιουργία αλγόριθμων μάθησης για την κατασκευή μοντέλων που βασίζονται σε δεδομένα με εμπειρίες που είναι και αυτές σε μορφή δεδομένων.
Οι αλγόριθμοι μηχανικής μάθησης επιτυγχάνουν μια επιθυμητή εργασία χωρίς να χρειάζεται να προγραμματιστούν κυριολεκτικά για να παράγουν το επιθυμητό αποτέλεσμα. Είναι ουσιαστικά ”soft coded” αφού αυτοί οι αλγόριθμοι μπορούν αυτόματα να προσαρμόσουν ή να αλλάξουν την αρχιτεκτονική τους μέσω επανάληψης, προκειμένου να βελτιώσουν την ικανότητά τους να επιτύχουν το συγκεκριμένο αποτέλεσμα. Στο σχήμα 0.0.10 φαίνονται οι 3 κατηγορίες αλγόριθμων μηχανικής μάθησης:

- Επιτηρούμενη μάθηση: υπάρχει προκαθορισμένη εξόδος για κάθε δεδομένο εισόδου
- Μη επιτηρούμενη μάθηση: δεν έχουν προκαθορισμένο στόχο όταν εκπαιδεύονται. Χρησιμοποιούν αλγόριθμους ομαδοποίησης για να συνειδη-

τοποιήσουν και να προσδιορίσουν ομοιότητες/ομαδοποιήσεις μεταξύ των δεδομένων που δεν έχουν ετικέτα και δίνουν σε κάθε ένα μία ετικέτα

- Ημι-επιτηρούμενη μάθηση: ένα ποσοστό δεδομένων έχει ετικέτα ενώ τα υπόλοιπα όχι. Τα ¨ονοματισμένα' δεδομένα βοηθούν στην εκπαίδευση του μη ονοματισμένου τμήματος



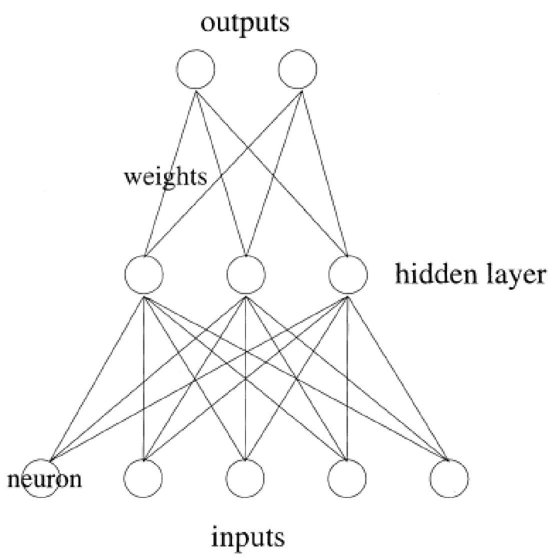
Σχήμα 0.0.10: Κατηγορίες Αλγόριθμων Μηχανικής Μάθησης με βάση την φύση των δεδομένων εκπαίδευσης

Τεχνητά Νευρωνικά Δίκτυα

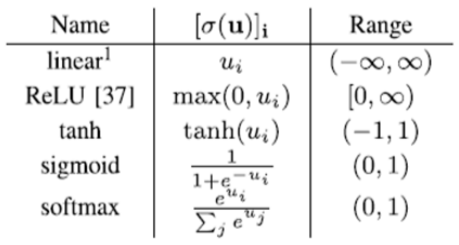
Ένα τεχνητό νευρωνικό δίκτυο περιέχει ένα επίπεδο εισόδου από νευρώνες, μερικά κρυμμένα στρώματα από νευρώνες και ένα τελικό στρώμα για τους νευρώνες εξόδου. Η αρχιτεκτονική ενός τεχνητού νευρωνικού δικτύου φαίνεται στο σχήμα 0.0.11. Κάθε κόμβος σχετίζεται με τον νευρώνα που συνδέεται με μια αριθμητική τιμή που ονομάζεται βάρος. Η έξοδος κάθε νευρώνα στα κρυμμένα στρώματα έχει την ακόλουθη γραμμική συνάρτηση:

$$y_i = \sigma(\sum_{j=1}^{N} w_{ij}x_j + b_i)$$

όπου σ είναι η συνάρτηση ενεργοποίησης, N ο αριθμός των νευρώνων εισόδου, w_{ij} τα βάροι, x_j οι εισόδοι και b_i η πόλωση του κρυμμένου νευρώνα. Οι συναρτησεις ενεργοποίησης μπορούν να είναι οποιεσδήποτε από τις παρακάτω (Σχήμα 0.0.12):



Σχήμα 0.0.11: Αρχιτεκτονική ενός τεχνητού νευρωνικού δικτύου



Name	$[\sigma(\mathbf{u})]_\mathbf{i}$	Range
linear[1]	u_i	$(-\infty, \infty)$
ReLU [37]	$\max(0, u_i)$	$[0, \infty)$
tanh	$\tanh(u_i)$	$(-1, 1)$
sigmoid	$\frac{1}{1+e^{-u_i}}$	$(0, 1)$
softmax	$\frac{e^{u_i}}{\sum_j e^{u_j}}$	$(0, 1)$

Σχήμα 0.0.12: Λίστα με συναρτήσεις ενεργοποίησης

Διόρθωση I-Q Imbalance

Όπως αναφεράμε πιο πριν, imbalances μεταξύ των I-Q συμβόλων συμβαίνουν σε δέκτες άμεσης μετατροπής όταν η διαφορά φάσης των δύο δεν είναι 90°ή δεν έχουν ίδιο πλάτος. Θεωρούμε τα παρακάτω τις εξόδους όταν μετατρέπεται μία μονοτονική κυματομορφή RF σε ζώνη βάσης:

$$I(t) = \cos(\omega t) \tag{0.0.1}$$

$$Q(t) = \sin(\omega t) \tag{0.0.2}$$

όπου ω είναι η συχνότητα της ζώνης βάσης της μονοτονικής κυματομορφής. Θεωρούμε το πλάτος ίσο με μονάδα και την φάση ίση με 0 καθώς δεν επηρεάζουν

τον διορθωτικό αλγόριθμο.
Ωστόσο, σε πραγματικούς δέκτες άμεσης μετατροπής τα I-Q είναι::

$$I'(t) = \alpha \cos(\omega t) + \beta_i \tag{0.0.3}$$

$$Q'(t) = \sin(\omega t + \psi) + \beta_q \tag{0.0.4}$$

όπου ψ είναι το σφάλμα φάσης, α το σφάλμα πλάτους και β_i και β_q είναι οι DC πολώσεις που συμβαίνουν σε κάθε σύμβολο.

Αλγόριθμος Διόρθωσης

Αρχικά, τα β_i και β_q είναι ο μέσος όρος των $I'(t), Q'(t)$, αντίστοιχα, σε ένα ακέραιο αριθμό περιόδων. Επομένως αφαιρείται ο μέσος όρος του κάθε μονοπατιού και οι εξόδοι είναι:

$$I"(t) = \alpha \cos(\omega t) \tag{0.0.5}$$

$$Q"(t) = \sin(\omega t + \psi) \tag{0.0.6}$$

Οι παράμετροι α, $\sin(\psi)$, $\cos(\psi)$ υπολογίζονται με την βοήθεια της εξίσωσης $< x(t) >= \frac{1}{NT} \int_{t-NT}^{t} x(u)\, du$, ως εξής:

$$< I"(t)I"(t) >= \alpha^2 < cos^2(\omega t) >= \alpha^2 < \frac{1}{2} + \frac{1}{2} cos(2\omega t) >= \frac{1}{2}\alpha^2 \tag{0.0.7}$$

$$< I"(t)Q"(t) >= \frac{1}{2}\alpha sin(\psi) \tag{0.0.8}$$

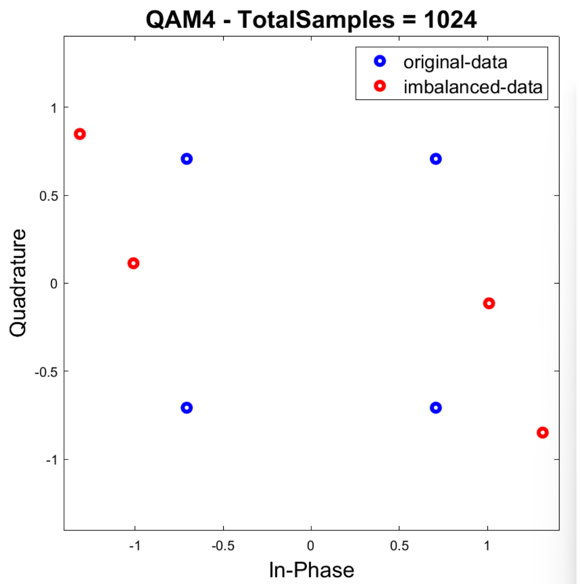
$$\cos(\psi) = \sqrt{1 - \sin^2(\psi} \tag{0.0.9}$$

Οι διορθωμένες έξοδοι με την χρήση της τριγωνομετρικής ταυτότητας $(sin(\omega t + \psi) = sin(\omega t)cos(\psi) + cos(\omega t)sin(\psi))$ είναι:

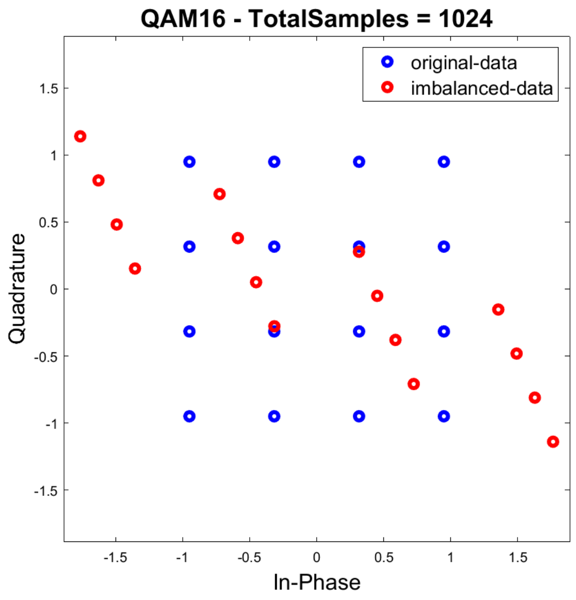
$$\begin{bmatrix} I(t) \\ Q(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha} & 0 \\ \frac{-\sin(\psi)}{\alpha \cos(\psi)} & \frac{1}{\cos(\psi)} \end{bmatrix} \begin{bmatrix} I"(t) \\ Q"(t) \end{bmatrix} \tag{0.0.10}$$

Εκτίμηση αλγόριθμου διόρθωσης

Με την βοήθεια του MATLAB παράγουμε σύμβολα με QAM διαμορφώσεις και πιο συγκεκριμένα 4-QAM και 16-QAM. Έπειτα, εφαρμόζουμε I-Q Imbalance στην κάθε διαμόρφωση.
Στα σχήματα 0.0.13α΄ και 0.0.13β΄ παρουσιάζονται οι έξοδοι I-Q χωρίς και με 10dB και 45°imbalance.



(α′) 4-QAM αρχικά και με imbalance δεδομένα



(β′) 16-QAM αρχικά και με imbalance δεδομένα

Σχήμα 0.0.13: Τάξης Διαμόρφωσης 4,16-QAM

Κατασκευάζουμε τον αλγόριθμο διόρθωσης με 4 διαφορετικά παράθυρα επεξεργασίας ίσα με 64, 128, 256 και 1024 και 3 διαφορετικά μεγέθη δειγμάτων για κάθε παράθυρο ίσα με 1024, 131072 και 1048576.
Τα διορθωμένα I-Q δεδομένα συγκρίνονται με τα αρχικά δεδομένα που παράχθηκαν στο MATLAB με την βοήθεια της εξίσωσης του σχετικού σφάλματος που φαίνεται παρακάτω:

$$RelativeErrorPercentage = \frac{|measured - real|}{real} \times 100 \qquad (0.0.11)$$

Τα αποτελέσματα του ποσοστού του σχετικού σφάλματος για τα I-Q σύμβολα για **4-QAM** φαίνονται στους παρακάτω πίνακες:

4-QAM	Μέγεθος Παραθύρου			
Πλήθος δειγμάτων	64	128	256	1024
1024	14.1903	12.7841	12.8236	12.7749
131072	14.7345	13.4893	13.0347	12.9650
1048576	14.7263	13.4075	13.0083	12.9775

Πίνακας 1: Ποσοστό Σχετικού Σφάλματος μεταξύ I-Αρχικών δεδομένων και I-Matlab αποτελεσμάτων 4-QAM

4-QAM	Μέγεθος Παραθύρου			
Πλήθος δειγμάτων	64	128	256	1024
1024	37.0323	34.8078	34.5293	34.4696
131072	38.7620	37.6060	37.1421	36.8795
1048576	38.8801	37.6928	37.2487	36.9780

Πίνακας 2: Ποσοστό Σχετικού Σφάλματος μεταξύ Q-Αρχικών δεδομένων και Q-Matlab αποτελεσμάτων 4-QAM

Τα αποτελέσματα του ποσοστού του σχετικού σφάλματος για τα I-Q σύμβολα για **16-QAM** φαίνονται στους παρακάτω πίνακες:

16-QAM	Μέγεθος Παραθύρου			
Πλήθος δειγμάτων	64	128	256	1024
1024	21.0646	18.5447	17.8823	16.9282
131072	22.4404	19.7023	18.3292	17.3180
1048576	22.3744	19.6561	18.2860	17.3941

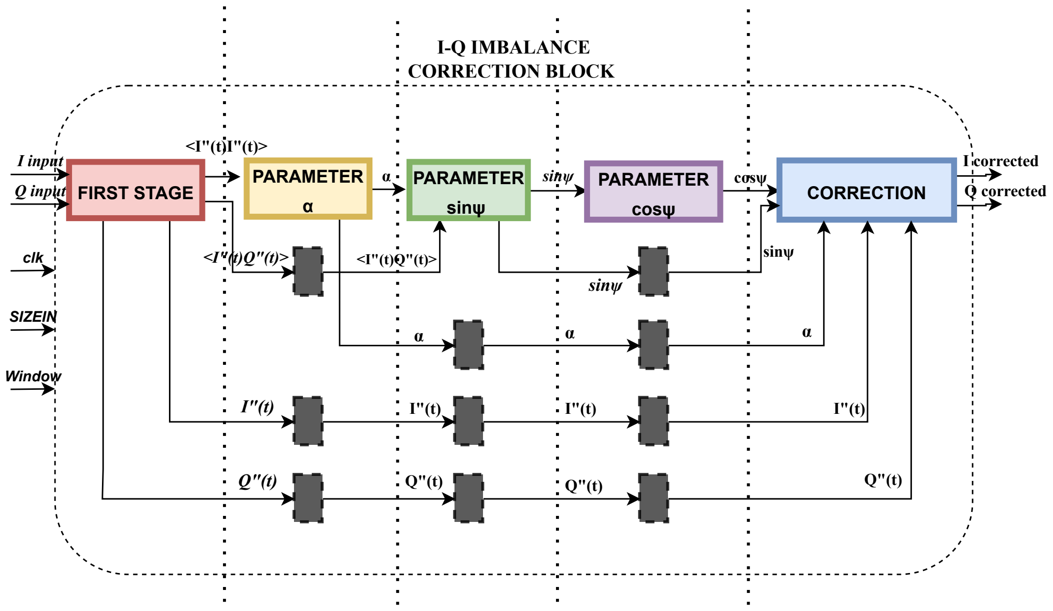
Πίνακας 3: Ποσοστό Σχετικού Σφάλματος μεταξύ I-Αρχικών δεδομένων και I-Matlab αποτελεσμάτων 16-QAM

16-QAM	Μέγεθος Παραθύρου			
Πλήθος δειγμάτων	64	128	256	1024
1024	41.5886	39.9195	39.1859	38.7407
131072	40.7508	38.7685	37.7118	36.8148
1048576	41.0351	38.9571	37.8650	37.0302

Πίνακας 4: Ποσοστό Σχετικού Σφάλματος μεταξύ Q-Αρχικών δεδομένων και Q-Matlab αποτελεσμάτων 16-QAM

Αρχιτεκτονική Αλγόριθμου

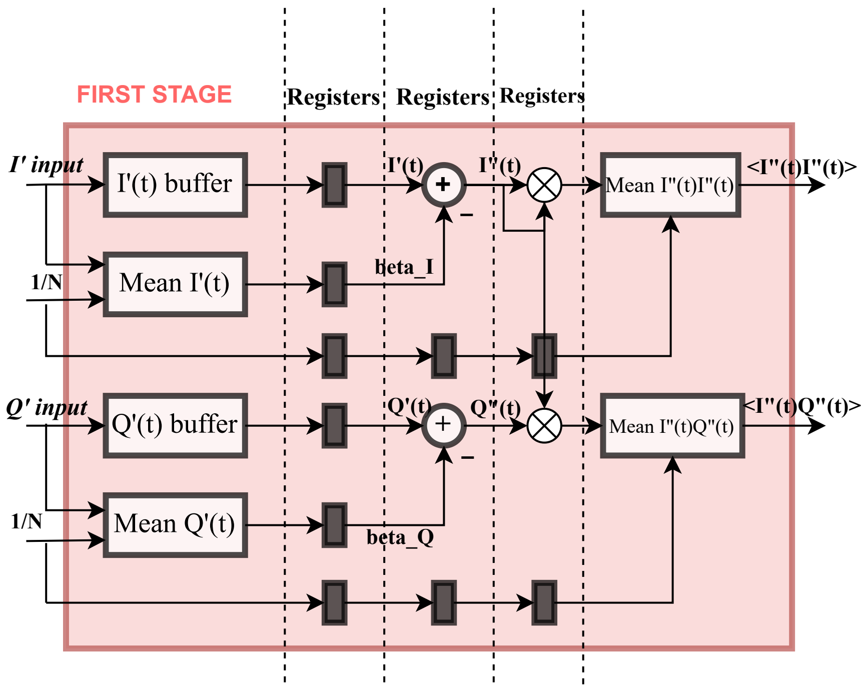
Αφού έχει γίνει η εκτίμηση του διορθωτικού αλγόριθμου σε επίπεδο MATLAB σε αυτό το σημείο εστιάζουμε στον σχεδιασμό και υλοποίηση σε γλώσσα υλικού VHDL. Παρακάτω παρουσιάζεται η αρχιτεκτονική του αλγορίθμου (Σχήμα 0.0.14 και στην συνέχεια εμβαθύνουμε σε κάθε μπλοκ ξεχωριστά.



Σχήμα 0.0.14: Αρχιτεκτονική Αλγόριθμου Διόρθωσης I/Q Imbalance

Πρώτο Στάδιο

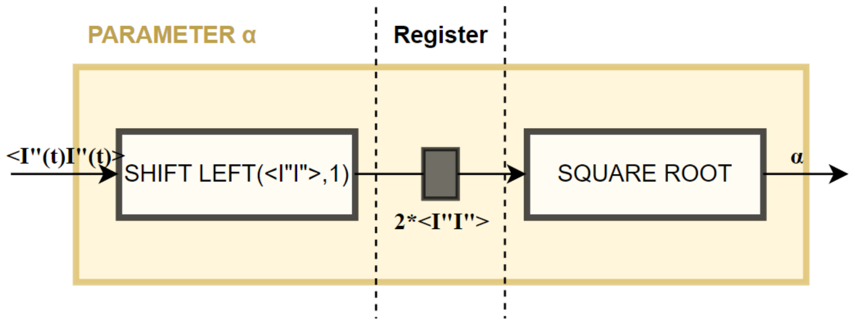
Στο πρώτο στάδιο, υπολογίζεται ο μέσος όρος των ανισόρροπων I/Q συμβόλων που εισέρχονται στο σύστημα. Ταυτόχρονα, αποθηκεύονται προσωρινά σε δύο ξεχωριστούς μπάφερ μέχρι να τελειώσει η διαδικασία του υπολογισμού του μέσου όρου για κάθε παράθυρο. Έπειτα, αφαιρούνται τα β_i και β_q από το κάθε μονοπάτι για να εξουδετερώσουμε τις DC πολώσεις. Η ίδια διαδικασία ακολουθείται για τον υπολογισμό των μέσων όρων των I”(t)I”(t), I”(t)Q”(t) (Σχήμα 0.0.15).



Σχήμα 0.0.15: Μπλοκ Πρώτου Σταδίου

Παράμετρος α

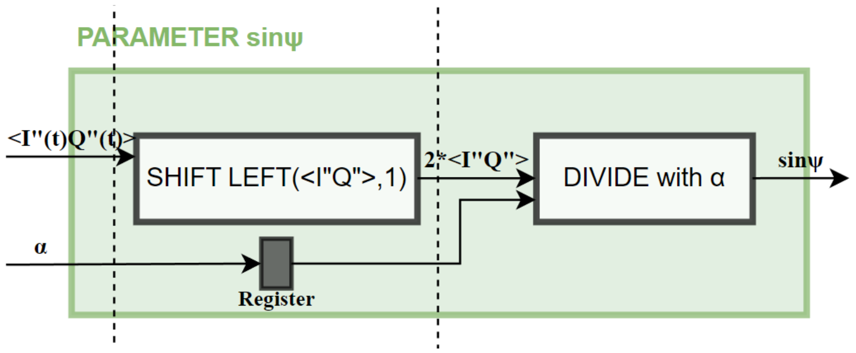
Για να υπολογιστεί η παράμετρος α, πολλαπλασιάζεται η τιμή $< I"(t)I"(t) >$ με 2 με το να μετατοπιστεί αριστερά κατά ένα μπιτ, και έπειτα με τετραγωνική ρίζα παίρνουμε ως αποτέλεσμα το α (Σχήμα 0.0.16).



