Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

# Unifying Software Optimization and Hardware Acceleration Techniques in Telecom Applications

Διπλωματική Εργασία

*του*

**Ηλία Παπαλάμπρου**

*Επιβλέπων:* Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Αθήνα, Οκτώβριος 2023

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

# Unifying Software Optimization and Hardware Acceleration Techniques in Telecom Applications

Διπλωματική Εργασία

*του*

**Ηλία Παπαλάμπρου**

*Επιβλέπων:* Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31η Οκτωβρίου 2023.

| *(Υπογραφή)* | *(Υπογραφή)* | *(Υπογραφή)* |
|---|---|---|
| Δημήτριος Σούντρης<br>Καθηγητής Ε.Μ.Π. | Διονύσιος Ρεΐσης<br>Καθηγητής Ε.Κ.Π.Α. | Σωτήριος Ξύδης<br>Επίκουρος Καθηγητής Ε.Μ.Π. |

Αθήνα, Οκτώβριος 2023

*(Υπογραφή)*

---

**Παπαλάμπρου Ηλίας**
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών, Ε.Μ.Π.

# *Περίληψη*

Τα τελευταία χρόνια, με την ανάπτυξη των ψηφιακών επικοινωνιών, οι εφαρμογές για διάφορα ασύρματα δίκτυα απαιτούν τη σχεδίαση ενός Chip, για την ικανοποίηση των απαιτήσεων απόδοσης. Η έννοια ενός συστήματος πολλαπλών επεξεργαστών σε ένα Chip είναι κατάλληλη για τη διευκόλυνση της υλοποίησης διαφόρων τηλεπικοινωνιακών αλγορίθμων για διάφορα ασύρματα πρωτόκολλα σε μία μόνο συσκευή. Σε αυτή τη διπλωματική εργασία διερευνήσαμε τα οφέλη από τις βελτιστοποιήσεις σε επίπεδο λογισμικού καθώς και τον σχεδιασμό επιταχυντών υλικού, στον τομέα των ψηφιακών επικοινωνιών. Αναφορικά με τις βελτιστοποιήσεις λογισμικού, υλοποιήθηκε ο αλγόριθμος της αποδιαμόρφωσης για QAM αστερισμούς σε ενσωματωμένες πλατφόρμες με ARM επεξεργαστές. Επωφεληθήκαμε από τη χρήση εντολών Single Instruction Multiple Data (SIMD) που παρέχονται από τον NEON (C++ Intrinsics). Επιπλέον για δύο αστερισμούς QAM ερευνήθηκε μια προσεγγιστική υλοποίηση του αρχικού αλγορίθμου, προκειμένου να μειωθούν οι απαιτούμενες αριθμητικές πράξεις. Σχετικά με τον επιταχυντή υλικού, υλοποιήθηκε με VHDL ένας προσομοιωτής fading καναλιού σε Field Programmable Gate Array (FPGA). Για τη μοντελοποίηση του καναλιού, το σήμα εισόδου επεξεργάζεται με ένα φίλτρο πεπερασμένης κρουστικής απόκρισης (FIR), του οποίου οι συντελεστές παράγονται σε πραγματικό χρόνο και ακολουθούν μια κανονική κατανομή Gauss. Η αξιολόγηση των προαναφερθέντων υλοποιήσεων σε επίπεδο λογισμικού πραγματοποιήθηκε με την προσομοίωση μιας ψηφιακής τηλεπικοινωνιακής αλυσίδας όπου λήφθηκαν υπόψιν δύο παράγοντες: ο χρόνος εκτέλεσης και η ακρίβεια του στυστήματος με τη μετρική Bit Error Rate (BER), με/χωρίς τη χρήση Forward Error Correction (FEC). Ειδικότερα, ανάλογα με τον αστερισμό, η προτεινόμενη υλοποίηση με μικρές αποκλίσεις στο BER, παρουσιάζει $38\times$ επιτάχυνση σε σχέση με το αρχικό μοντέλο, καθώς και συνολική βελτίωση 10-20% για τη μονάδα του δέκτη με FEC. Όσο αναφορά τον εξομοιωτή Fading καναλιού, εξετάσαμε τη συνάρτηση πυκνότητας πιθανότητας (PDF) των δεδομένων εξόδου, ενώ η σύγκριση με ένα μοντέλο με αριθμητική κινητής υποδιαστολής έξειξε αμελητέα απώλεια ακρίβειας. Συγκεκριμένα τα σύμβολα στην έξοδο του εξομοιωτή καναλιού, παρουσιάζουν <0.5% Μέσο Σχετικό Σφάλμα, ενώ το μοντέλο μπορεί να λειτουργήσει σε συχνότητα 500MHz.