Σχήμα 0.0.16: Μπλοκ Παραμέτρου α

Παράμετρος sin(ψ)

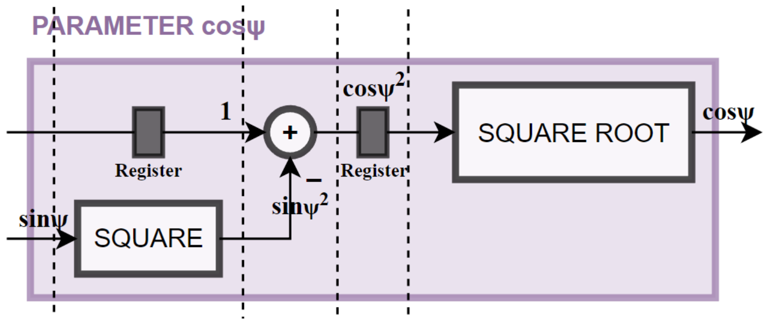
Αφού έχει υπολογιστεί η παράμετρος α και το $< I"(t)Q"(t) >$ υπολογίζεται το sin(ψ) με το να μετατοπιστεί αριστερά κατά ένα μπιτ το $< I"(t)Q"(t) >$, για να διπλασιαστεί, και έπειτα διαιρείται με την παράμετρο α (Σχήμα 0.0.17).



Σχήμα 0.0.17: Μπλοκ Παραμέτρου $sin\psi$

Παράμετρος cos(ψ)

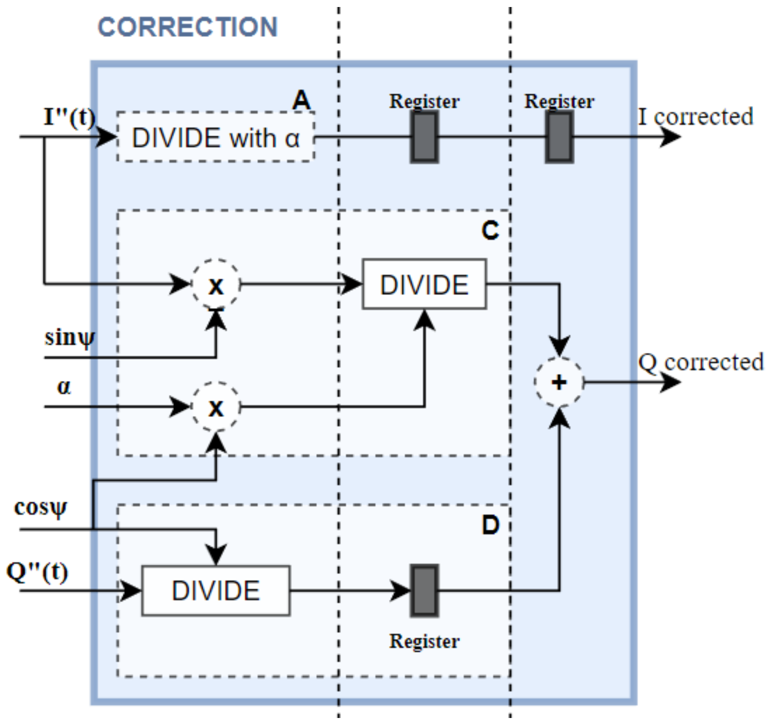
Χρησιμοποιείται η ταυτότητα $sin^2(\psi) + cos^2(\psi) = 1$ για να υπολογιστεί το συνημίτονο. Τετραγωνίζεται το ημίτονο και έπειτα αφαιρείται από το 1 και με τετραγωνική ρίζα καθορίζεται το συνημίτονο (Σχήμα 0.0.18).



Σχήμα 0.0.18: Μπλοκ παραμέτρου $cos\psi$

Μπλοκ Διόρθωσης

Το τελικό στάδιο είναι το στάδιο διόρθωσης. Σε αυτό δημιουργούνται οι παράμετροι πίνακα A,C,D που συμβάλουν στην διόρθωση των ανισόρροπων I,Q συμβόλων. Τα διορθωμένα I δεδομένα υπολογίζονται με το να διαιρεθούν με την παράμετρο α. Για την διόρθωση των Q δεδομένων: πολλαπλασιάζουμε το I”(t) με την παράμετρο *sinψ* και έπειτα διαιρείται με την παράμετρο α πολλαπλασιαζόμενη με την παράμετρο *cosψ*. Το πηλίκο αυτό είναι ίσο με την παραμέτρο C. Ταυτόχρονα, διαιρείται το Q”(t) με την παράμετρο *cosψ* που είναι ίσο με την παράμετρο D. Οι δύο παραμέτροι C, D προστίθενται και το άθροισμα είναι ίσο με το διορθωμένο Q (Σχήμα 0.0.19).



Σχήμα 0.0.19: Μπλοκ Διόρθωσης

Υλοποίηση μέσου όρου, μπάφερ, τετραγωνικής ρίζας και διαίρεσης

Για να υλοποιηθούν κάποια στάδια του αλγόριθμου στην VHDL χρειάστηκε να χρησιμοποιηθούν κάποια Language Templates και Math Functions από τον IP Catalog.

Συγκεκριμένα, για την υλοποίηση του μέσου όρου χρησιμοποιήθηκε το Multiply and Accumulate (macc) Language Template. Για την προσωρινή αποθήκευση των δεδομένων χρησιμοποιήθηκε η Simple Dual Port 1 clock Block

RAM Language Template με την λογική του First-In First-Out (FIFO). Για την τετραγωνική ρίζα επιλέχτηκε ο πυρήνας CORDIC, ο οποίος αρχικά είχε σχεδιαστεί για να εκτελεί περιστροφή διανύσματος για να υπολογίζει τριγωνομετρικές εξισώσεις και στην συνέχεια η χρήση του επεκτάθηκε για να λύνει διάφορες εξισώσεις όπως την τετραγωνική ρίζα. Όσο αφορά την διαίρεση, καθώς το σύμβολο ¨/' δεν υποστηρίζεται στο στάδιο σύνθεσης του Vivado, χρησιμοποιήθηκε ο LogiCORE IP Divider Generator. Ο συγκεκριμένος πυρήνας έχει 3 διαφορετικές υλοποιήσεις οι οποίες επιτρέπουν την εξισορρόπηση στην απόδοση, καθυστέρηση και χρήση πόρων. Στην παρούσα διπλωματική χρησιμοποιήθηκε η υλοποίηση Radix-2 καθώς επιτρέπει τον έλεγχο μεταξύ τις παραλληλίας στον αλγόριθμο και της σωστής αναλογίας μεταξύ απόδοσης και πόρους.

Επαλήθευση Σχεδιασμού

Σε αυτό το σημείο θα γίνει η επαλήθευση του σχεδιασμού που έγινε με την χρήση του Vivado Design Suite 2022.1. Η επαλήθευση θα γίνει με το να συγκρίνουμε τα VHDL αποτελέσματα με αυτά που παράχθηκαν στην αρχή και με τα αποτελέσματα από το Matlab με την χρήση του σχετικού σφάλματος.

Τα ποσοστά σχετικού σφάλματος μεταξύ των **Αρχικών** και των **VHDL** αποτελεσμάτων για τα I-Q σύμβολα για **4-QAM** παρουσιάζονται στους πίνακες 5 και 6 αντίστοιχα:

4-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	14.1893	12.7839	12.8234	12.7749
131072	14.7343	13.4889	13.0339	12.9647
1048576	14.7261	13.4071	13.0079	12.9773

Πίνακας 5: Ποσοστό Σχετικού Σφάλματος μεταξύ των I-Αρχικών Δεδομένων και I-VHDL αποτελεσμάτων 4-QAM

4-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	37.0367	34.8073	34.5408	34.4803
131072	38.7687	37.6134	37.1501	36.8886
1048576	38.9443	37.7006	37.2573	36.9872

Πίνακας 6: Ποσοστό Σχετικού Σφάλματος μεταξύ των Q-Αρχικών Δεδομένων και Q-VHDL αποτελεσμάτων 4-QAM

Τα ποσοστά σχετικού σφάλματος μεταξύ των **Αρχικών** και των **VHDL** αποτελεσμάτων για τα I-Q σύμβολα για **16-QAM** παρουσιάζονται στους πίνακες 7 και 8 αντίστοιχα:

16-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	21.0657	18.5471	17.8829	16.9261
131072	22.4395	19.7012	18.3275	17.3181
1048576	22.3737	19.6553	18.2851	17.3940

Πίνακας 7: Ποσοστό Σχετικού Σφάλματος μεταξύ των I-Αρχικών Δεδομένων και I-VHDL αποτελεσμάτων 16-QAM

16-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	41.5913	39.9255	39.2021	38.7646
131072	40.7499	38.7683	37.7141	36.8248
1048576	41.3597	38.9572	37.8665	37.0371

Πίνακας 8: Ποσοστό Σχετικού Σφάλματος μεταξύ των Q-Αρχικών Δεδομένων και Q-VHDL αποτελεσμάτων 16-QAM

Τα ποσοστά σχετικού σφάλματος μεταξύ των **Matlab** και των **VHDL** αποτελεσμάτων για τα I-Q σύμβολα για **4-QAM** παρουσιάζονται στους πίνακες 9 και 10 αντίστοιχα:

4-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	0.0058	0.0215	0.0426	0.1689
131072	0.0055	0.0217	0.0431	0.1714
1048576	0.0055	0.0217	0.0431	0.1714

Πίνακας 9: Ποσοστό Σχετικού Σφάλματος μεταξύ των I-Matlab Δεδομένων και I-VHDL αποτελεσμάτων 4-QAM

4-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	0.0562	0.1291	0.2568	1.0261
131072	0.0548	0.1244	0.2449	0.9666
1048576	0.0896	0.1243	0.2443	0.9652

Πίνακας 10: Ποσοστό Σχετικού Σφάλματος μεταξύ των Q-Matlab Δεδομένων και Q-VHDL αποτελεσμάτων 4-QAM

Τα ποσοστά σχετικού σφάλματος μεταξύ των **Matlab** και των **VHDL** α-ποτελεσμάτων για τα I-Q σύμβολα για **16-QAM** παρουσιάζονται στους πίνακες 11 και 12 αντίστοιχα:

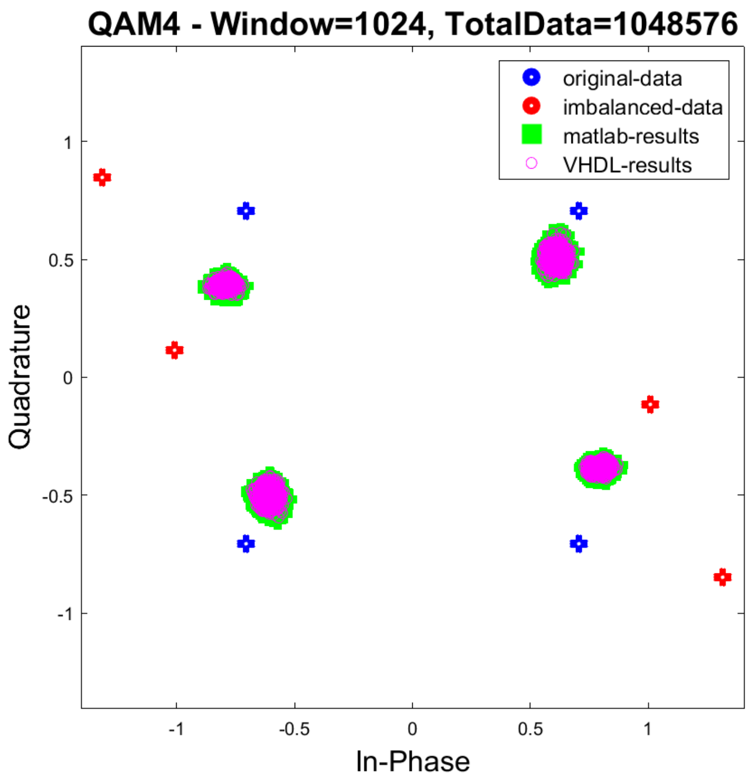
16-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	0.0173	0.0313	0.0606	0.2679
131072	0.0205	0.0350	0.0608	0.2707
1048576	0.0225	0.0341	0.0609	0.2718

Πίνακας 11: Ποσοστό Σχετικού Σφάλματος μεταξύ των I-Matlab Δεδομένων και I-VHDL αποτελεσμάτων 16-QAM

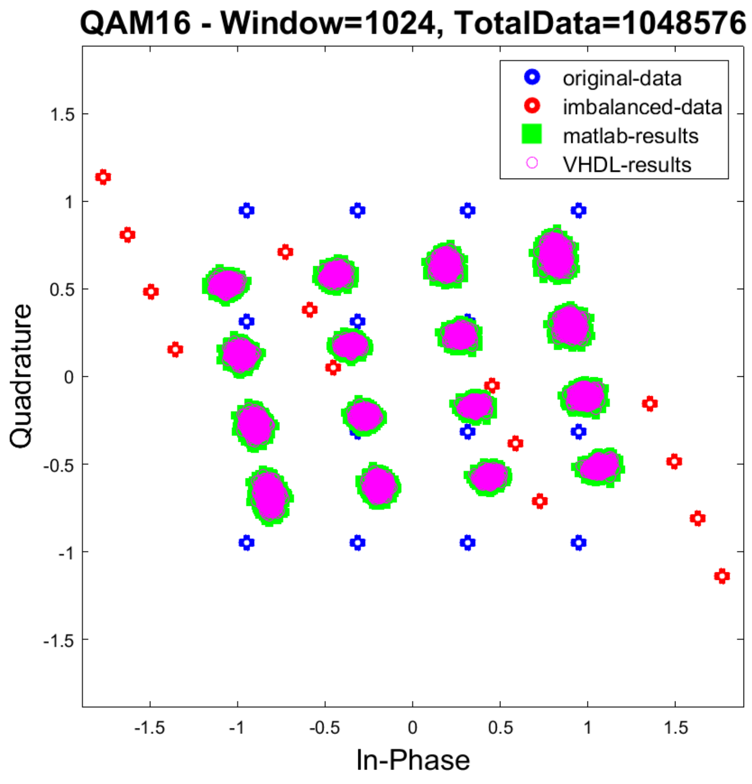
16-QAM	Μέγεθος Παραθύρου			
npoint	64	128	256	1024
1024	0.1019	0.1946	0.3806	1.4510
131072	0.2437	0.3252	0.5709	1.5526
1048576	0.7410	0.3814	0.4515	1.5479

Πίνακας 12: Ποσοστό Σχετικού Σφάλματος μεταξύ των Q-Matlab Δεδομένων και Q-VHDL αποτελεσμάτων 16-QAM

Το διάγραμμα αστερισμού μεταξύ των Αρχικών δεδομένων, των ανισόρροπων και των Matlab-VHDL αποτελεσμάτων για μέγεθος παραθύρου = 1024, npoint = 1048576 και 4-QAM και 16-QAM φαίνονται στα σχήματα 0.0.20 και 0.0.21 αντίστοιχα:



Σχήμα 0.0.20: 4-QAM αρχικά, με imbalance δεδομένα και MATLAB,FPGA αποτελέσματα



Σχήμα 0.0.21: 16-QAM αρχικά, με imbalance δεδομένα και MATLAB,FPGA αποτελέσματα

Τα resources που χρησιμοποιηθήκαν καθώς και η συχνότητα λειτουργίας του συστήματος φαίνεται στον Πίνακα 13

Table 13: Operating frequency and resource utilization of the system

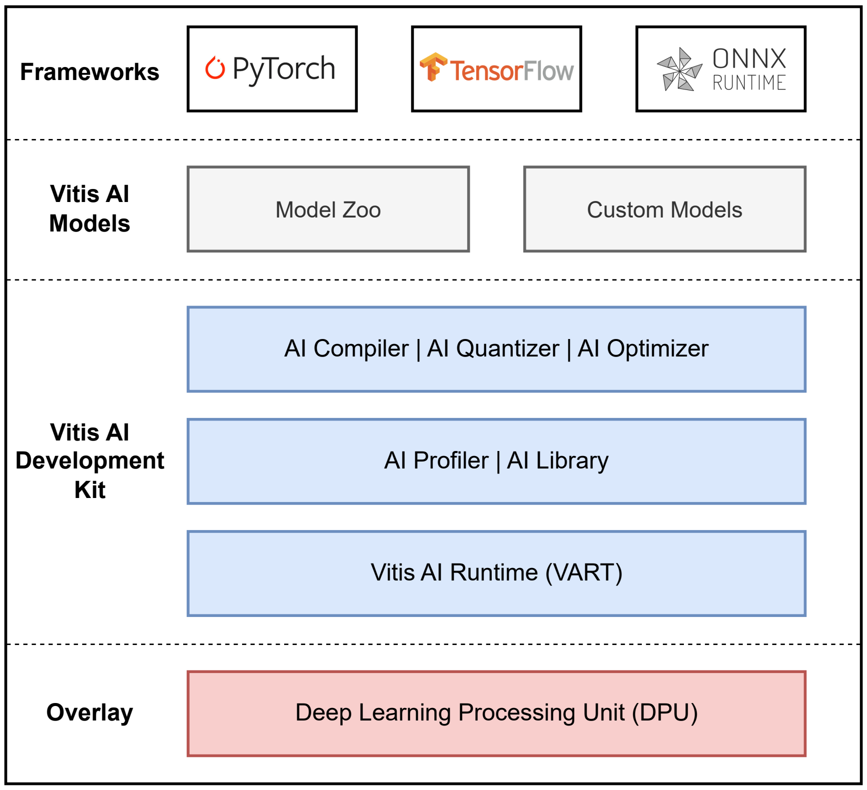
Window		**Resources** LUTs	LUTRAMs	DFFs	BRAMs	DSPs	**Max. Operating Freq.**
	Available	230400	101760	460800	312	1728	
64	Used	10063	260	26446	0	7	342MHz
	Utilization	4.37%	0.26%	5.74%	0%	0.41%	
128	Used	10133	312	26511	0	7	339MHz
	Utilization	4.40%	0.31%	5.75%	0%	0.41%	
256	Used	9876	104	26377	2	7	346MHz
	Utilization	4.29%	0.10%	5.72%	0.64%	0.41%	
1024	Used	9878	104	26381	4	7	340MHz
	Utilization	4.29%	0.10%	5.73%	1.28%	0.41%	

Αναγνώριση RF Διαμόρφωσης με το Vitis AI

Για την υλοποίηση της αναγνώρισης RF διαμόρφωσης στο FPGA χρησιμοποιήθηκε το Vitis AI περιβάλλον.

Vitis AI

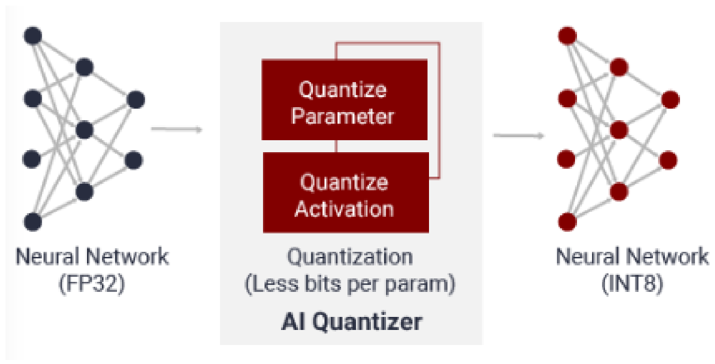
Πιο συγκεκριμένα το Vitis AI είναι ένα περιβάλλον ανάπτυξης που επιτρέπει την επιτάχυνση AI συμπερασμάτων σε πλατφόρμες υλικού της Xilinx. Στο Σχήμα 0.0.22 παρουσιάζονται τα κύρια στοιχεία που απαρτίζουν το περιβάλλον του Vitis AI.



Σχήμα 0.0.22: Δομή του Vitis AI

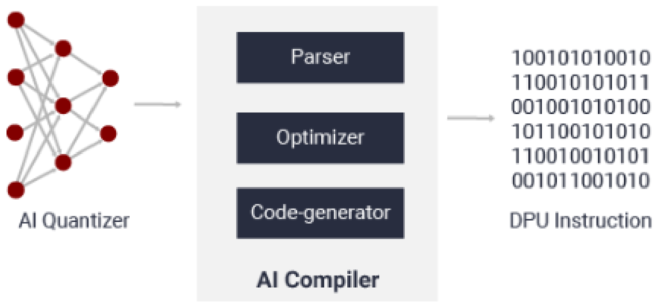
Στην παρούσα διπλωματική θα επικεντρωθούμε στα παρακάτω εργαλεία:

- Κβαντιστής Vitis AI: O AI κβαντιστής μετατρέπει τα βάρη και τις συναρτήσεις ενεργοποίησης από 32-bit floating-point σε 8-bit fixed-point ακεραίων, μειώνοντας την υπολογιστική πολυπλοκότητα χωρίς απώλεια ακρίβειας



Σχήμα 0.0.23: Κβαντιστής Vitis AI

- Μεταγλωττιστής Vitis AI: O μεταγλωττιστής μετατρέπει το κβαντισμένο μοντέλο νευρικού δικτύου σε ένα αποδοτικό σύνολο οδηγιών DPU

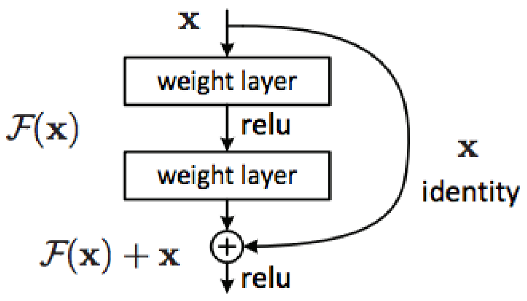


Σχήμα 0.0.24: Μεταγλωττιστής Vitis AI

- Deep Learning Processing Unit (DPU): Είναι ο πυρήνας υλικού του περιβάλλοντος ο οποίος είναι μια προγραμματιζόμενη μηχανή που είναι βέλτιστη για deep neural networks. Αποτελείται από ήδη υλοποιημένους IP πυρήνες που δεν χρειάζονται τοποθέτηση και δρομολόγηση. Η DPU χρησιμοποιείται για την επιτάχυνση του υπολογιστικού φόρτου εργασίας αλγόριθμων βαθιάς μάθησης, που εντοπίζονται σε συχνές εφαρμογές όρασης υπολογιστών.