***Keywords*** – ARM, SIMD, NEON Intrinsics, FPGA, Επιταχυντής Υλικού, VHDL, Τηλεπικοινωνίες, QAM, Εξομοίωση Καναλιού

# *Abstract*

In recent years with the development of digital communications, applications for various wireless networks necessitate a single-chip design to meet performance requirements. The Multi-Processor System-on-Chip concept is well suited to facilitate the implementation of various telecommunication algorithms for multiple wireless standards within a single device. In this thesis we explored the benefits from software optimizations as well as hardware acceleration techniques, in the field of digital communications. Regarding the Software Optimizations, the Quadrature Amplitude Demodulation algorithm was implemented on ARM-based embedded platform. We benefited from the use of Single Instruction Multiple Data (SIMD) commands the NEON engine provides (C++ Intrinsics). Furthermore, for two QAM Constellations an approximation technique over the original Demodulation algorithm was proposed, in order to reduce the required arithmetic operations. About the hardware accelerator, a Field Programmable Gate Array (FPGA) based fading channel emulator was implemented in VHDL. For the modelling of a fading channel, the input signal has to be processed by a Finite Impulse Response (FIR) Filter, whose coefficients are generated in real time and follow a Normal Gaussian distribution. The evaluation of the aforementioned software implementations was conducted by simulating a Digital Telecommunication chain and taking into account two factors: the execution time and the accuracy of the system with the Bit Error Rate (BER) metric, with and without the use of Forward Error Correction (FEC). In particular, depending on the constellation, our proposed Demodulation implementation, having minimal BER deviations, shows up to $38\times$ speedup over the initial floating point model, and an overall 10-20% improvement for a Receiver Module with FEC decoding. Regarding the Fading Channel emulator, we examined the Probability Density Function (PDF) of the output, while a comparison with a floating-point base model showed negligible loss in precision. Specifically the fixed-point faded symbols present <0.5% Mean Relative Error, while the design is capable of operating at 500MHz frequency.

***Keywords*** — ARM, SIMD, NEON Intrinsics, FPGA, Hardware Accelerator, VHDL, Telecommunications, QAM, Fading Channel

# *Ευχαριστίες*

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κύριο Δημήτριο Σούντρη, που μου έδωσε την ευκαιρία να εκπονήσω μια ενδιαφέρουσα διπλωματική εργασία στο Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων. Στη συνέχεια, θα ήθελα να ευχαριστήσω τους υποψήφιους διδάκτορες Ιωάννη Στρατάκο και Γεώργιο Αρμενιάκο για όλη τη βοήθεια που προσέφεραν, την καθοδήγηση, καθώς και την επίλυση αποριών κατά τη διάρκεια εκπόνησης της διπλωματικής εργασίας. Ακόμα θα ήθελα να ευχαριστήσω όλα τα μέλη στο Microlab τα οποία δημιουργούν ένα ευχάριστο και παραγωγικό περιβάλλον, ενώ είμαι ιδιαίτερα χαρούμενος που θα έχω τη δυνατότητα να συνεχίσω εκεί τα επόμενα χρόνια. Θα ήθελα επίσης να ευχαριστήσω τους γονείς μου και τον αδερφό μου, που ήταν δίπλα μου όλα αυτά τα χρόνια και με στήριξαν στις επιλογές μου. Τέλος, ευχαριστώ όλους μου τους φίλους καθώς και τη Χριστίνα, για τις όμορφες στιγμές μαζί τους.