Νευρωνικό Δίκτυο Αναγνώρισης RF Διαμόρφωσης

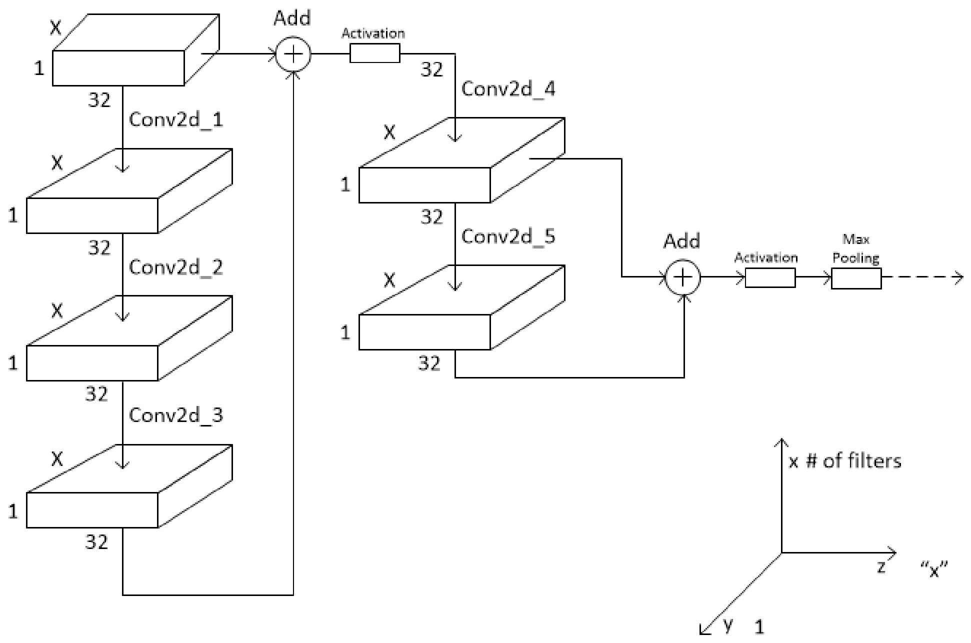
Για την υλοποίηση του νευρωνικού δικτύου που αναγνωρίζει τις RF Διαμορφώσεις χρησιμοποιήθηκαν residual neural networks (ResNet). Τα ResNets όπως φαίνονται και στο Σχήμα 0.0.25 περιέχουν συνδέσεις παράλειψης/συντομεύσεις που επιτρέπουν την μετάβαση σε μετέπειτα επίπεδα. Το συγκεκριμένο είδος νευρωνικών δικτύων διευκολύνει την εκπαίδευση βαθύτερων μοντέλων.



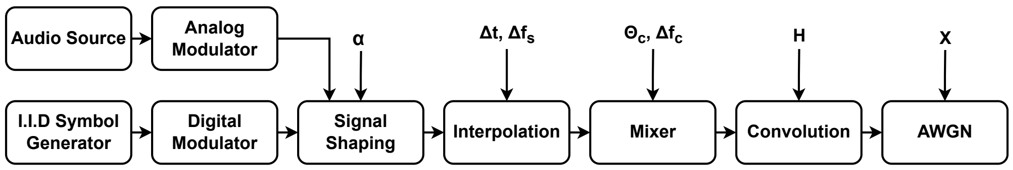
Σχήμα 0.0.25: Residual Training

Η διάταξη του κάθε ResNet stack παρουσιάζεται στο Σχήμα 0.0.26. Το μοντέλο στην συγκεκριμένη εφαρμογή περιέχει 4 ResNet stacks. Όπως είναι εμφανές στο σχήμα κάθε resnet stack περιέχει convolutional επίπεδα τα οποία περιέχουν convolutional neural networks με 32 φίλτρα στο κάθε επίπεδο. Επιπλέον, οι συνάρτησεις ενεργοποίησης είναι Rectified Linear Units (ReLUs).

Το dataset που χρησιμοποιήθηκε για το training/testing του παραπάνω μοντέλου περιέχει 24 διαφορετικές αναλογικές και ψηφιακές διαμορφώσεις. Οι διαμορφώσεις αυτές δημιουργήθηκαν με δύο διαφορετικούς τρόπους. Συγκεκριμένα, η πρώτη ήταν η δημιουργία ασύρματων καναλιών και εφαρμογή σε αυτά impairments οπώς φαίνεται στο Σχήμα 0.0.27. Η δεύτερη μέθοδος ήταν over-the-air μετάδοση καναλιών χωρίς impairments και φαίνεται στο Σχήμα 0.0.28.

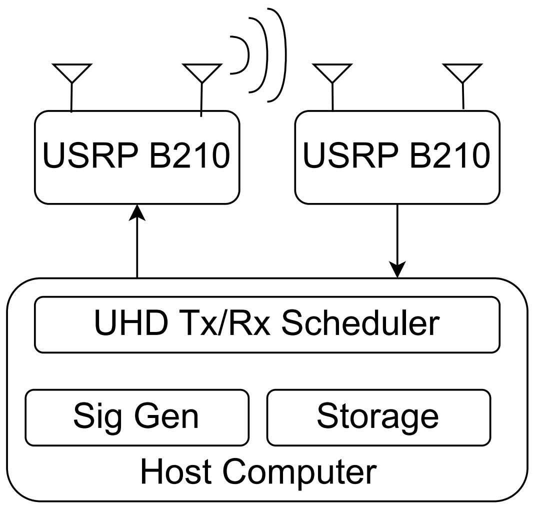


Σχήμα 0.0.26: ResNet Stack



Σχήμα 0.0.27: Το σύστημα για την παραγωγή των σημάτων του dataset και τα impairments του συνθετικού καναλιού

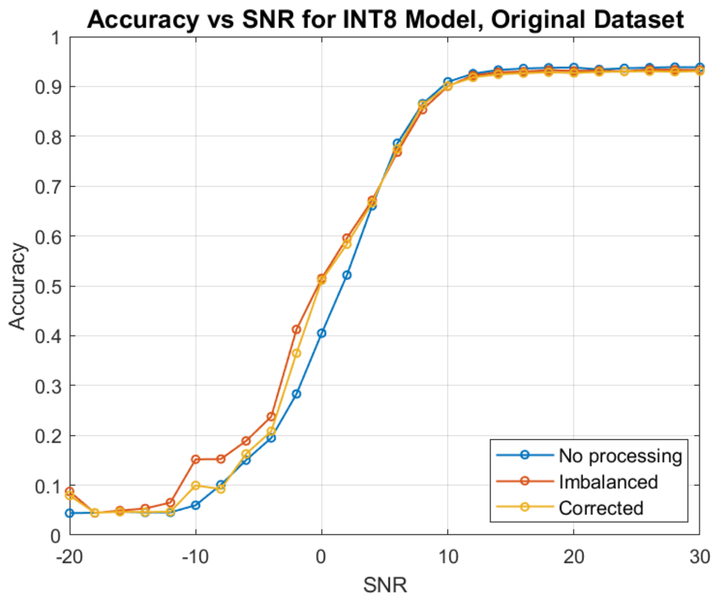
Τα signal-to-noise ratios (SNRs) για κάθε παράδειγμα που παράχθηκε κυμαίνεται απο -20dB σε +30dB. Οι συγκεκριμένες διαμορφώσεις που περιλαμβάνονται στο dataset είναι: OOK, 4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 32PSK, 16APSK, 32APSK, 64APSK, 128APSK, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM, AM-SSB-WC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM, GMSK, OQPSK.



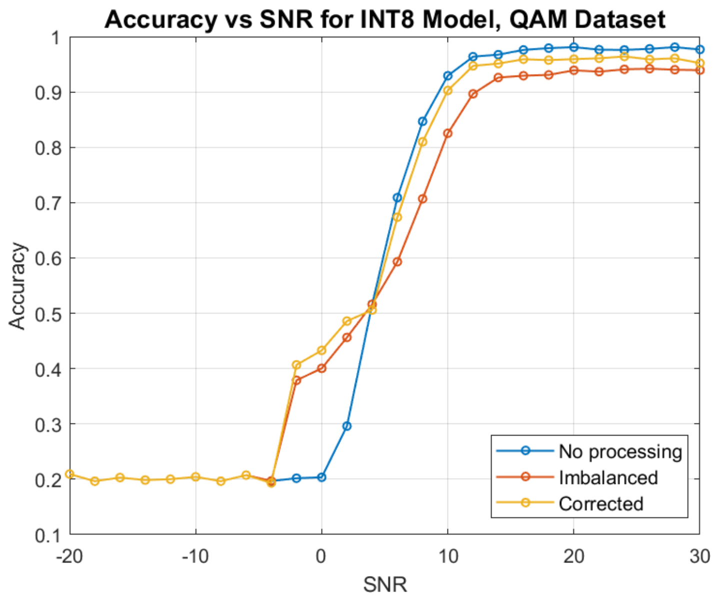
Σχήμα 0.0.28: Over-the-air test configuration

Αποτελέσματα Αναγνώρισης RF Διαμόρφωσης

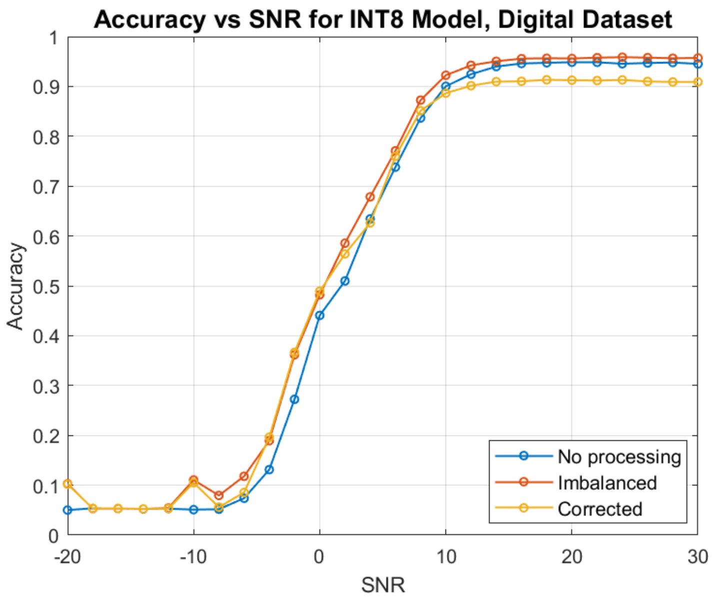
Αφού έχει γίνει εκπάιδευση του μοντέλου, το dataset που αναφέρθηκε παραπάνω χωρίστηκε σε 3 διαφορετικές παραλλαγές. Μία που περιέχει όλες τις διαμορφώσεις που αναφέρθηκαν παραπάνω, μία που περιέχει μόνο τις QAM διαμορφώσεις και μία με τις ψηφιακές διαμορφώσεις μόνο. Έπειτα συγκρίνονται τα αποτελέσματα ακρίβειας για 3 διαφορετικές εκδοχές του κάθε dataset. Η πρώτη είναι το dataset χωρίς επεξεργασία, στην δεύτερη εκδοχή έχει εφαρμοστεί 10dB imbalance πλάτους και 45°imbalance φάσης στα δεδομένα τις διαμόρφωσης 16-QAM για SNRs ίσα με [-20, -10, -2, 0, 2, 10, 20] dB και η τρίτη είναι τα διορθωμένα δεδομένα. Τα αποτελέσματα των 3 παραλλαγών του dataset για κάθε μία απο τις 3 εκδοχές, αφού έχει γίνει quantize μέσω του vitis-ai, φαίνονται στα Σχήματα 0.0.29, 0.0.30 και 0.0.31 αντίστοιχα.



Σχήμα 0.0.29: Accuracy vs SNRs για το αρχικό dataset

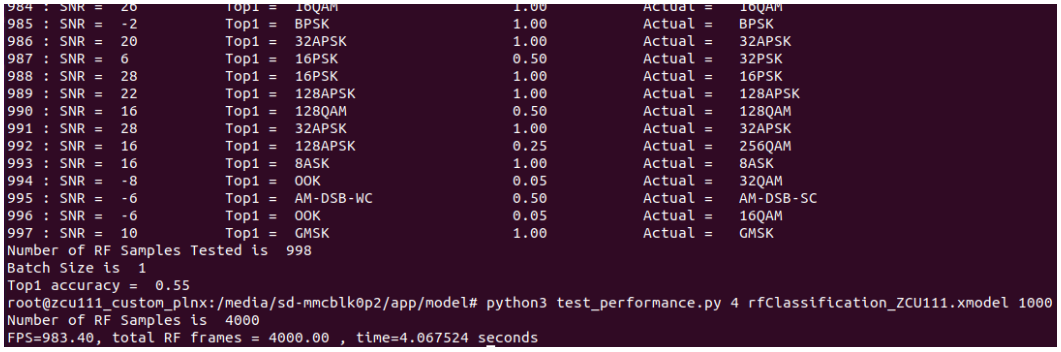


Σχήμα 0.0.30: Accuracy vs SNRs για το QAM dataset



Σχήμα 0.0.31: Accuracy vs SNRs για το ψηφιακό dataset

Μόλις το μοντέλο έχει γίνει compiled, τρέχουμε τα accuracy και performance tests για τα Original, QAM and Digital Dataset χωρίς επεξεργασία στην Zynq Ultrascale+ RFSoC ZCU111 πλακετα. Τα αποτελέσματα της πλακέτας φαίνονται στο 0.0.32 και τα υπόλοιπα αποτελέσματα τα παίρνουμε online στο περιβάλλον του Vitis-AI.



```
984 : SNR =  26          Top1 =  16QAM                          1.00          Actual =   16QAM
985 : SNR =  -2          Top1 =  BPSK                           1.00          Actual =   BPSK
986 : SNR =  20          Top1 =  32APSK                         1.00          Actual =   32APSK
987 : SNR =  6           Top1 =  16PSK                          0.50          Actual =   32PSK
988 : SNR =  28          Top1 =  16PSK                          1.00          Actual =   16PSK
989 : SNR =  22          Top1 =  128APSK                        1.00          Actual =   128APSK
990 : SNR =  16          Top1 =  128QAM                         0.50          Actual =   128QAM
991 : SNR =  28          Top1 =  32APSK                         1.00          Actual =   32APSK
992 : SNR =  16          Top1 =  128APSK                        0.25          Actual =   256QAM
993 : SNR =  16          Top1 =  8ASK                           1.00          Actual =   8ASK
994 : SNR =  -8          Top1 =  OOK                            0.05          Actual =   32QAM
995 : SNR =  -6          Top1 =  AM-DSB-WC                      0.50          Actual =   AM-DSB-SC
996 : SNR =  -6          Top1 =  OOK                            0.05          Actual =   16QAM
997 : SNR =  10          Top1 =  GMSK                           1.00          Actual =   GMSK
Number of RF Samples Tested is  998
Batch Size is  1
Top1 accuracy =  0.55
root@zcu111_custom_plnx:/media/sd-mmcblk0p2/app/model# python3 test_performance.py 4 rfClassification_ZCU111.xmodel 1000
Number of RF Samples is  4000
FPS=983.40, total RF frames = 4000.00 , time=4.067524 seconds
```

Σχήμα 0.0.32: Accuracy and Performance results for the Original Dataset with no processing on the ZCU111 board

Table 14: Accuracy and Performance results on the Zynq Ultrascale+ RF-SoC ZCU111 board

		Accuracy	**Performance**
Original	No processing	0.55	FPS=983.40, time=4.067524sec
	Imbalanced	0.57	-
	Corrected	0.56	-
QAM	No processing	0.58	FPS=981.11, time=4.077023sec
	Imbalanced	0.57	-
	Corrected	0.59	-
Digital	No processing	0.55	FPS=982.24, time=4.072324sec
	Imbalanced	0.57	-
	Corrected	0.55	-

Chapter 1

Introduction

Communication can be delineated as the transfer of information from one point to another in distinct places in time and location. The information can have numerous different types of formats such as audio, video, data files and more. However, there are many vital systems of communication in people's everyday lives, that do not include humans. Such examples include packet transfer between routers on the Internet, computer-to-computer as well as computer-to-peripherals communication, control signals in communication networks etc. [1] [2]

Nevertheless, why is the communication world going digital when the majority of information transmitted is analogue? The advantages of using digital transmission have been proven to outnumber those of the analog transmission and thus justifying its prevalence on the design of communication systems.[3] The surge in demand for data transmission, the higher reliability of digital electronics for signal processing and their lower cost, the ease to reduce the effects of noise and interference with the employment of channel coding and numerous more, illustrate why digital communications are the superior choice for communication systems and why they have replaced many corresponding analogue systems.[4]

The tremendous growth of information and wireless communication systems have caused the radio spectrum to be full due to the enormous increase of end-users. Thus, it is imperative to have stable, reliable and intelligent communication systems to cater these demanding needs. On top of that, there is a need to ameliorate the spectrum efficiency in order to have facility on the control of the network resources. [5] In optical communication systems, it is vital for the detectors to sample at high speeds and store the information locally for a more convenient processing. Hence, they require the employment of high-speed digital interfaces. Such digital interfaces can be found in Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs), whereas Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are not suitable. While CPUs provide satisfactory performance in sequential computing, their performance improvement depends on increased clock speed and even multicore CPUs cannot be used for sufficient parallelization. Consequently, in a large proportion of digital signal processing and machine learning applications, GPUs and FPGAs replace them. However, FPGAs have prevailed amongst the three(GPUs, ASICs, FPGAs) in the field of optical communications electronics on the grounds that, even though ASICs have the optimum performance and energy efficiency, they have an extravagant manufacture cost, cannot be used in general purpose applications and they require an extensive time for development. On the contrary, FPGAs have direct connection with high-speed interfaces, exploitation of parallel architectures for DSP algorithms and the option to turn the designs into ASICs if needed. [6]

In the past few years, studies have shown that in the transmitter and receiver chains of an optical communication system, numerous different physical layer algorithms can be strengthen or even replaced with Artificial Intelligence (AI) tools, such as machine learning algorithms and neural networks. [7] It has been proven that replacing algorithms which were used for decades by experts for radio modulation with not that complex convolutional neural networks, illustrate a much more impressing performance compare to the previous ones. [8]

1.1 Motivation and Thesis Objectives

This thesis contributes to the fields of digital communications and digital systems and is concentrated on a correction algorithm for imbalances on the receiver chain. More specifically, extensive research has been conducted on implementing the correction algorithm of I-Q imbalance on FPGAs and evaluating its accuracy by comparing the classification of the corrected Quadrature Amplitude Modulations(QAMs) and the imbalanced ones, using Deep Neural Networks.

In more detail, the current research targets the following:

- Studying and understanding when I-Q imbalances occur on the transmitter-receiver chain.

- Testing and Verifying the correction algorithm and its accuracy on QAM modulated data through MATLAB
- Creating an efficient implementation of the algorithm in Vivado and aiming at the utilization of minimum resources with high frequency
- Comparing the results from MATLAB and Vivado with the original data
- Apply I-Q imbalance to a dataset and then correcting it through MATLAB and Vivado
- Classifying the imbalanced and corrected dataset with Deep Neural Networks and run the model for accuracy results on the Zynq Ultrascale+ RFSoC ZCU111 board

1.2 Thesis Outline

In chapter 2, the theoretical background will be presented which is needed to comprehend digital communications, the diverse modulations used in telecommunication systems, the block diagram of a communication system, what is I-Q imbalance and at what stage of the transmitter-receiver chain it occurs. Furthermore, what are FPGAs and why we are employing them in this thesis. What is machine learning, how neural networks are used for classification, and how they can be utilized in telecommunications.

In chapter 3, the I-Q imbalance correction algorithm will be explained. Later on it will be tested and verified in MATLAB by generating 4-QAM and 16-QAM modulations, applying I-Q imbalance to their symbols and then correcting them. We are comparing the relative error with the original symbols to test its accuracy.

Next, in chapter 4 we will illustrate the architecture of the implementation of the algorithm on Vivado. The experimental results from Vivado will be presented as well as the hardware resources and its maximum operating frequency for different data windows. The VHDL results will be compared with the original data generated in MATLAB as well with the MATLAB results.

Following this, in chapter 5, the Vitis-AI development environment and its principal components which is utilized for this application will be presented as well as the neural network used for the RF modulation recognition will be analyzed.

In chapter 6, the experimental results will be presented. More specifically, the accuracy of the compiled model will be shown against the various SNRs of three different versions and variations of the dataset. Then, the performance and accuracy of these models will be verified on the Zynq Ultrascale+ RFSoC ZCU111 board.

Finally, chapter 7 shows the conclusions drawn from the results as well as some recommendations for future work based on this thesis.

Chapter 2

Theoretical Background

2.1 Digital Communication

Digital Communication can serve an analog or a digital signal, for instance, speech and images or text files respectively. Nevertheless, the needs of a communication system for each case are distinct. Moreover, recent communication systems offer a variety of services in order to deal with the demands of each signal.

2.1.1 Digital Communication System

Figure 2.1.1 presents the block diagram of a digital communication system, including the transmitter and receiver.

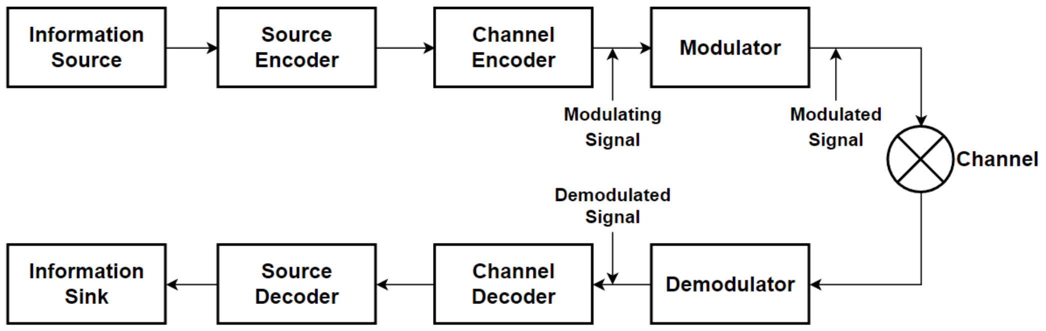


Figure 2.1.1: Block Diagram of a Digital Communication system

In the transmitter chain the following occur:

The source encoder is used to represent any information in digital form up to arbitrary precision and minimize the redundancy it may possess in order to meet the requirements of the receiver.

On the contrary, the channel encoder is responsible to apply controlled redundancy in the signal in order to be able to face potential errors that channel imperfections and noise might cause. The channel encoder generates a codeword which is able to meet the anticipated channel characteristics and requirements.

The modulator translates the output of the channel encoder, which are discrete symbols, into a format which can be transmitted over the physical channel, that format is an analog waveform.

The form of the channel used in a digital communication system can vary, and in order for the communication system to be efficient it is crucial to select the right model in each setting. For example, in wireline communication the channel used is extremely different from that used in a wireless mobile communication. In the former setting, a linear time-invariant system is employed and often feedback from the receiver to the transmitter can be obtained about the modulation used in the previous step. Whereas, for the latter, channel feedback to the transmitter is not common, hence, the channel must be estimated or otherwise utilize methods that do not demand exact channel estimation. Consequently, the channel may vary due to this mobility between the transmitter and receiver. Furthermore, in the latter setting, due to wireless being a broadcast medium, the suitable sharing mechanisms should be used to eschew simultaneous transmissions or the receivers need to be designed in a way to provide robust performance when interference occurs.

In the receiver part of the digital communication the following take place:

The demodulator targets the processing of the analog waveform received, which is "disturbed" with noise after its exit from the channel. Aside from this, it must synchronize the phase, frequency and time shifts with those in the transmitter and is also responsible for the channel equalization or compensation if an intersymbol interference takes place in the channel. In particularly, its main role is to take "hard" or "soft" decisions regarding the transmitted symbols.

These decisions are entered in the channel decoder,which uses the redundancy to represent the estimate of the sequence of the symbols that the channel encoder had as an input.

Finally, the source decoder transforms the estimated information, which was generated in the channel decoder, into a format that the receiver can understand, not necessarily in its original format. [1]

2.1.2 Classification of Digital Modulation Techniques

The reason as to why digital modulation techniques are preferred over analog ones is because they minimize the hardware, interference and noise issues. They require less bandwidth compared to analogue signals which include large waveforms and thus larger bandwidth for the information to be transmitted. [9]

The digital modulation techniques can be classified in three large categories:

- Amplitude Shift Keying (ASK)
  In Amplitude Shift Keying modulation, as the name suggests,the carrier amplitude is shifted between two different amplitude levels in order to represent the digital data. This is clearly illustrated in figure 2.1.2, where the first figure shows the input binary sequence and the second shows the output of the signal when it is modulated with an ASK. However, It is not a suitable choice for wireless or mobile applications since it has poor bandwidth efficiency, is extremely prone to noise and it has an established performance only in the linear region.

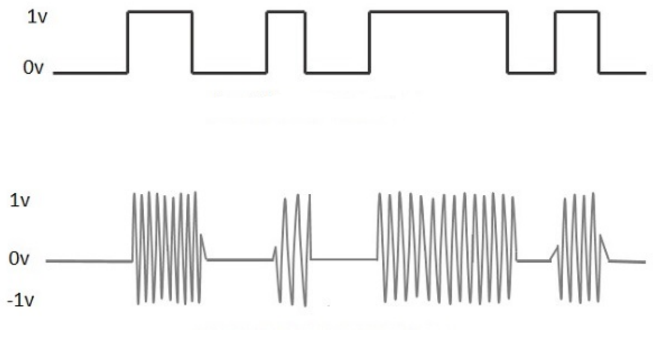


Figure 2.1.2: ASK Modulated output wave

- Frequency Shift Keying (FSK)
  When Frequency Shift Keying is used then the output wave is shifted between several different frequencies in order to represent the digital symbols. Fig.2.1.3 presents the input binary sequence and its output when FSK modulation is used. While, it is a cost effective and has

an easy implementation, it is not bandwidth efficient and has a lot of complexities in the receiver design.

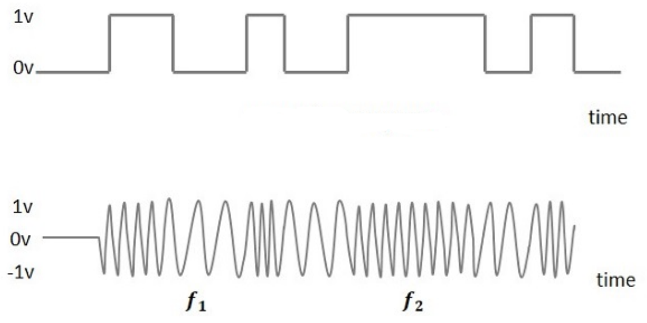


Figure 2.1.3: FSK Modulated output wave

- Phase Shift Keying (PSK)
  In the Phase Shift Keying modulation, the phase of a constant frequency carrier sine wave changes every time there is a change in the state of the input signal. Fig.2.1.4 illustrates the input binary sequence and the output PSK modulated wave.. It is robust and has simple implementation but is an inefficient user of the bandwidth. PSK is highly used in applications like wireless LANs and Bluetooth communication.

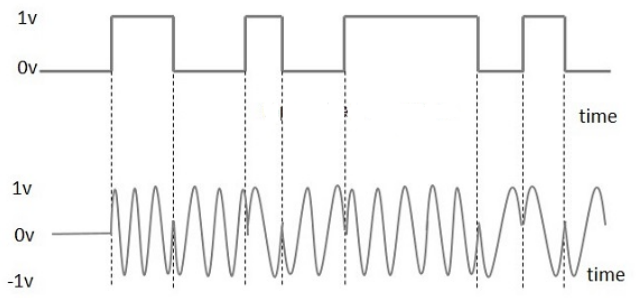


Figure 2.1.4: PSK Modulated output wave

Then the combination of two or three of the techniques lead to other popular modulation techniques such as:

- Quadrature Amplitude Modulation (QAM)
  When this modulation is used, two digital bit streams or two analog signals are transmitted be altering their amplitude, with the use of an ASK modulation. Simultaneously, the two carrier waves have the same frequency but have phase difference of 90°, which is a condition called quadrature. In QAM modulation the symbols can transmit more bits per symbol, for example, 8QAM utilizes four carrier phases and two amplitude levels to convey 3 bits per symbol. Similarly, 16QAM,64QAM and 256QAM transmit 4,6 ans 8 bits per symbol. This can be clearly seen in fig.2.1.5. [10]

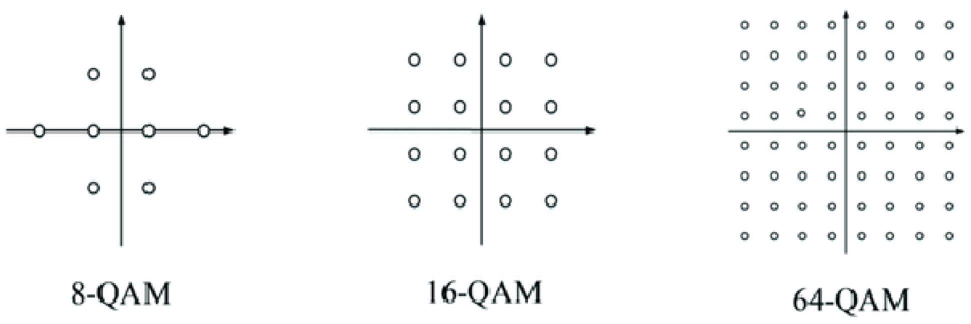


Figure 2.1.5: Different QAM Modulations

and many more.

2.2 I-Q imbalance

2.2.1 Direct-Conversion Receivers

I-Q imbalances can be found in direct-conversion receivers where there is no intermediate frequency f_{IF}, as can be seen in fig.2.2.1, and thus the received radio frequency(RF) or pass-band signal translates directly from the carrier frequency f_C to baseband with one mixing stage. The IQ imbalance happens because of the imperfections of the Local Oscillator (LO). If the phase difference between the quadrature waveforms are not exactly 90°or if their amplitudes are not equal, then I-Q imbalance occurs and causes a high bit

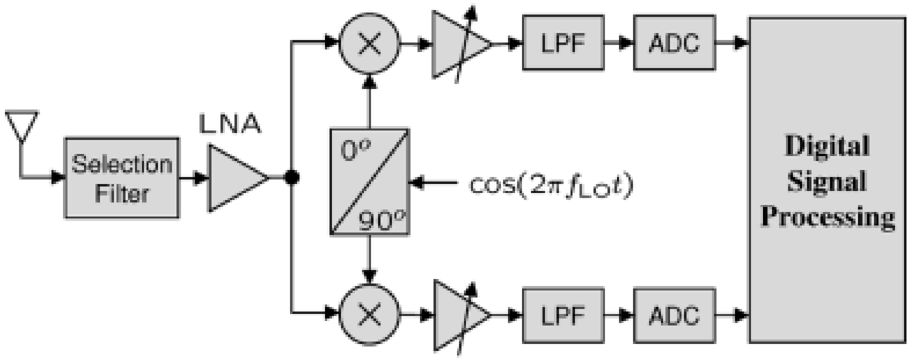


Figure 2.2.1: Architecture of Zero intermediate frequency (ZIF) receiver

error rate (BER).

2.2.2 Heterodyne Receivers

This is the main shortcoming of direct-conversion receivers compared to the heterodyne ones. In Heterodyne Receiver, the high carrier frequency f_C signal is multiplied by local oscillating (LO) signals to be transferred to intermediate frequencies f_{IF} appropriate for further amplification and processing and finally to the zero frequency (baseband). Fig.2.2.2 shows this architecture of a heterodyne receiver with the intermediate frequency f_{IF}. [11] [12]

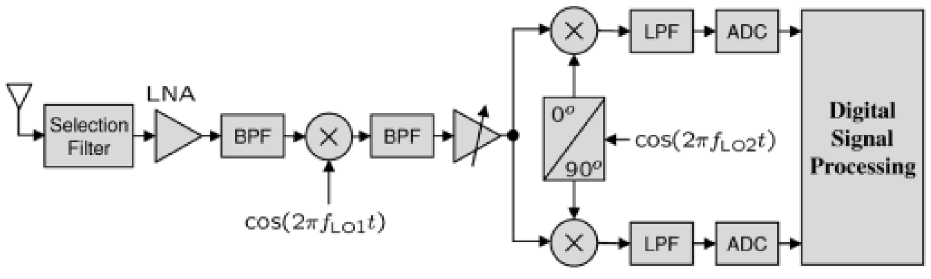


Figure 2.2.2: Architecture of Heterodyne receiver with intermediate frequency (IF)

2.2.3 Down-Conversion

In the down-conversion to the baseband, the RF signal is multiplied by the complex waveform $e^{-j2pif_{LO}}$, where f_{LO} is the frequency of the local oscillator at the receiver. In order for the complex down-conversion to happen, both sine and cosine waveforms must be present. The receiver is distinguished in I and Q branches, which are the real an imaginary part of the signal respectively. If after the down-conversion there is any mismatch between the I and Q branches, i.e they are not orthogonal (90°phase difference) or do not have the same amplitude, then this IQ imbalance will affect the performance of the system. [11]

2.3 Field-Programmable Gate Arrays (FPGAs)

Field-Programmable gate arrays are reprogrammable hardware chips used for digital logic. FPGAs are an array of logic gates that can be configured to build arbitrary digital circuits. The specification of these circuits is done using either circuit schematics or either with hardware description languages, Verilog or VHDL. [13]

2.3.1 FPGA architecture

FPGA consists of three main components:

- Programmable Logic Elements , which expresses a logic function
- Programmable I/O Elements, which provide an external interface
- Programmable Interconnect Element, which connects different parts of the board

Furthermore, on a FPGA there are digital signal processing (DSP) units, embedded memory which facilitates the calculation ability and phase-locked loop (PPL) or delay-locked loop (DLL) which supply the board with a clock network. The figure 2.3.1 presents the schematic of a FPGA. When supplied with the design data, these elements can implement on the FPGA the desired digital circuit. The logic block and multiplier block are utilized as hardware resources to comprehend logic functions. The memory blocks are responsible for storage. Multiplier and memory blocks are called "Hard Logic", whereas when logic blocks are used to implement function they are called "Soft Logic". Each FPGA manufacturing company has a different name for the logic blocks, for example Xilinx calls them configurable logic

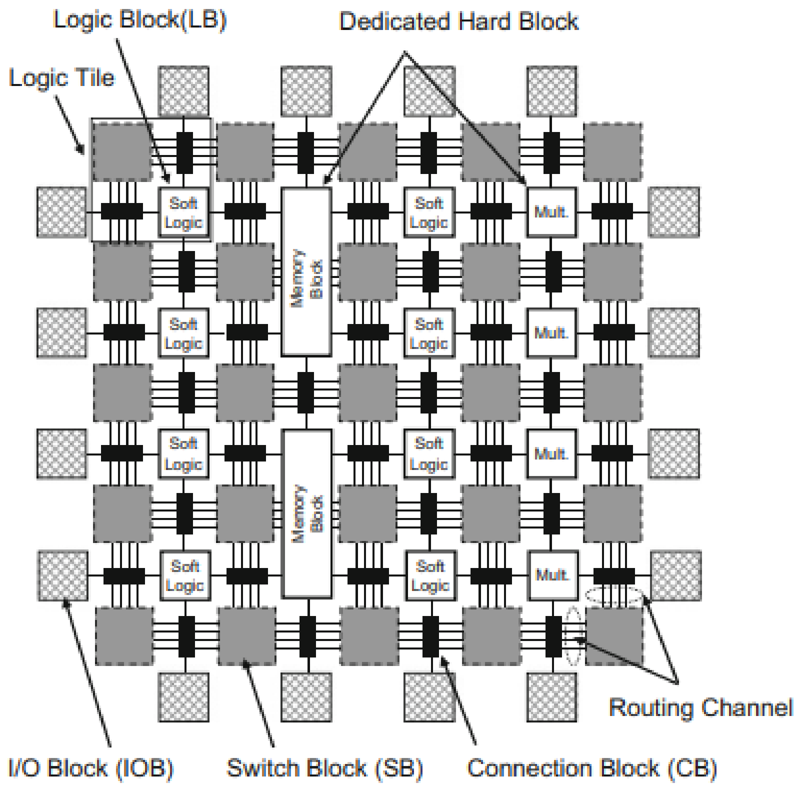


Figure 2.3.1: Architecture of a FPGA

blocks (CLB) while Altera(now part of Intel) gives them the name logic array blocks (LAB). [14]

2.3.2 FPGAs vs ASICs, GPUs and CPUs

Recently, FPGAs are utilized for hardware acceleration, where a FPGA accelerates specific parts of an algorithm and divide part of the computation between the FPGA and a generic processor. For example, the search engine Bing has adopted FPGA acceleration for its search algorithm [15]. Numerous industrial applications such as wearable computing, sensorless motor drive control etc, that require microcontrollers or DSPs and peripherals that are implemented with programmable devices, could be done with FPGAs. In the past few years, FPGAs are used more often as AI accelerators as well as for accelerating artificial neural-networks for diverse machine learning applications. The main reasons FPGAs have become so popular in the industry are:

- FPGAs have the possibility to be reconfigured, while their soft/hard IP cores are configured for a precise need, they can be reconfigured multiple times after installation, whereas ASICs cannot do this.

- FPGAs are able to process large amount of data in parallel, which makes them an excellent tool for edge computing applications due to their low latency. CPUs/GPUs on the other hand, operate sequentially and cannot process information simultaneously.

- Due to their low-latency they can be employed for time-demanding applications such as Software-Defined Radio, medical devices etc. Admittedly, ASICs could have less latency, yet they are developed for a specific reason.

- FPGAs have impressively better performance per watt compared to CPUs or GPUs. Even though ASICs can beat this performance, the extravagant manufacturing cost renders this argument as unjustified.

Indeed, each application has different requirements and ASICs, GPUs or CPUs could be better suited for these, nevertheless, FPGAs have an excellent trade-off of price- processing power- configurability- efficiency.[16] [17] This can be seen in Figure 2.3.2.

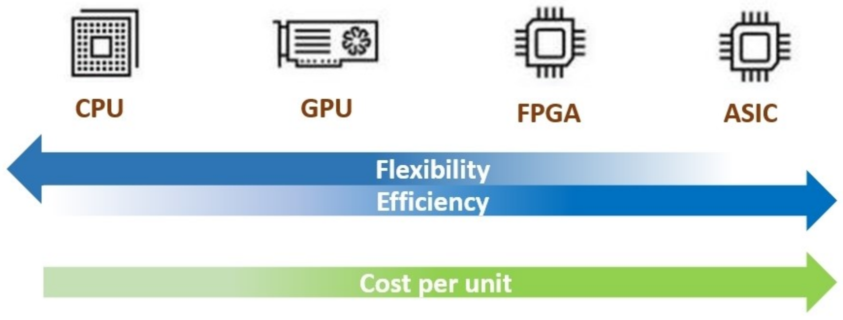


Figure 2.3.2: Trade-offs of CPU, GPU, FPGA and ASIC

2.4 Machine Learning

Machine Learning utilizes computational methods to improve the performance of a system through experience. It imitates human intelligence by learning from its surrounding environment. Experiences, in a computer system, exist in the form of data and the principal goal of machine learning is to generate learning algorithms in order to construct models based on data. The model that can make predictions occur by feeding the learning algorithm with experience data, based on new observations. Machine learning techniques have been applied with success in many fields ranging from pattern recognition, computer vision, spacecraft engineering, finance, entertainment, biomedical and medical applications and numerous more. [18] [19]

2.4.1 Process of training and testing a ML algorithm

Machine Learning algorithms achieve a desired task without the need to be literally programmed to produce the desired outcome. They are essentially "soft coded" since these algorithms can automatically adapt or change their architecture using repetition, in order to improve their ability to achieve the particular result.
The process of adaptation to create models from specific data is named learning or training. The data employed in the training phase are called training data and these samples include input data along with desired outcomes. With repetition and the training inputs, the algorithm learns to configure itself to not only generate the wanted result but to be able to produce the desired outcome from new data, not used in the training phase.
The process of making predictions for new fed data with a learned model is called testing. The samples used to make these predictions are names testing samples.
Besides predictions, another way used for learning is clustering. For instance, the data fed could share some underlying concepts to assist in further processing, in other words, clustering provides data insights that form the basis for the imminent analysis. [18] [19]

2.4.2 Categories of ML Algorithms

Depending on the nature of the training data, machine learning algorithms can be categorized in three approaches as can be seen in Figure 2.4.1:

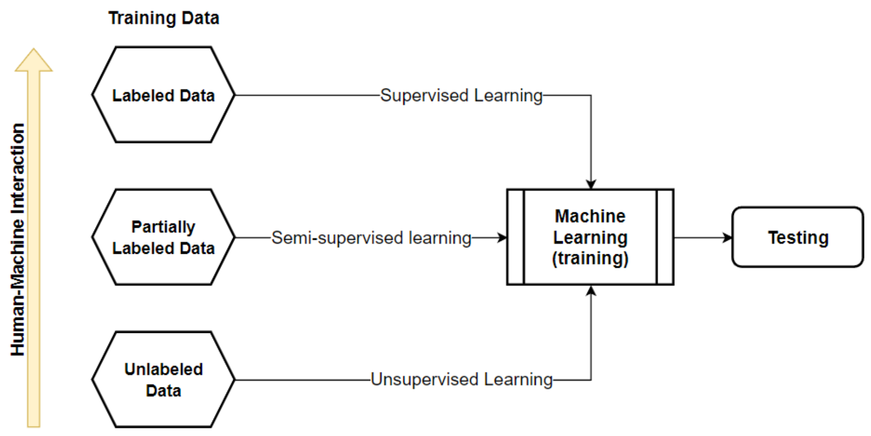


Figure 2.4.1: Categories of Machine Learning algorithms based on the nature of the training data

Supervised Learning

Supervised machine learning is when there is a predetermined output attribute alongside the use of input attributes. This kind of algorithms try to predict and classify the predetermined attribute. The accuracy and misclassification as well as their performance measures depend on the number of times the predetermined attribute is predicted correctly. It is also worth noting that, the training/learning process ends when the algorithm reaches a sufficient level of performance, which is prespecified by humans. [20] [21]

Unsupervised Learning

Unsupervised machine learning algorithms, are those that do not have a target attribute when trained. This technique is appropriate for clustering and association mining techniques due to the fact that all variables involved are utilized as inputs. Thus, they create by their own labels in the data given and then they are used to execute supervised learning. More specifically, unsupervised clustering algorithms realize and identify similarities/groupings between the unlabeled data and allocate to each a label. While, unsupervised association mining algorithms recognize rules that depicts relationships among the attributes. [20] [22] [23]

Semi-Supervised Learning

In semi-supervised machine learning, a proportion of the data is labeled and the rest are unlabeled. In this case, the labeled data assist in the learning/-training of the unlabeled part. This scenario is the most comparable with how humans develop their skills. [19]

The main difference between the supervised and unsupervised technique can be seen in Figure 2.4.2.