*Ηλίας Παπαλάμπρου*
*Οκτώβριος 2023*

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **AWGN** | Additive White Gaussian Noise |
| **BER** | Bit Error Rate |
| **DSP** | Digital Signal Processing |
| **FEC** | Forward Error Correction |
| **FIR** | Finite Impulse Response |
| **FPGA** | Field Programmable Gate Array |
| **LDPC** | Low-Density Parity Check |
| **LLR** | Log-Likelihood Ratio |
| **LoS** | Line of Sight |
| **LZD** | Leading Zero Detector |
| **MRE** | Mean Relative Error |
| **PDF** | Probability Density Function |
| **QAM** | Quadrature Amplitude Modulation |
| **SIMD** | Single Instruction Multiple Data |
| **SoC** | System on Chip |
| **URNG** | Uniform Random Number Generator |

# Εκτεταμένη Ελληνική Περίληψη

## Εισαγωγή

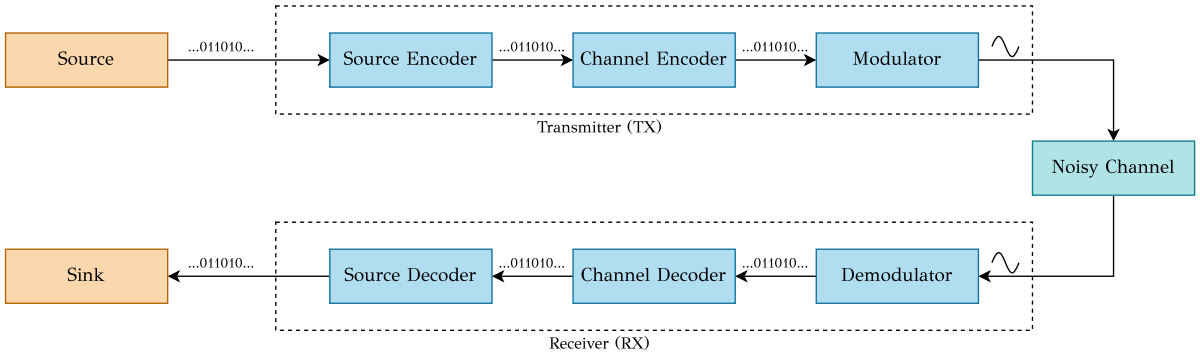
Οι ψηφιακές επικοινωνίες χρησιμοποιούνται για σήματα που περιλαμβάνουν δεδομένα όπως ο ήχος, οι εικόνες, κείμενο, αρχεία κλπ. Με τη συνεχή αύξηση της ζήτησης μεταφοράς δεδομένων, είναι γνωστό πως η ψηφιακή επικοινωνία καθίσταται μία από τις βασικές τεχνολογίες της εποχής μας [9]. Επιπλέον, με τη συνεχή άνοδο των τεχνολογιών σχεδίασης ολοκληρωμένων κυκλωμάτων, που επιτρέπουν την ενσωμάτωση ολόκληρων συστημάτων σε ένα μόνο ολοκληρωμένο κύκλωμα, η χρήση του System-On-Chip σε τηλεπικοινωνιακές εφαρμογές αποτελεί ιδανική επιλογή. Τα SoCs αναφέρονται σε ένα σύστημα που περιλαμβάνει σχεδόν όλα τα βασικά στοιχεία που απαιτούνται για μια συγκεκριμένη εφαρμογή. Με άλλα λόγια, πρόκειται για ένα ολοκληρωμένο σύστημα όπου η Κεντρική Μονάδα Επεξεργασίας, η μνήμη, οι θύρες εισόδου/εξόδου, οι αναλογικές είσοδοι και έξοδοι, καθώς και εξειδικευμένα κυκλώματα ανάλογα με την εφαρμογή, προσδιορίζονται να συνδυαστούν σε μια μονάδα [10].

## Θεωρητικό Υπόβαθρο

Στην ενότητα αυτή θα εξετάσουμε τα θεμελιώδη στοιχεία ενός ψηφιακού συστήματος επικοινωνίας, καθώς και τη θεωρία των αλγορίθμων αποδιαμόρφωσης, καθώς και τεχνικές μοντελοποίησης καναλιών επικοινωνίας. Επιπλέον παρουσιάζεται μια σύντομη επισκόπηση της αρχιτεκτονικής SoC-FPGA, όπου αποτέλεσε την πλατφόρμα που χρησιμοποιήθηκε κατά τη διπλωματική εργασία.