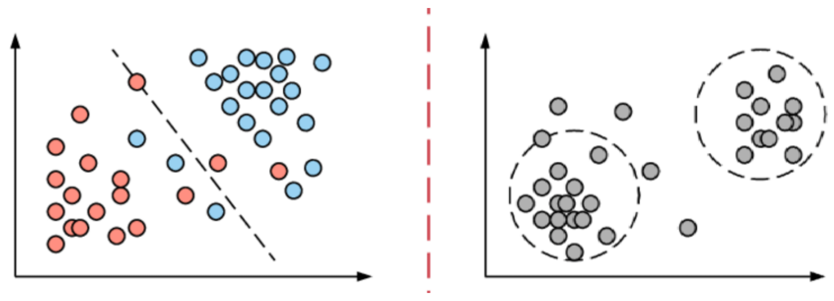


Figure 2.4.2: Supervised and Unsupervised Learning(clustering)

2.5 Artificial Neural Networks

Recently, Artificial Neural Networks(ANNs) are favored for classification, clustering, pattern recognition and prediction. ANNs are competitive opponents of conventional regression ans statistical models when it comes to usefulness. They can be evaluated in terms of accuracy, processing speed, latency, performance, fault tolerance, volume, scalability and convergence. The capability of ANNs to process in high-speeds due to their parallel implementation, justifies their popularity in numerous diverse applications. [24]

2.5.1 Artificial Neural Network Architecture

An ANN consists of an input layer of neurons(nodes), some hidden layers of neurons and a final layer for the output neurons. The typical architecture of an ANN where the neurons are connected is presented in Figure2.5.1.
Each node is associated with the neuron is connected with through a numeric number called weight. The output y_i of a neuron i in the hidden layer has

the following linear function:

$$y_i = \sigma(\sum_{j=1}^{N} w_{ij}x_j + b_i)$$

where σ is the activation function , N the number of input neurons, w_{ij} the weights, x_j inputs and b_i the bias of the hidden neuron. [25]

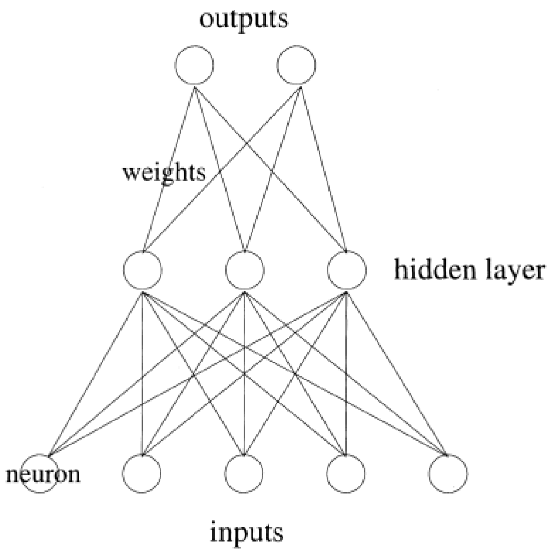
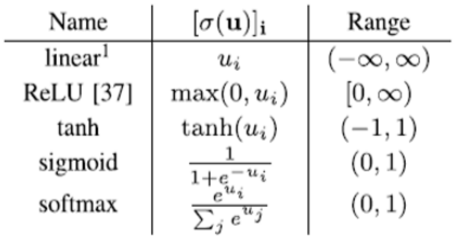


Figure 2.5.1: Architecture of a typical Artificial Neural Network

The activation function can be any of the following functions in Figure 2.5.2:



Name	$[\sigma(\mathbf{u})]_\mathbf{i}$	Range
linear[1]	u_i	$(-\infty, \infty)$
ReLU [37]	$\max(0, u_i)$	$[0, \infty)$
tanh	$\tanh(u_i)$	$(-1, 1)$
sigmoid	$\frac{1}{1+e^{-u_i}}$	$(0, 1)$
softmax	$\frac{e^{u_i}}{\sum_j e^{u_j}}$	$(0, 1)$

Figure 2.5.2: List with activation functions

2.5.2 Convolutional Neural Network

In the past few years, Convolutional Neural Networks have revolutionary results in diverse fields that have to do with patter recognition. The greatest contribution of CNNs is that they decrease the number of parameters in ANNs, which is why it is now possible to tackle larger and more complex tasks. [26]. The architecture of a CNN is greatly inspired by the natural visual perception mechanism of living creatures.
CNNs have had many variations in terms of their architecture in the literature. Nevertheless, the classic components that compose a CNN are convolutional, pooling and fully-connected layers. These layers are evident in Figure 2.5.3. [27]

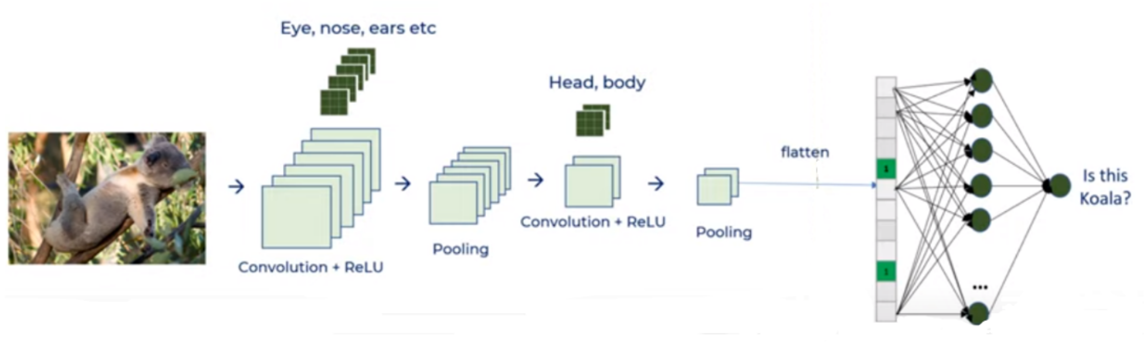
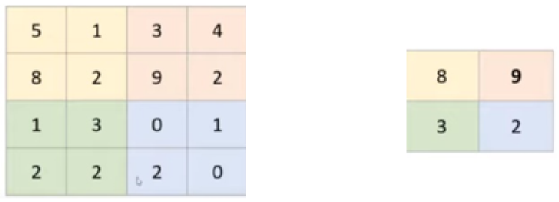
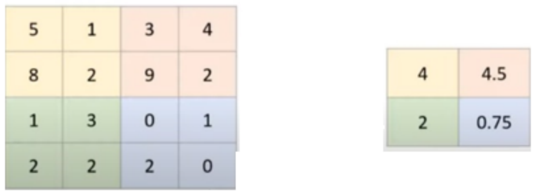


Figure 2.5.3: Basic Architecture of a Convolutional Neural Network

As can be seen in Figure 2.5.3 a convolution layer includes several convolution kernels which are utilized to process different feature maps. These layers strive to learn feature representations of the inputs. The new feature map is acquired by convolving the input with a learned kernel and afterwards, apply an activation function on the convolved results. These activation functions can be any of those presented in Figure 2.5.2.
The pooling layer's purpose is to apply shift-invariance by decreasing the resolution of the feature maps. Each feature of a pooling layer is connected to its preceding respectively convolutional layer feature map. The most common pooling operations are max and average pooling as can be seen in Figure 2.5.4 [27]
After the convolutional and pooling layers usually a fully-connected layer is present in order to execute high-level reasoning.



(a) Max Pooling



(b) Average Pooling

Figure 2.5.4: Types of Pooling Operations

2.5.3 Why Artificial Neural Networks in Communications?

It is irrefutable that the largest proportion of signal processing algorithms in communications are based on statistics and information theory, and are mostly proved to be optimal for adjustable mathematically models. However, most of these are usually linear, stationary and have Gaussian statistics. But in a real-life system there are imperfections and non-linearities. This is why a communication system which is based on Deep-Learning that does not demand a mathematically adjustable and can be optimized based on its targeted hardware configuration and channel, could overcome these imperfections.
On top of that, in communications is very common to split the signal processing into a chain of independent blocks. Each of these blocks are responsible for a single well-defined and isolated function, yet it is not clear whether these blocks individually succeed in the best end-to-end performance. However, if we have a learned end-to-end communication system then this facilitates the design of its structure and optimizes the end-to-end performance.
Furthermore, the execution of Neural Networks is highly parallelized and implemented with low-precision data types. Consequently, "learned" algo-

rithms with this form could be executed much faster and with less energy cost than their usual ”programmed” counterparts. [28]

Chapter 3

I-Q Imbalance Correction

As mentioned in chapter 2.2 IQ imbalance occurs in direct conversion receivers where there is zero-intermediate frequency(IF) and is generated when the in-phase component and quadrature-phase component do not have orthogonality. [12] [29]

3.1 Algorithm Analysis

3.1.1 Amplitude and Phase Imbalance

Assuming the I and Q outputs of an analog direct conversion receiver when converting a single tone RF waveform to baseband, are :

$$I(t) = \cos(\omega t) \tag{3.1.1}$$

$$Q(t) = \sin(\omega t) \tag{3.1.2}$$

where ω is the baseband frequency of the single tone. We set the amplitude equal to unit and the phase equal to zero since they do not affect the correction algorithm.
However, in a practical direct conversion receiver the I and Q outputs are:

$$I'(t) = \alpha \cos(\omega t) + \beta_i \tag{3.1.3}$$

$$Q'(t) = \sin(\omega t + \psi) + \beta_q \tag{3.1.4}$$

where ψ is the phase error, allocated in the Q output, α is the amplitude error, which is assigned to the I output and β_i and β_q are the DC biases that occur in each symbol.

3.1.2 Correction Procedure

The DC biases β_i and β_q are the mean of $I'(t)$ and $Q'(t)$ over an integer number of periods respectively. Thus, removing the DC biases is done by calculating the mean of each path($I'(t)$ and $Q'(t)$) and subtracting it from the corresponding path. Consequently, the outputs are now:

$$I"(t) = \alpha \cos(\omega t) \tag{3.1.5}$$

$$Q"(t) = \sin(\omega t + \psi) \tag{3.1.6}$$

Rewriting the above in the form of a matrix we get:

$$\begin{bmatrix} I"(t) \\ Q"(t) \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} I(t) \\ Q(t) \end{bmatrix} \tag{3.1.7}$$

Utilizing the following trigonometric identity:

$$sin(\omega t + \psi) = sin(\omega t)cos(\psi) + cos(\omega t)sin(\psi) \tag{3.1.8}$$

Inversing the above matrix we get the corrected outputs:

$$\begin{bmatrix} I(t) \\ Q(t) \end{bmatrix} = \begin{bmatrix} \alpha^{-1} & 0 \\ \alpha^{-1}\tan(\psi) & \sec(\psi) \end{bmatrix} \begin{bmatrix} I"(t) \\ Q"(t) \end{bmatrix} \tag{3.1.9}$$

The last thing needed to find $I(t)$ and $Q(t)$ is to calculate α and ψ. We employ the following equation to find α:

$$< x(t) >= \frac{1}{NT} \int_{t-NT}^{t} x(u)\, du \tag{3.1.10}$$

where T is the period $\frac{2\pi}{\omega}$ and N is any integer greater than zero. Then we get:

$$< I"(t)I"(t) >= \alpha^2 < cos^2(\omega t) >= \alpha^2 < \frac{1}{2} + \frac{1}{2}cos(2\omega t) >= \frac{1}{2}\alpha^2 \tag{3.1.11}$$

and similarly:

$$< I"(t)Q"(t) >= \frac{1}{2}\alpha sin(\psi) \tag{3.1.12}$$

As a result, equation 3.1.11 can be used to find α and equation 3.1.12 to find sin(ψ) which then we can use to directly calculate cos(ψ).

3.1.3 Summarizing the Algorithm

To recap, the detailed steps to execute the I-Q imbalance correction are [30]:

1. Calculate the mean of $I'(t)$ and $Q'(t)$: $\beta_I =< I'(t) >$ and $\beta_Q =< Q'(t) >$

2. Subtract the DC biases: $I"(t) = I'(t) - \beta_I$ and $Q"(t) = Q'(t) - \beta_Q$

3. Calculate parameter amplitude error: $\alpha = \sqrt{2 < I"(t)I"(t) >}$

4. Compute $\sin(\psi) = \frac{2}{a} < I"(t)Q"(t) >$

5. Directly obtain: $\cos(\psi) = \sqrt{1 - \sin^2(\psi)}$

6. The correction matrix is:

$$\begin{bmatrix} I(t) \\ Q(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & D \end{bmatrix} \begin{bmatrix} I'(t) - \beta_I \\ Q'(t) - \beta_Q \end{bmatrix} \tag{3.1.13}$$

   where the correction parameters A,C and D are:

$$A = \frac{1}{\alpha}$$

$$C = \frac{-\sin(\psi)}{\alpha\cos(\psi)}$$

$$D = \frac{1}{\cos(\psi)}$$

3.2 IQ imbalance algorithm evaluation

In this section, we present the steps taken to implement the correction algorithm in MATLAB for three different sizes of samples and four different processing windows for each sample. We test and verify the algorithm on QAMs and more specifically for 4-QAM and 16-QAM.

3.2.1 QAM generation and IQ imbalance application

In order to generate the input QAM data for the algorithm we employ the function *qammod* from MathWorks. We load it with an integer data sequence for modulation order 4 and 16. The function is presented in 3.2.1.

$$y = qammod(randi([0\ M-1], npoint, 1), M, 'UnitAveragePower', true) \tag{3.2.1}$$

where M is the modulation order, npoint is the number of samples and we set the Unit Average Power to true in order to scale the constellation output to the average power of one watt referenced to 1 Ohm. The output of this function plotted with scatterplot in MATLAB can be seen in Figures 3.2.1 3.2.2.

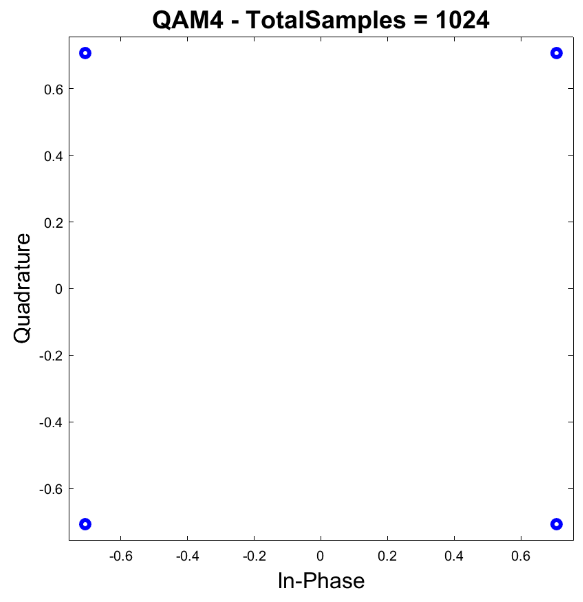


Figure 3.2.1: 4-QAM original

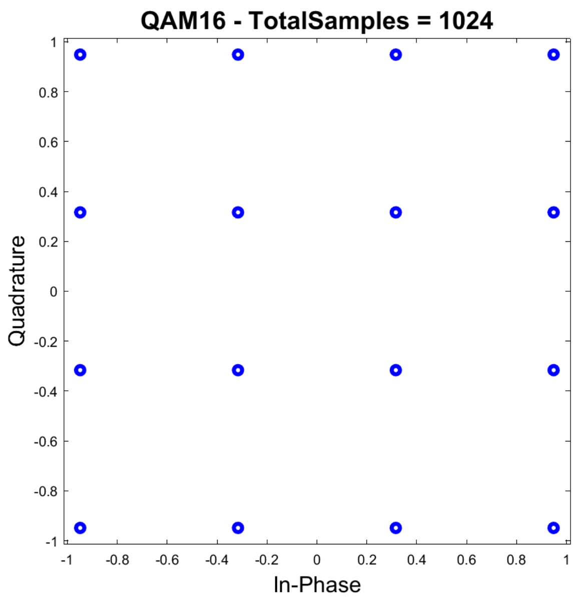
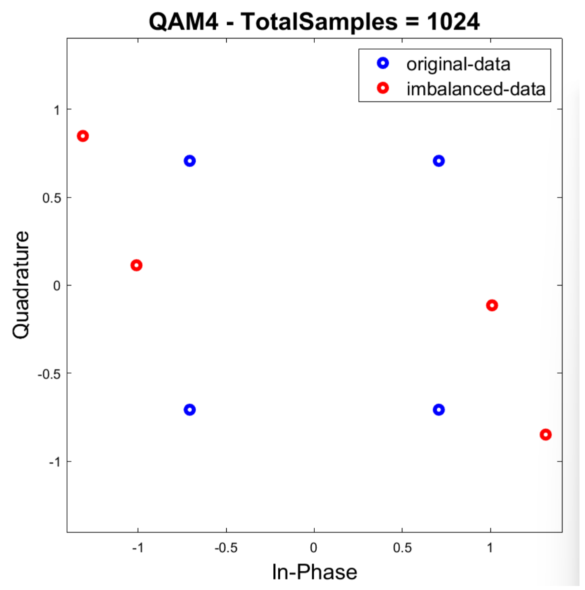


Figure 3.2.2: 16-QAM original

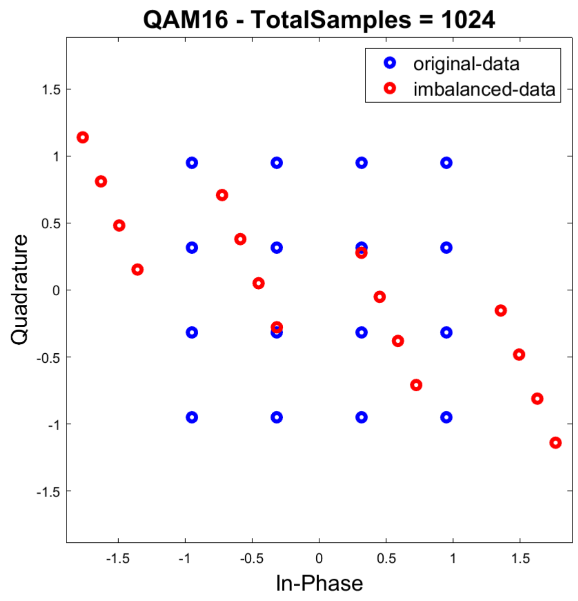
Then we applied I/Q imbalance to the input QAM signal with an amplitude imbalance of 10dB and phase imbalance of 45° with the function *iqimbal* from MathWorks seen in 3.2.2 .

$$y = iqimbal(y, 10, 45) \tag{3.2.2}$$

The output of the two function seen above for 4-QAM and 16-QAM for npoint equal to 1024 and I/Q imbalance of 10dB and 45°plotted with *scatterplot* can be seen in Figures 3.2.3a and 3.2.3b.



(a) 4-QAM original and imbalanced data



(b) 16-QAM original and imbalanced data

Figure 3.2.3: Modulation order 4,16-QAM

Then we separate the I and Q symbols in each example, in order to use them as inputs in the algorithm, by extracting the real and imaginary parts of the output from the *iqimbal* function respectively.

3.2.2 Multiply Accumulator

We create the prototype of the algorithm in MATLAB. For the mean operation in steps 1 and 3 in 3.1.3 we design a Multiply Accumulate Unit (macc) function in MATLAB, because it is going to be utilized in Vivado in the design of the FPGA circuit. The structure of a macc unit can be seen in Figure 3.2.4

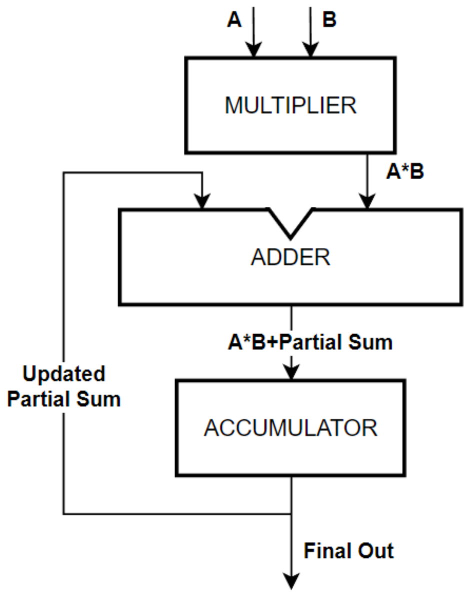


Figure 3.2.4: Structure of Multiply Accumulate Unit

3.2.3 Algorithm Evaluation for 4-QAM & 16-QAM

We executed the steps for the correction algorithm as presented in 3.1.3 and then run the algorithm with processing window size equal to 64, 128, 256 and 1024. The number of samples for each window is equal to 1024, 131072 and 1048576.

The corrected I and Q symbols are compared with the original I and Q data that were generated in the *qammod* in order to verify the accuracy of the algorithm. We utilize the relative error equation which can be seen in 3.2.3.

$$RelativeErrorPercentage = \frac{|measured - real|}{real} \times 100 \tag{3.2.3}$$

The relative error percentage results for the I and Q symbols for **4-QAM** are illustrated in the tables 3.1 3.2 correspondingly.

4-QAM	Window Size			
npoint	64	128	256	1024
1024	14.1903	12.7841	12.8236	12.7749
131072	14.7345	13.4893	13.0347	12.9650
1048576	14.7263	13.4075	13.0083	12.9775

Table 3.1: Relative Error Percentage Between I-Original Data and I-Matlab Results 4-QAM

4-QAM	Window Size			
npoint	64	128	256	1024
1024	37.0323	34.8078	34.5293	34.4696
131072	38.7620	37.6060	37.1421	36.8795
1048576	38.8801	37.6928	37.2487	36.9780

Table 3.2: Relative Error Percentage Between Q-Original Data and Q-Matlab Results 4-QAM

The relative error percentage results for the I and Q symbols for **16-QAM** are illustrated in the tables 3.3 3.4 respectively.

16-QAM	Window Size			
npoint	64	128	256	1024
1024	21.0646	18.5447	17.8823	16.9282
131072	22.4404	19.7023	18.3292	17.3180
1048576	22.3744	19.6561	18.2860	17.3941

Table 3.3: Relative Error Percentage Between I-Original Data and I-Matlab Results 16-QAM

16-QAM	Window Size			
npoint	64	128	256	1024
1024	41.5886	39.9195	39.1859	38.7407
131072	40.7508	38.7685	37.7118	36.8148
1048576	41.0351	38.9571	37.8650	37.0302

Table 3.4: Relative Error Percentage Between Q-Original Data and Q-Matlab Results 16-QAM

Chapter 4

FPGA Circuit Design

4.1 Introduction

Until this chapter we have analyzed and tested how the I/Q imbalance algorithm behaves with the assistance of MATLAB codes. In the current chapter, we are focusing on the design and implementation in VHDL language. In summary, we are presenting the block design of the algorithm and then we delve into each block in particular. Afterwards, the components used to execute the algorithm are shown and explained. Finally, the results from the VHDL code are compared with the original and MATLAB data. The tool employed for this purpose was the Xilinx Vivado Design Suite 2022.1.

4.2 Block Design

First, the top level block design is shown in Figure 4.2.1 and each block is going to by analyzed in more detail later on.
As can be seen in this Figure the input signals of the system are:

- the clock signal(clk)
- the I and Q imbalanced data(I input, Q input)
- the width of the input in bits(SIZEIN)
- the size of the processing window(Window)

Whilst the output signals of the system are:

- the I and Q corrected data

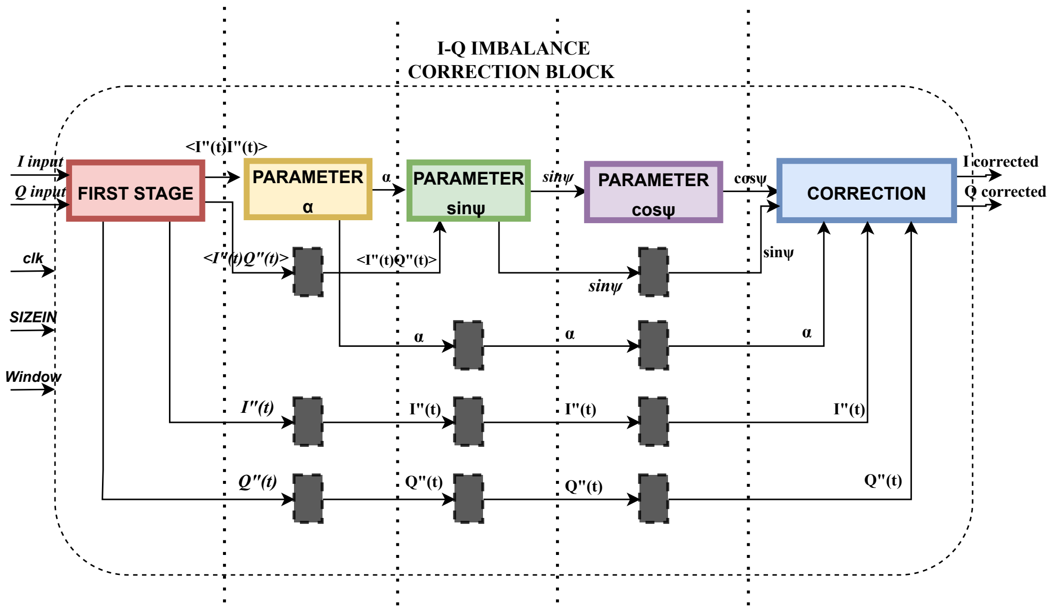


Figure 4.2.1: I/Q Imbalance Correction Top-Level Block Design

4.2.1 First Stage Block

In the beginning, the imbalanced I and Q symbols enter the first stage alongside the window size in order to calculate the mean of the I'(t) and Q'(t). By multiplying each I and Q input with $\frac{1}{N}$ and then adding it to the result of the previous sum we end up with β_I and β_Q. Simultaneously, the input data are stored temporarily in two separate buffers in order to "wait" while the mean calculation for each window takes place. Once β_I and β_Q are calculated for the particular window, they are subtracted from the respective I'(t) and Q'(t) to remove the DC biases. Next, we repeat the mean calculation for the I"(t)I"(t) and I"(t)Q"(t). The structure of the First Stage Block is presented in Figure 4.2.2.

4.2.2 Parameter α block

In this block, we target the calculation of parameter a. As seen in the 2.2 we multiply the mean of I"(t)*I"(t) which is taken from the previous block with two. Concerning the multiplication we use the shift left function of VHDL and shift left the the mean of I"(t)*I"(t) by 1 bit which is essentially equivalent to multiplying the data with 2^1. Then we pass the result in a

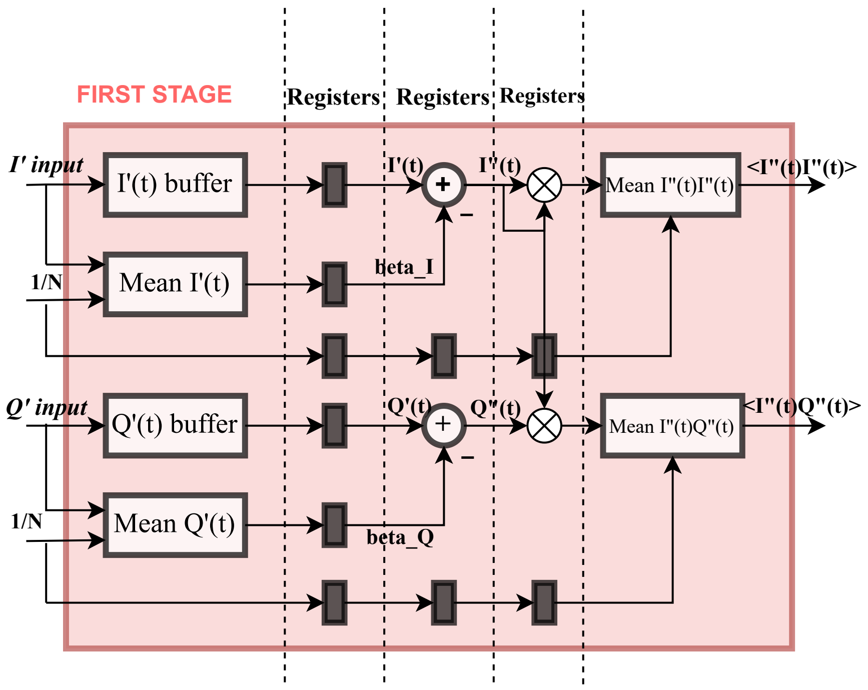


Figure 4.2.2: First Stage Block Design

square root to get the desirable α parameter. The structure of the Parameter a Block is shown in Figure 4.2.3.

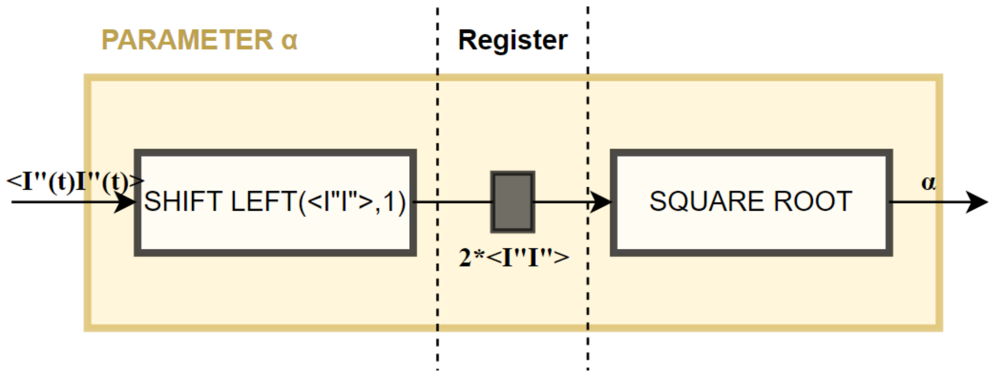


Figure 4.2.3: Parameter α Block Design

4.2.3 Sin(ψ) parameter block

Once parameter a is calculated we use it in conjunction with the mean of I"(t)Q"(t), which was calculated in the First Stage, to find the sin(ψ) parameter. First, we shift left the $< I"(t)Q"(t) >$ by one bit to acquire the 2*$< I"(t)Q"(t) >$ similarly with the previous block. Following this, we divide it with parameter α which results to parameter sin(ψ). The structure of the Parameter sin(ψ) Block is shown in Figure 4.2.4.

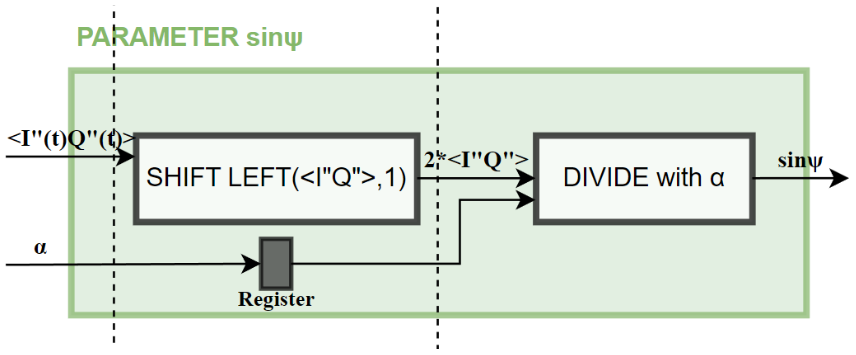


Figure 4.2.4: Parameter $sin\psi$ Block Design

4.2.4 Cos(ψ) parameter block

In order to calculate the cosy(ψ) parameter we multiply the sin(ψ) with itself in order to obtain $sin^2(\psi)$. Then, we acquire $cos^2(\psi)$ by subtracting $sin^2(\psi)$ from 1. Finally, we use square root and obtain cos(ψ). The structure of the Parameter cos(ψ) Block is shown in Figure 4.2.5.

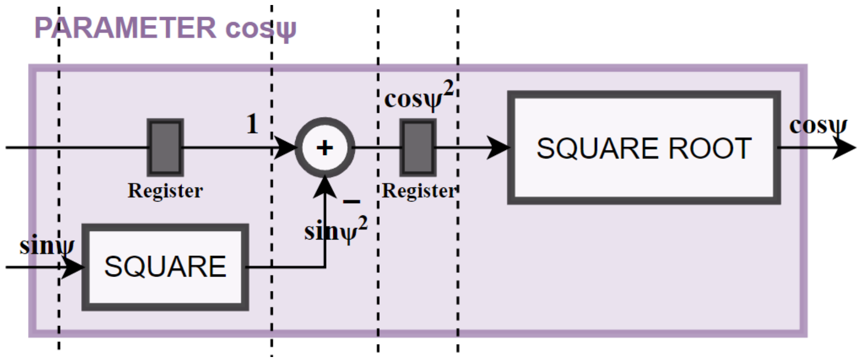


Figure 4.2.5: Parameter $cos\psi$ Block Design

4.2.5 Correction block

In the correction block we assemble the correction matrix parameters A,C and D. The A matrix parameter is calculated by dividing the I"(t) with the α parameter which results immediately to the corrected I data. To calculate the Q corrected data we have to form the C and D matrix parameters. First, the C matrix parameter is obtained by multiplying I"(t) with -sin(ψ) and at the same time multiply α with cos(ψ) and then dividing the first with the second. The D matrix parameter is acquired by dividing Q"(t) with cos(ψ). We add the two parameters C and D and we get the corrected Q data. The structure of the Correction Block is shown in Figure 4.2.6.

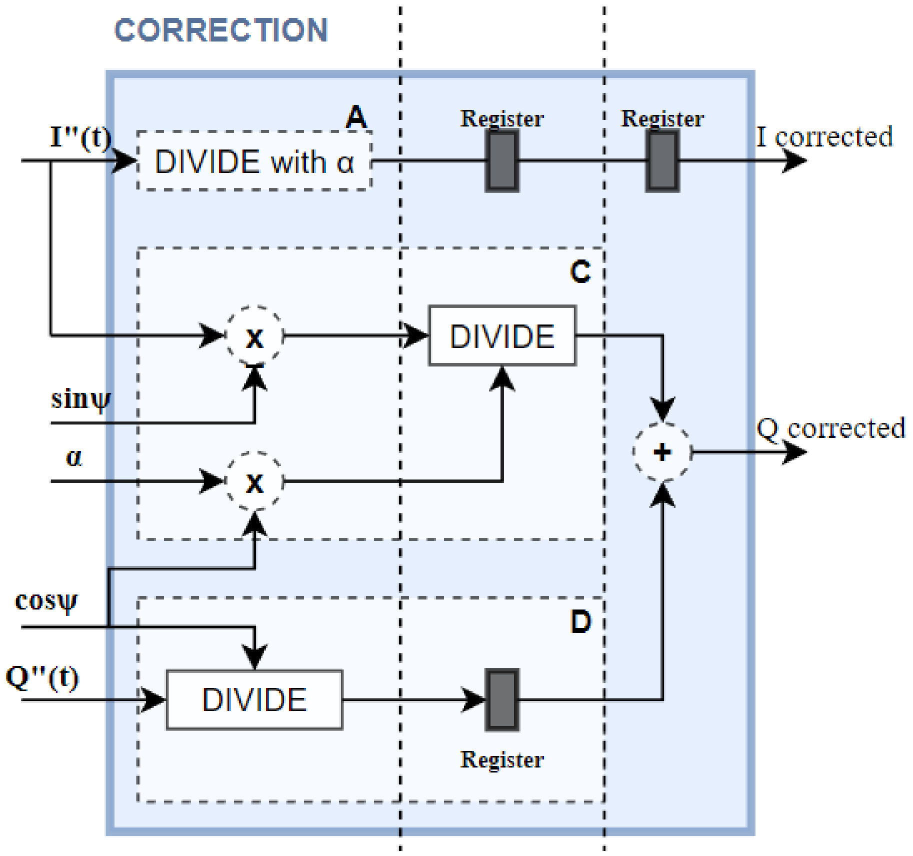


Figure 4.2.6: Correction Block Design

4.3 VHDL components

During the implementation of the I/Q correction algorithm in VHDL we had to employ some Language Templates and Math Functions from IP Catalog. Particularly, we had to execute the mean and buffer operations, the square root and the division process. We will utilize the macc and Block RAM language templates for the former two and the CORDIC and DIVIDER functions from the Math Functions for the latter two. The summary of each of the aforementioned processes are presented below.

4.3.1 Multiply and Accumulate(macc)

During the First Stage block 4.2.2 we have to implement the mean function to calculate β_I and β_Q which are the DC biases, and then subtract them from the initial data. Afterwards, we use the mean function to calculate the $< I"(t)I"(t) >$ and $< I"(t)Q"(t) >$.
We utilize the Multiply and Accumulate(macc) language template to execute the mean operation. The macc operation accepts two operands, a multiplier B and a multiplicand A and produces a product (Mul-Reg=A*B) that is added/subtracted to the previous adder/subtracter result (Adder Out=Mul-Reg ± Mul-Reg).
We supply two different macc units with the I'(t) and Q'(t) data as the multiplicands A and $\frac{1}{Window}$ for the multiplier B in both units. The sum of these fractions result to the mean for each processed window.

$$AccumulateOut = I_1'(t)\frac{1}{Window} + I_2'(t)\frac{1}{Window} + ... + I_N'(t)\frac{1}{Window} \tag{4.3.1}$$

$$= \frac{I_1' + I_2' + ... + I_N'}{Window} =< I'(t) > \tag{4.3.2}$$

Similarly we execute the $< I"(t)I"(t) >$ and $< I"(t)Q"(t) >$.

Vivado Synthesis implements Multiply-Accumulate implementation on DSP block resources. It attempts to maximize circuit performance by leveraging all the pipelining capabilities of DSP blocks.

4.3.2 Block RAM

During numerous operations the data need to be temporarily stored in a buffer in order to be synchronized properly. In this implementation, we employ the Simple Dual Port 1 clock Block RAM from the language templates in Vivado to execute the buffer. We utilize the First-In First-Out logic for the Block RAM, thus during one clock cycle we first read the output data before writing the new input. The larger the processing window the more

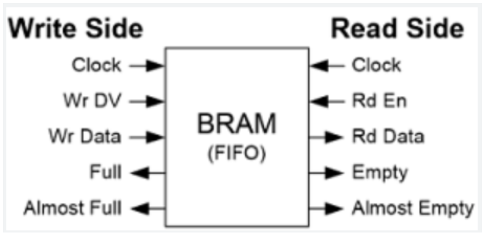


Figure 4.3.1: Block RAM-FIFO

BRAMs are utilized during the synthesis step because more data need to be written in the Block Ram before they are read. In this implementation for window=64 and 128, no BRAMs are utilized. Whilst for processing window=256 and 1024 the BRAMs utilized are 2 and 4 respectively.

4.3.3 CORDIC

The COordinate Rotation DIgital Computer(CORDIC) is a special-purpose digital computer where a unique computing technique is employed which enables to solve trigonometric relationships that occur in plane coordinate rotation and conversion from rectangular to polar coordinates [31]. It was originally developed in [32] to confront trigonometric equations and later in [33] its use was extended to solve a wide variety of equations such as hyperbolic and square root.
The CORDIC core in Vivado implements a generalized CORDIC algorithm that solves the following expressions [34]:

- Rectangular $<->$ Polar Conversion
- Trigonometric
- Hyperbolic
- Square Root

Initially the CORDIC algorithm was developed to execute a vector rotation, where the vector (X,Y) is rotated through the angle θ creating a new vector (X',Y').

$$X' = (cos(\theta) \times X - sin(\theta) \times Y)$$

$$Y' = (cos(\theta) \times Y - sin(\theta) \times X)$$

The following expressions represent the i^{th} iteration of vector rotation:

$$x_{i+1} = x_i - a_i y_i 2^{-i}$$

$$y_{i+1} = y_i - a_i x_i 2^{-i}$$

$$\theta_{i+1} = \theta_i - a_i tanh^{-1}(2^{-i})$$

where $a_i = \pm\, 1$ and it represents the direction of the rotation

We utilize the CORDIC function to apply the square root to the necessary stages of the I/Q correction algorithm.

To obtain the square root function we have to set the width of the data and the data to be square rooted. If the input X-IN is set to unsigned fraction, then X-IN and X-OUT is restricted to the range:

$$0 <= X_{IN}/X_{OUT} < 2$$

Whereas if the data format is set to Unsigned Integer, X-IN is limited to the range:

$$0 <= X_{IN} < 2^{InputWidth}$$

and the X-OUT to:

$$0 <= X_{OUT} < 2^{int(InputWidth/2)+1}$$

4.3.4 DIVIDER

Division is considered the most complex of the four basic arithmetic operations. The division operand "/" is not supported in Vivado Synthesis if the divisor is not a power of 2 or both dividend and divisor are not constant. Thus we need to use a different way to execute the divisions during the last step of the I-Q imbalance correction algorithm 2.2. The LogiCORE IP Divider Generator is an excellent alternative for the division operation since it is resource efficient and has a high performance for integer division. The Divider Generator supports three different implementations of division to allow trade-offs between throughput, latency and resource use:

1. LUTMult: utilizes a simple lookup estimate of the reciprocal of the divisor followed by a multiplication by the dividend. This implementation uses DSP slices, block RAM and a small amount of FPGA logic primitives(registers and LUTs).

2. Radix-2: Non-restoring algorithm solves one bit of the quotient per cycle using addition and subtraction. The Radix2 solution uses registers and not any DSP or block RAM primitives.

3. High Radix: Division with prescaling, which is suitable for operand widths grater than 16 bits due to the overhead caused by the prescaling. This implementation uses DSPs and block RAMs.

In our algorithm we have utilized the Radix-2 implementation because it has the ability to control the degree of parallelism used in the algorithm, hence we can make trade-offs between performance and resources.

4.4 Design Verification

Design Verification is a vital step in the development process. It aims to confirm that the system design complies with the system requirements and specifications. The verification workflow we followed to verify that the theoretical simulations on MATLAB and Behavioral Simulation on Vivado are what we anticipated is:
We generated the 4-QAM and 16-QAM signals in MATLAB. Then, we applied IQ-imbalance to these signals and converted them to data files. Afterwards, these data files are read by a VHDL testbench through an input file and then the simulation output is written to an output text file. At last, we read the output .txt file in MATLAB and produce the Relative Error Percentage between the VHDL results and the original data and MATLAB results.
The relative error percentage results between the **Original** and the **VHDL** results for the I and Q symbols for **4-QAM** are shown below in the tables 4.1 and 4.2 respectively:

4-QAM	Window Size			
npoint	64	128	256	1024
1024	14.1893	12.7839	12.8234	12.7749
131072	14.7343	13.4889	13.0339	12.9647
1048576	14.7261	13.4071	13.0079	12.9773

Table 4.1: Relative Error Percentage Between I-Original Data and I-VHDL Results 4-QAM

4-QAM	Window Size			
npoint	64	128	256	1024
1024	37.0367	34.8073	34.5408	34.4803
131072	38.7687	37.6134	37.1501	36.8886
1048576	38.9443	37.7006	37.2573	36.9872

Table 4.2: Relative Error Percentage Between Q-Original Data and Q-VHDL Results 4-QAM

The relative error percentage results between the **Original** and the **VHDL** results for the I and Q symbols for **16-QAM** are shown below in the tables 4.3 and 4.4 respectively:

16-QAM	Window Size			
npoint	64	128	256	1024
1024	21.0657	18.5471	17.8829	16.9261
131072	22.4395	19.7012	18.3275	17.3181
1048576	22.3737	19.6553	18.2851	17.3940

Table 4.3: Relative Error Percentage Between I-Original Data and I-VHDL Results 16-QAM

16-QAM	Window Size			
npoint	64	128	256	1024
1024	41.5913	39.9255	39.2021	38.7646
131072	40.7499	38.7683	37.7141	36.8248
1048576	41.3597	38.9572	37.8665	37.0371

Table 4.4: Relative Error Percentage Between Q-Original Data and Q-VHDL Results 16-QAM

While, the relative error percentage results between the **MATLAB** and the **VHDL** results for the I and Q symbols for **4-QAM** are shown below in the tables 4.5 and 4.6 respectively:

4-QAM	Window Size			
npoint	64	128	256	1024
1024	0.0058	0.0215	0.0426	0.1689
131072	0.0055	0.0217	0.0431	0.1714
1048576	0.0055	0.0217	0.0431	0.1714

Table 4.5: Relative Error Percentage Between I-MATLAB Data and I-VHDL Results 4-QAM

4-QAM	Window Size			
npoint	64	128	256	1024
1024	0.0562	0.1291	0.2568	1.0261
131072	0.0548	0.1244	0.2449	0.9666
1048576	0.0896	0.1243	0.2443	0.9652

Table 4.6: Relative Error Percentage Between Q-MATLAB Data and Q-VHDL Results 4-QAM

The relative error percentage results between the **MATLAB** and the **VHDL** results for the I and Q symbols for **16-QAM** are shown below in the tables 4.7 and 4.8 respectively:

16-QAM	Window Size			
npoint	64	128	256	1024
1024	0.0173	0.0313	0.0606	0.2679
131072	0.0205	0.0350	0.0608	0.2707
1048576	0.0225	0.0341	0.0609	0.2718

Table 4.7: Relative Error Percentage Between I-MATLAB Data and I-VHDL Results 16-QAM

16-QAM	Window Size			
npoint	64	128	256	1024
1024	0.1019	0.1946	0.3806	1.4510
131072	0.2437	0.3252	0.5709	1.5526
1048576	0.7410	0.3814	0.4515	1.5479

Table 4.8: Relative Error Percentage Between Q-MATLAB Data and Q-VHDL Results 16-QAM

The constellation diagrams between the Original data, the Imbalanced data and the MATLAB-VHDL results for window=1024, number of samples=1048576 and 4-QAM and 16-QAM are illustrated below in Figures 4.4.1 4.4.2:

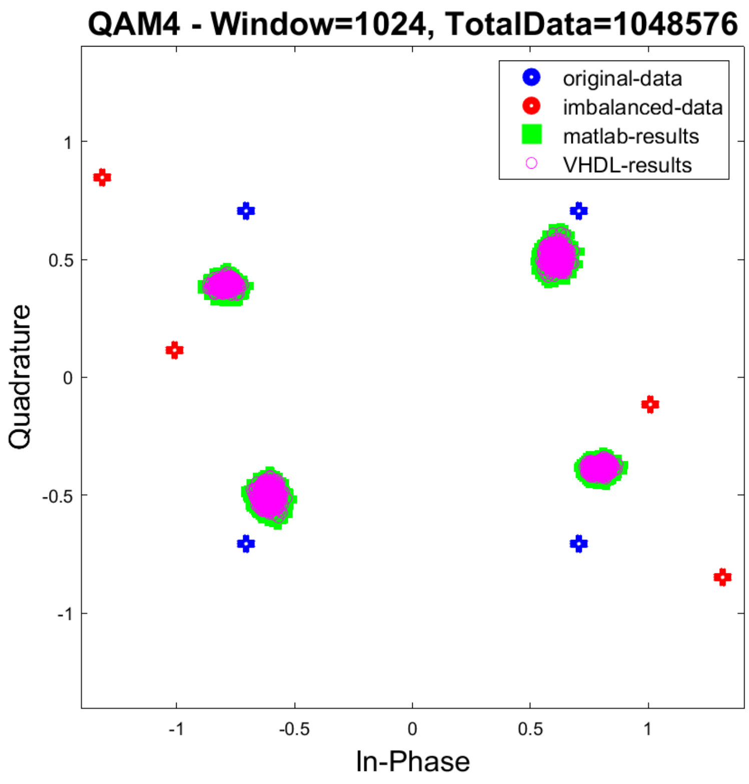


Figure 4.4.1: 4-QAM original,imbalanced data and MATLAB,FPGA results

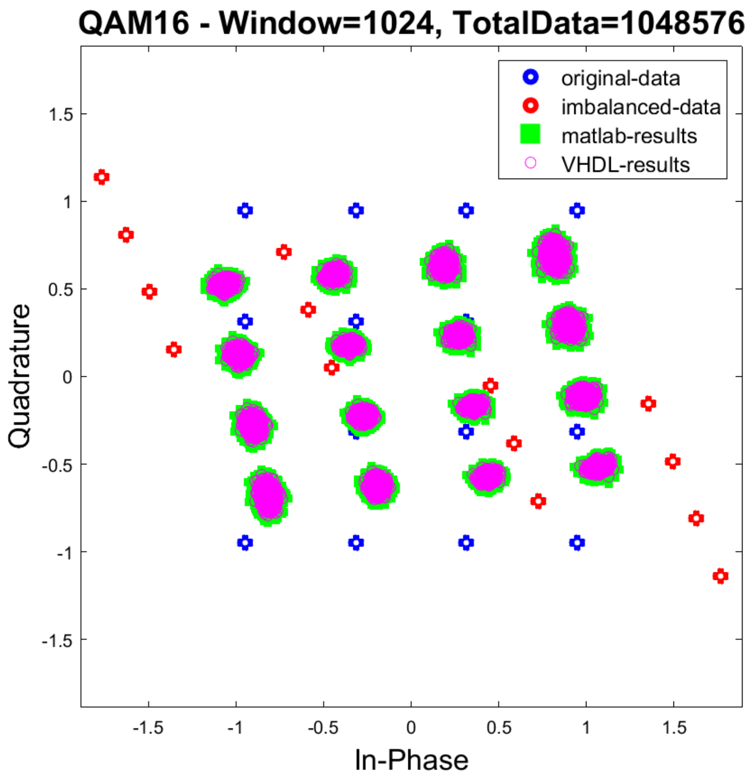


Figure 4.4.2: 16-QAM original,imbalanced data and MATLAB,FPGA results

The Utilization of the Hardware Resources and the Maximum operating frequency change based on the processing window of the input data. The Table 4.9 illustrates the utilization of the hardware resources and the max operating frequency for processing windows equal to 64, 128, 256 and 1024 respectively. We set the period as a constraint equal to 3ns. Once the implementation of the design in Vivado was finished, the worst negative slack was subtracted from that period and the maximum operating frequency for each window could be calculated (f=1/T).

Table 4.9: Operating frequency and resource utilization of the system for the four different processing windows.

Window		**Resources**					**Max. Operating Freq.**
		LUTs	LUTRAMs	DFFs	BRAMs	DSPs	
	Available	230400	101760	460800	312	1728	
64	Used	10063	260	26446	0	7	342MHz
	Utilization	4.37%	0.26%	5.74%	0%	0.41%	
128	Used	10133	312	26511	0	7	339MHz
	Utilization	4.40%	0.31%	5.75%	0%	0.41%	
256	Used	9876	104	26377	2	7	346MHz
	Utilization	4.29%	0.10%	5.72%	0.64%	0.41%	
1024	Used	9878	104	26381	4	7	340MHz
	Utilization	4.29%	0.10%	5.73%	1.28%	0.41%	

Chapter 5

RF Modulation Recognition with Vitis-AI

In the current chapter we present the tools used to implement the RF modulation recognition on a FPGA, more specifically the Vitis-AI platform, the dataset which was utilized for the training and testing of the neural network and finally the structure and components of the AI model.

5.1 Vitis-AI

Xilinx Vitis AI is a development environment which enables acceleration of AI inference on the Xilinx hardware platforms. It can be leveraged by users to conduct deep-learning-related research and development with the utilization of its optimized IP cores, tools, libraries, models and example designs. The purpose of Vitis AI is to provide high-efficiency and ease-of-use in order to exploit the full potential of acceleration on Xilinx FPGAs [35] [36] [37] [38]. As seen in Figure 5.1.1, the Vitis AI environment includes the following main components:

- AI Model Zoo: Consists of a variety of pre-optimized deep learning models to facilitate the inference on Xilinx Platforms.
- AI Optimizer: An optional model compression technology, that can prune a model and decrease the model complexity by 5-50 times with negligent accuracy degradation.
- AI Quantizer: An efficient quantizer that allow model quantization,calibration and fine-tuning.

- AI Compiler: Converts the quantized neural network model to an efficient DPU instruction set.
- AI Profiler: Implements an analysis of its efficiency and the allocation of the computing resources.
- AI Library: It is a set of high-level libraries and APIs that can be used for AI applications from the edge to the cloud.
- DPU: Pre-implemented on the hardware IP cores that can be parameterized to accelerate a plethora of widely adopted applications.

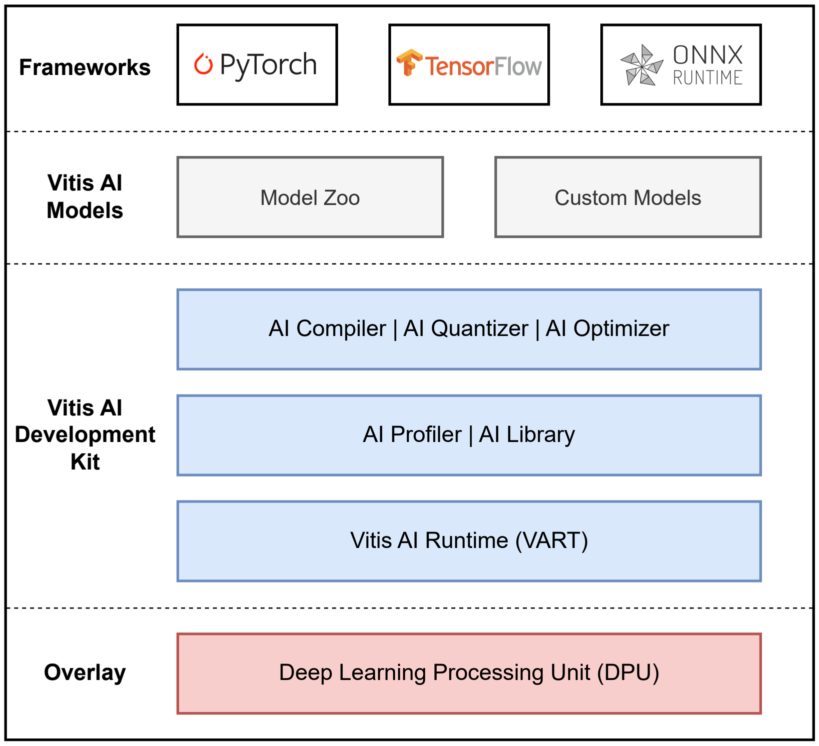


Figure 5.1.1: Vitis AI Structure

In this implementation we are focused on the following tools: AI Quantizer, AI Compiler and the DPU.

5.1.1 Vitis-AI Quantizer

Usually, neural networks are composed of single-precision floating-point weights and activations. The AI Quantizer converts the 32-bit floating-point weights and activations to 8-bit fixed-point integers. Thus, it decreases the computing complexity with insignificant loss of accuracy. In addition, the new fixed-point model demands less memory bandwidth and compared to the floating-point model has higher speeds and more power efficiency. [35] [37]

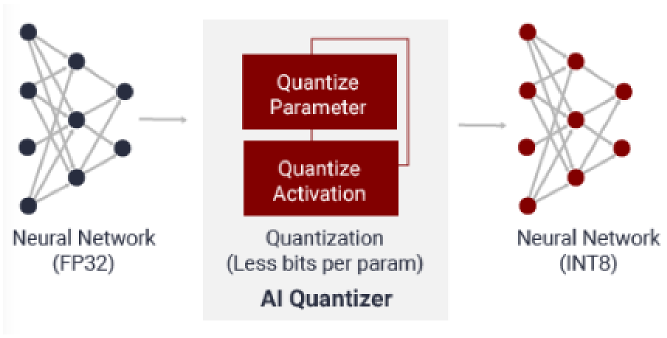


Figure 5.1.2: Vitis AI Quantizer

5.1.2 Vitis-AI Compiler

Once the model is quantized, the AI Compiler executes sophisticated optimizations and creates a .xmodel file. This file includes a highly efficient DPU instruction set which is compiled for a specific FPGA board using a configuration file, arch.json. The optimizations are layer fusion and instruction scheduling that uses parallel processing, or data reuse. [35] [37]

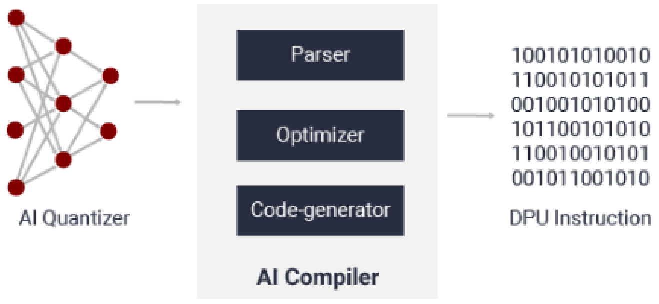


Figure 5.1.3: Vitis AI Compiler

5.1.3 Deep Learning Processing Unit (DPU)

The hardware core of the development environment is named Deep Learning Processing Unit (DPU). It is a programmable engine optimized for deep neural networks. It consists of pre-implemented on the hardware parameterizable IP cores that do not require place and routing. The DPU is used in order to accelerate the computing workloads of deep learning inference algorithms that are commonly seen in various computer vision applications. [37]

There are a variety of different DPUs available for different tasks and Xilinx Platforms. In Figure 5.1.4 is illustrated the DPU naming convention which is useful to comprehend the features, characteristics and target hardware platforms from a given DPU name.

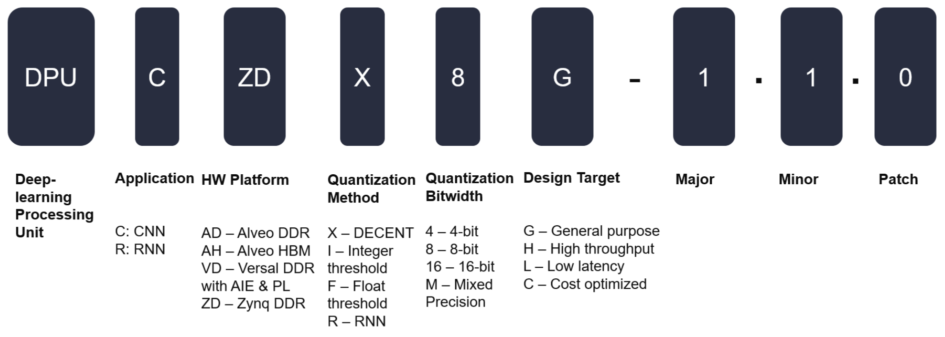


Figure 5.1.4: DPU naming convention

In this application we are going to use DPUCZDX8G DPU which is designed for the Zynq Ultrascale+ MPSoC and optimized for convolutional neural networks. The DPUCZDX8G IP is implemented in the Programmable Logic (PL) of the selected device with direct connections to the Processing System (PS).

In Figure 5.1.5 the Top-Level Block Diagram of this DPU is presented. In summary, it executes the microcode which is generated from a compiled neural-network graph and then demands accessible locations for input images and temporary output data. In addition, it requires a program running on the Application Processing Unit (APU) in order to service the interrupts and arrange data transfers.

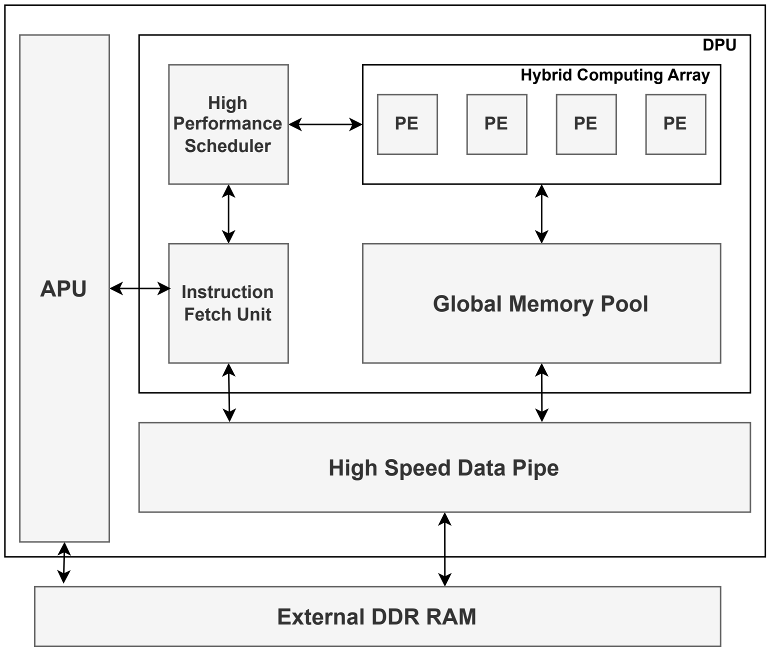


Figure 5.1.5: DPU Top-Level Diagram

In Figure 5.1.6 the hardware architecture of DPUCZDX8G is shown. Following the start-up, in order to control the computing engine, the DPU fetches instructions,that are generated by the Vitis AI, from the off-chip memory. The on-chip memory buffers input activations, intermediate feature-maps and output meta-data so it can reach high throughput and efficiency. The DPU reduces external memory bandwidth by reusing the data. For the computing engine a deep pipelined design is utilized and the Processing Elements (PE) exploit the multipliers, adders and accumulators in the target Xilinx Devices. [39]

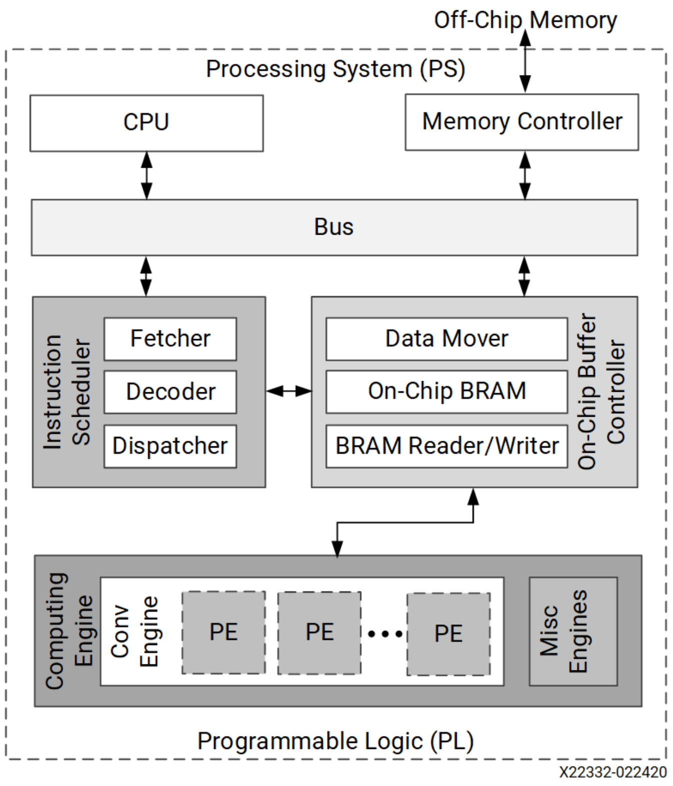


Figure 5.1.6: DPU Hardware

5.2 Dataset

In this section, we are going to analyze the dataset used in the training and testing of the neural network utilized for the RF modulation recognition. Based on [40], 24 different and improved analog and digital modulators are used. In particular, we are employing the dataset generated in [41].

Two different propagation scenarios are considered, the first is generating wireless channels generated from the model shown in Figure 5.2.1 , while the second is over-the-air transmission channel of clean signal as presents in Figure 5.2.2 with no synthetic channel impairments. In the former, for each example a random value for each of the variables shown below in 5.1 are drew in order to generate new and uncorrelated random channel initialization for each example.

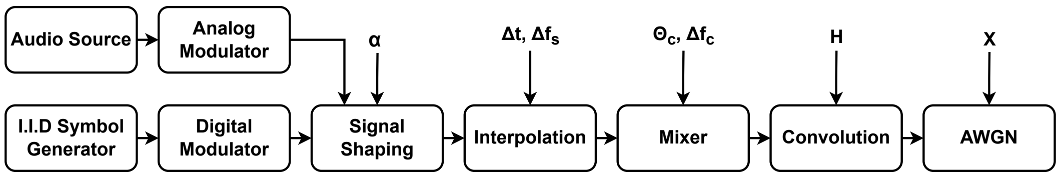


Figure 5.2.1: System for dataset signal generation and synthetic channel impairment modeling

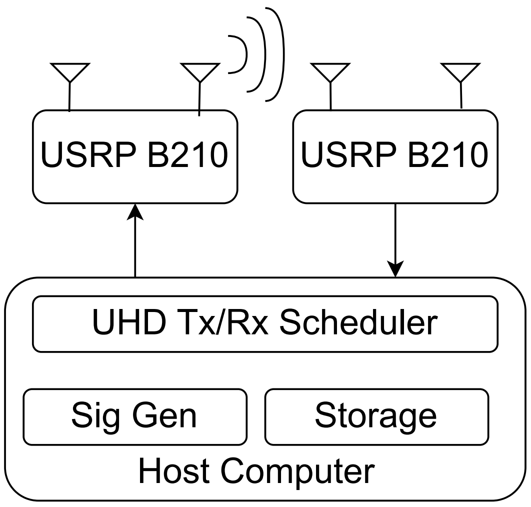


Figure 5.2.2: Over-the-air test configuration

In the 24 different modulations, various high order modulations are included that are used in real world in very high SNR with low-fading channel environments and also to the synthetic part of the dataset impairments are applied . The SNRs for each example range from -20dB to +30dB. The exact modulations included in the dataset are:
OOK, 4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 32PSK, 16APSK, 32APSK, 64APSK, 128APSK, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM, AM-SSB-WC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM, GMSK, OQPSK. [41]

Random Variable	Distribution
α	U(0.1, 0.4)
δ_τ	U(0, 16)
δf_s	N(0, σ_{clk}
θ_c	U(0, 2π)
δf_c	N(0, σ_{clk}
H	$\Sigma_i \delta(t - Rayleigh_i(\tau))$

Table 5.1: Random Variable Initialization

5.3 RF Modulation Recognition

For the classification of the aforementioned modulations we are using convolutional neural networks (CNNs). It is worth mentioning that the features of the CNN used in this application are the raw I-Q samples of each radio signal example. No feature extraction or any pre-processing takes place on the raw samples, instead the network ”learns” the raw time-series features directly. [41]

5.3.1 Residual Neural Network

In this application, we execute the modulation recognition with deep residual networks. A residual neural network (ResNet) is an artificial neural network (ANN) which contains skip connections/ shortcuts that allow to jump over some layers as seen in Figure 5.3.1. They are usually implemented with double or triple layer skips and in between there are nonlinearities, such as ReLU, and batch normalization [42].

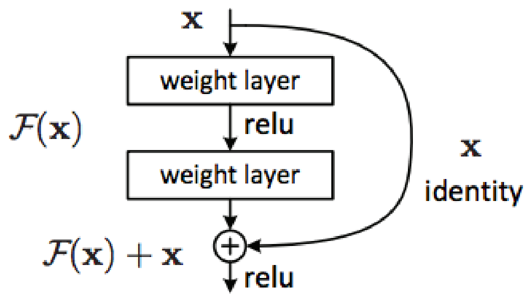


Figure 5.3.1: Residual Training

There are two reasons to add shortcuts in a neural network. Firstly, to avoid the vanishing gradient problem, which occurs in neural networks with gradient-based learning methods and backpropagation. In these methods, in every training iteration the weights are updated based on the partial derivative of the error function of the current weight. In some cases, the gradient will be extremely small and consequently, preventing the weight to change or even stopping the training of the network. [43]. Secondly, skip connections facilitate the training of deeper models since it minimizes the accuracy saturation problem. [42]

5.3.2 Network Layout

The layout of each resnet stack is presented below in Figure 5.3.2. The model contains 4 resnet stacks. Each convolutional layer includes 32 filters and all the activation functions are Rectified Linear Units (ReLUs). [44]

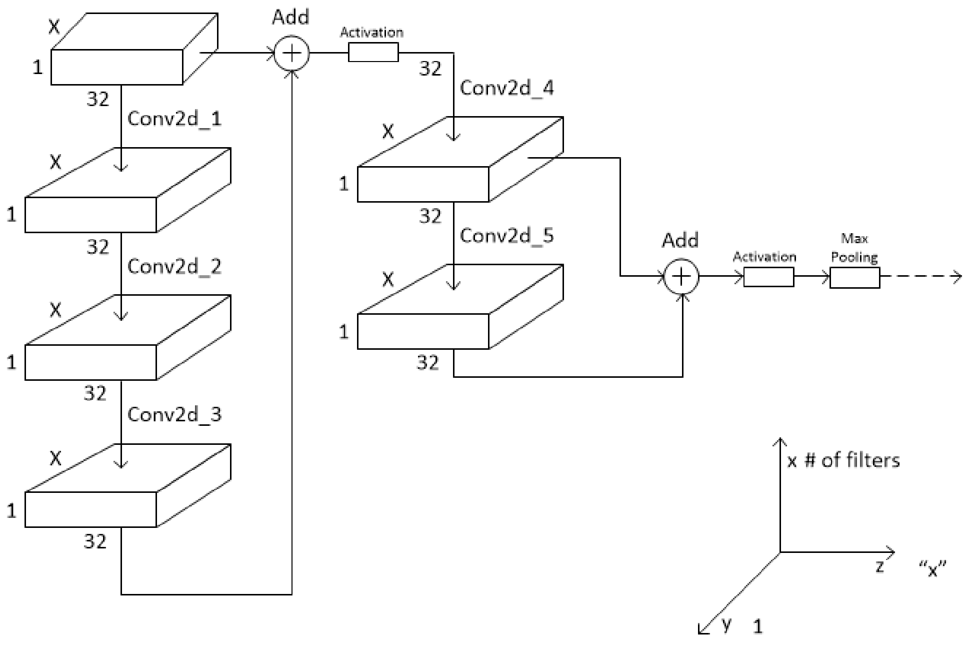


Figure 5.3.2: ResNet Stack

Chapter 6

Experimental Results

In this section, the RF modulation recognition model is trained and tested. The dataset mentioned in 5.2 is split into three different versions. The Original with all the modulations mentioned, the QAM version which includes only the five QAM modulation and the Digital version which does not contain any of the analogue modulation. Then its accuracy results are compared with three different variations of the dataset for each version.
The first is the original dataset with no processing. In the second variation of the dataset, I-Q imbalance is manually added to the 16-QAM example modulations of the dataset using Matlab. More specifically, 10dB amplitude imbalance and 45°phase imbalance is applied to the examples of the 16-QAM for SNR equal to [-20, -10, -2, 0, 2, 10, 20] dB. The third variation of the dataset for training and testing, is the corrected dataset. In more details, the imbalanced dataset is passed through the I-Q imbalance correction algorithm and the corrected results assembled the third variation of the dataset.

6.1 Vitis-AI Results

The model mentioned in 5.3 is trained and tested in the Vitis-AI environment with a jupyter notebook server. The accuracy results once the model it has been quantized (INT8) for each SNR and for each version and variation of the dataset is shown below.

For the **Whole** Dataset the results with no processing, with imbalance and the corrected ones are shown in Figure 6.1.1.

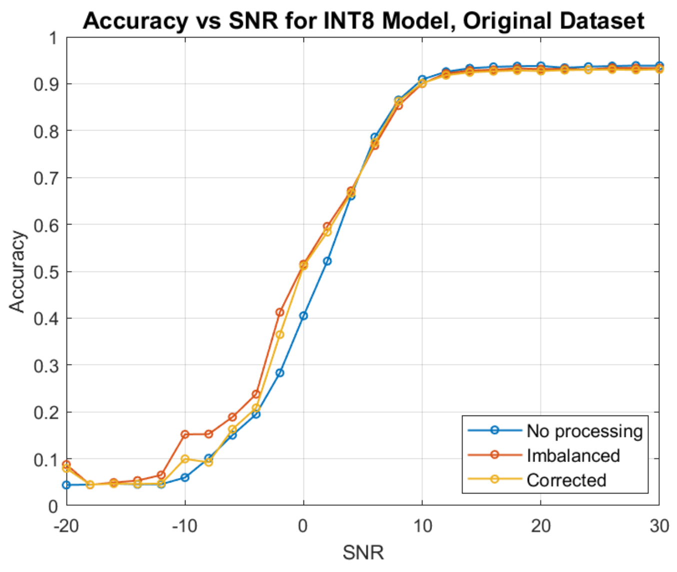


Figure 6.1.1: Accuracy vs SNRs for the Original Dataset

For the **QAM** Dataset the results with no processing, with imbalance and the corrected ones are shown in Figure 6.1.2.

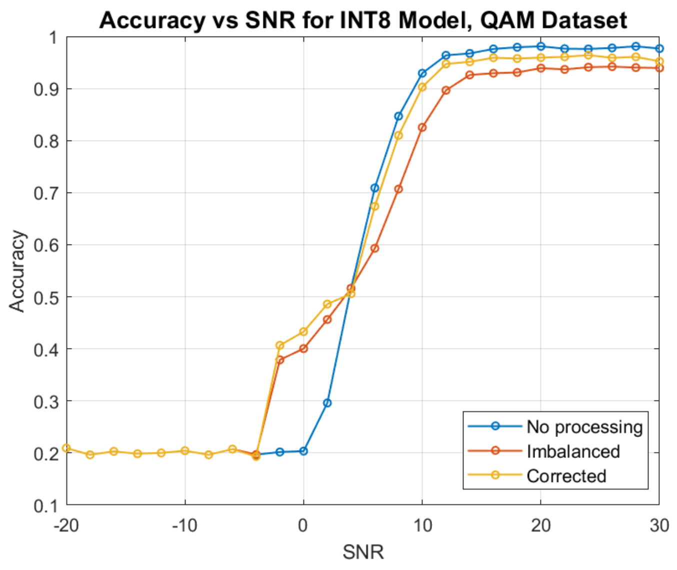


Figure 6.1.2: Accuracy vs SNRs for the QAM Dataset

For the **Digital** Dataset the results with no processing, with imbalance and the corrected ones are shown in Figure 6.1.3.

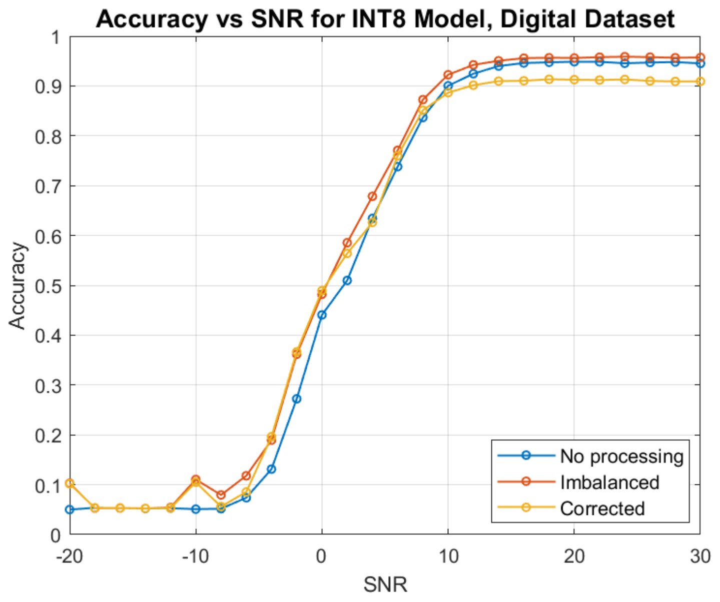
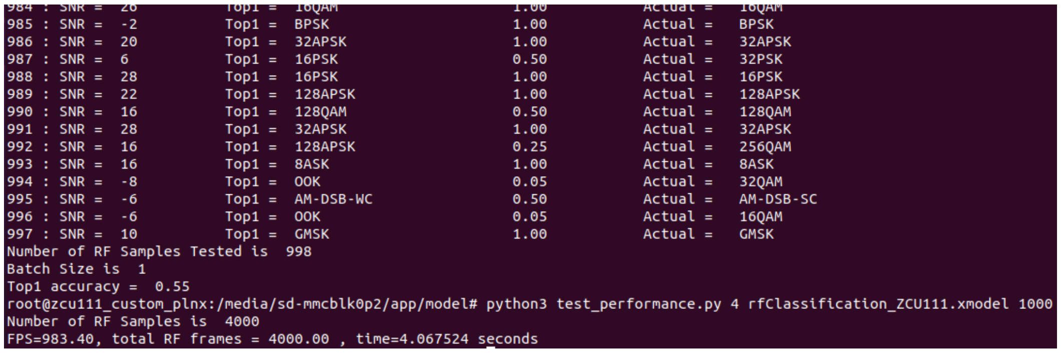


Figure 6.1.3: Accuracy vs SNRs for the Digital Dataset

6.2 Hardware Results

Once the model is compiled and the .xmodel file is created, the accuracy and performance tests for the Original, QAM and Digital Dataset with no processing are run on the Zynq Ultrascale+ RFSoC ZCU111 board. In order to acquire these results, we utilize two different python scripts which have an output as seen in Figure 6.2.1. The rest accuracy results are obtained online in the Vitis-AI environment.
Similarly, we get the results for the rest variations and versions of the dataset which can be seen in Table 6.1.



```
984 : SNR =  26          Top1 =  16QAM                  1.00          Actual =   16QAM
985 : SNR =  -2          Top1 =  BPSK                   1.00          Actual =   BPSK
986 : SNR =  20          Top1 =  32APSK                 1.00          Actual =   32APSK
987 : SNR =  6           Top1 =  16PSK                  0.50          Actual =   32PSK
988 : SNR =  28          Top1 =  16PSK                  1.00          Actual =   16PSK
989 : SNR =  22          Top1 =  128APSK                1.00          Actual =   128APSK
990 : SNR =  16          Top1 =  128QAM                 0.50          Actual =   128QAM
991 : SNR =  28          Top1 =  32APSK                 1.00          Actual =   32APSK
992 : SNR =  16          Top1 =  128APSK                0.25          Actual =   256QAM
993 : SNR =  16          Top1 =  8ASK                   1.00          Actual =   8ASK
994 : SNR =  -8          Top1 =  OOK                    0.05          Actual =   32QAM
995 : SNR =  -6          Top1 =  AM-DSB-WC              0.50          Actual =   AM-DSB-SC
996 : SNR =  -6          Top1 =  OOK                    0.05          Actual =   16QAM
997 : SNR =  10          Top1 =  GMSK                   1.00          Actual =   GMSK
Number of RF Samples Tested is  998
Batch Size is  1
Top1 accuracy =  0.55
root@zcu111_custom_plnx:/media/sd-mmcblk0p2/app/model# python3 test_performance.py 4 rfClassification_ZCU111.xmodel 1000
Number of RF Samples is  4000
FPS=983.40, total RF frames = 4000.00 , time=4.067524 seconds
```

Figure 6.2.1: Accuracy and Performance results for the Original Dataset with no processing on the ZCU111 board

Table 6.1: Accuracy and Performance results on the Zynq Ultrascale+ RF-SoC ZCU111 board

		Accuracy	**Performance**
Original	No processing	0.55	FPS=983.40, time=4.067524sec
	Imbalanced	0.57	-
	Corrected	0.56	-
QAM	No processing	0.58	FPS=981.11, time=4.077023sec
	Imbalanced	0.57	-
	Corrected	0.59	-
Digital	No processing	0.55	FPS=982.24, time=4.072324sec
	Imbalanced	0.57	-
	Corrected	0.55	-

Chapter 7

Conclusions and Future Work

In the current thesis, the definition of I-Q Imbalances and their correction algorithm were presented, the algorithm was assessed through MATLAB and finally implemented on an FPGA platform through Vivado Design Suite. Afterwards, there was a comparison of the accuracy of an RF Modulation Recognition Neural Network when I-Q imbalances were applied to its dataset and when it was corrected through the correction algorithm.

To evaluate the correction algorithm we utilized the relative error between the original generated 4,16-QAM symbols and the corrected ones through MATLAB and Vivado. As seen in the results shown in Chapter 3 and Chapter 4 the relative error percentage for the I symbols between the original data, the MATLAB and the VHDL results ranged from 12% to 14% for the 4-QAM data and from 16% to 21% for the 16-QAM. Whilst for the Q symbols for 4-QAM it ranged from 34% to 37% and for 16-QAM from 36% to 41%. It is also worth noting that, the results from the algorithms implemented in MATLAB and Vivado have negligent difference, since their relative error percentage for the I-symbols have maximum of 0.17% for the 4-QAM and 0.27% for the 16-QAM, whereas for the Q-symbols the maximum for the 4-QAM is 1% and for the 16-QAM 1.5%. Consequently, there was significant compensation of the imbalanced data back to their original state.

Furthermore, the utilization of the resources, post-implementation, was minimum since on average only 4.3% of the LUTs were used and 5.7% of the DFFs. BRAMs were used only for the processing windows equal to 256 and 1024 with 0.64% and 1.28% utilization respectively and finally only 0.41%

of DSPs were utilized. The operating frequency for the four processing windows ranged from 428MHz to 500MHz.

As far as the accuracy results of the RF modulation recognition model for the three different versions of the dataset (whole, only QAMs and only Digital Modulations) when each one has three different variations (no processing, imbalanced 16-QAM data and corrected 16-QAM data) the neural network shows negligent differences for each one. As can be seen in Chapter 6, the neural network is able to handle imbalances in its dataset with not significant loss of accuracy. It can also be observed that in the values of the SNR that IQ-imbalance was applied and then corrected, the neural network had improved accuracy compared to when no processing took place in its dataset. This can be explained, on the grounds that the model ”missclasified” the modulations which in reality resulted in predicting them better than before. Overall, we conclude that the particular neural network used for RF modulaltion recognition is trustworthy to classify modulations that have impairments and imbalances which usually occur in real-life situations. Furthermore, we could assume that if the quantizer provided within the Vitis-AI environment converted the 32-bit floating point weights and activations to 16 bit, instead of 8-bit, fixed point integers, the accuracy could increase since there would be more information included in each weight and activation.

For future projects based on this diploma thesis, the neural network could be extended into recognizing more components required in the receiver chain. For instance, recognizing the configuration used in the channel encoder to minimize the unnecessary information send from the transmitter to free space on the bandwidth during packet transmission. In addition, create a neural network that could combine more than one stages of the receiver, such as a joint modulation recognition and demodulation using deep learning. Another compelling future work would be, if the whole correcting and modulation recognizing stages could be integrated into one in the FPGA, without the external processing of the dataset. On top of that, to collect real-time signals to feed the correction algorithm and sequentially the neural network, using ADCs and DACs on the FPGA platform.

Bibliography

[1] U. Madhow, *Fundamentals of digital communication.* Cambridge university press, 2008.

[2] K. Prasad, *Principles of Digital Communication System & Computer Network.* Dreamtech Press, 2003.

[3] E. A. Lee and D. G. Messerschmitt, *Digital communication.* Springer Science & Business Media, 2012.

[4] I. Glover and P. M. Grant, *Digital communications.* Pearson Education, 2010.

[5] B. Jdid, K. Hassan, I. Dayoub, W. H. Lim, and M. Mokayef, "Machine learning based automatic modulation recognition for wireless communications: A comprehensive survey," *IEEE Access*, vol. 9, pp. 57851–57873, 2021.

[6] D. Marquez-Viloria, L. Castano-Londono, and N. Guerrero-Gonzalez, "A modified knn algorithm for high-performance computing on fpga of real-time m-qam demodulators," *Electronics*, vol. 10, no. 5, p. 627, 2021.

[7] I. Ahmed, W. Xu, R. Annavajjala, and W.-S. Yoo, "Joint demodulation and decoding with multi-label classification using deep neural networks," in *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 365–370, IEEE, 2021.

[8] N. E. West and T. O'shea, "Deep architectures for modulation recognition," in *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–6, IEEE, 2017.

[9] D. K. Sharma, A. Mishra, and R. Saxena, "Analog & digital modulation techniques: an overview," *International Journal of Computing Science and Communication Technologies*, vol. 3, no. 1, p. 2007, 2010.

[10] L. Frenzel, "Understanding modern digital modulation techniques," *Electron. Des. Technol. Commun*, 2012.

[11] A. Tarighat, R. Bagheri, and A. H. Sayed, "Compensation schemes and performance analysis of iq imbalances in ofdm receivers," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 3257–3268, 2005.

[12] T. Yoshida, D. Nojima, Y. Nagao, M. Kurosaki, and H. Ochi, "Fpga implementation of joint cfo and iq-imbalance compensator for narrow-band wireless system," in *The 2011 International Conference on Advanced Technologies for Communications (ATC 2011)*, pp. 327–332, IEEE, 2011.

[13] R. Mueller, J. Teubner, and G. Alonso, "Data processing on fpgas," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 910–921, 2009.

[14] H. Amano, *Principles and structures of FPGAs.* Springer, 2018.

[15] T. P. Morgan, "How microsoft is using fpgas to speed up bing search," 2014.

[16] D. Horn, "Six reasons you should consider fpgas over asics or cpu/gpus." `https://digilent.com/blog/six-reasons-you-should-consider-fpgas-over-asics-or-cpu-gpus/`, 2020. Accessed: 2023-01-21.

[17] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[18] Z.-H. Zhou, *Machine learning.* Springer Nature, 2021.

[19] I. El Naqa and M. J. Murphy, "What is machine learning?," in *machine learning in radiation oncology*, pp. 3–11, Springer, 2015.

[20] M. W. Berry, A. Mohamed, and B. W. Yap, *Supervised and unsupervised learning for data science.* Springer, 2019.

[21] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[22] T. Hofmann, "Unsupervised learning by probabilistic latent semantic analysis," *Machine learning*, vol. 42, no. 1, pp. 177–196, 2001.

[23] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Machine learning proceedings 1995*, pp. 194–202, Elsevier, 1995.

[24] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[25] S.-C. Wang and S.-C. Wang, "Artificial neural network," *Interdisciplinary computing in java programming*, pp. 81–100, 2003.

[26] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, pp. 1–6, Ieee, 2017.

[27] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[28] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[29] J. Tubbax, B. Come, L. Van der Perre, S. Donnay, M. Engels, H. De Man, and M. Moonen, "Compensation of iq imbalance and phase noise in ofdm systems," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 872–877, 2005.

[30] S. ELLINGSON, "Correcting iq imbalance in direct conversion receivers/virginia tech."

[31] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959.

[32] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[33] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference*, pp. 379–385, 1971.

[34] Xilinx, "CORDIC." `https://www.xilinx.com/products/intellectual-property/cordic.html`. Accessed: 2023-01-31.

[35] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, "Convolutional neural network implementations using vitis ai," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0365–0371, 2022.

[36] J. Wang and S. Gu, "Fpga implementation of object detection accelerator based on vitis-ai," in *2021 11th International Conference on Information Science and Technology (ICIST)*, pp. 571–577, IEEE, 2021.

[37] Xilinx, "VITIS AI User Guide." `https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html`. Accessed: 2023-02-09.

[38] Xilinx, "Vitis-AI." `https://github.com/Xilinx/Vitis-AI`. Accessed: 2023-02-09.

[39] Xilinx, "DPUCZDX8G for Zynq UltraScale+ MPSoCs." `https://www.xilinx.com/products/intellectual-property/dpu.html`. Accessed: 2023-02-12.

[40] T. J. O'shea and N. West, "Radio machine learning dataset generation with gnu radio," in *Proceedings of the GNU Radio Conference*, vol. 1, 2016.

[41] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[42] 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*, pp. 770–778, 2016.

[43] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020.

[44] Xilinx, "RF Modulation Recognition-with Vitis AI." `https://github.com/Xilinx/Vitis-AI-Tutorials/tree/1.4/Design_Tutorials/10-RF_modulation_recognition`. Accessed: 2023-02-13.