### Ψηφιακές επικοινωνίες

Ένα τυπικό ψηφιακό σύστημα επικοινωνίας παρουσιάζεται στο Σχ. 1, το οποίο ανεξάρτητα από την εφαρμογή μπορεί να χωριστεί σε τρία κύρια μέρη: τον Πομπό (TX), το κανάλι και τον δέκτη (RX) [11]. Αναλυτικότερα, ο πομπός είναι υπεύθυνος για τη μετατροπή του μηνύματος σε μορφή κατάλληλη για μετάδοση μέσω ενός φυσικού καναλιού, (π.χ. ένα καλώδιο). Ο ρόλος του κωδικοποιητή πηγής περιλαμβάνει την εξάλειψη των περιττών πληροφοριών από το μήνυμα προκειμένου να βελτιστοποιηθεί η χρήση του καναλιού. Ο Διαμορφωτής πλάτους λαμβάνει

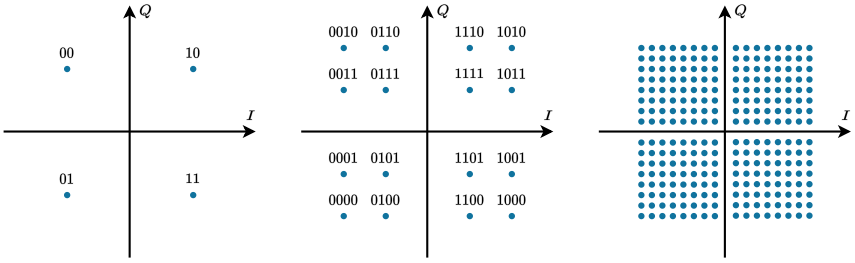



**Σχήμα 1**: Τηλεπικοινωνιακή αλυσίδα

μια σειρά από bits του κωδικοποιητή καναλιού και μεταφράζει κάθε ακολουθία σε μια κυματομορφή, κατάλληλη για μετάδοση. Το φυσικό μέσο μεταξύ της μονάδας πομπού και του δέκτη ονομάζεται κανάλι επικοινωνίας, το οποίο προσθέτει θόρυβο στο σήμα καθώς δεν υπάρχει κανένα ιδανικό μέσο. Μπορούν να συμβούν διάφορες μορφές τροποποιήσεων στο σήμα, όπως εξασθένηση, θόρυβος και παραμόρφωση. Η μονάδα του δέκτη (RX) ενός τηλεπικοινωνιακού συστήματος, είναι υπεύθυνη για την εξαγωγή του λαμβανόμενου σήματος, εκτελώντας αποδιαμόρφωση, μαζί με ορισμένες συμπληρωματικές λειτουργίες όπως το φιλτράρισμα και η ενίσχυση, δεδομένου ότι οι επιδράσεις της υποβάθμισης του σήματος πρέπει να αντιμετωπιστούν. Η αποδιαμόρφωση χειρίζεται τις λαμβάνουσες κυματομορφές και αντιστοιχίζει την κάθε μία με την αντίστοιχη ακολουθία bit.

### QAM Διαμόρφωση

Μια από τις πολλές διαφορετικές τεχνικές ψηφιακής διαμόρφωσης ονομάζεται τετραγωνική διαμόρφωση πλάτους (QAM), η οποία λειτουργεί με τη μετάδοση δύο σημάτων DSB χρησιμοποιώντας φορείς της ίδιας συχνότητας. Η διαμόρφωση M-ary QAM επεκτείνει την έννοια της M-ary PAM σε δύο διαστάσεις. Με αυτόν τον τρόπο, συνδυάζοντας το πλάτος και τη φάση, ο πομπός μπορεί να στείλει περισσότερα bit ανά σύμβολο. Για παράδειγμα, ορισμένοι συνηθισμένοι αστερισμοί όπως QAM4, QAM16, QAM256 κλπ. μεταδίδουν όπως φαίνεται και στο σχήμα 2 2, 4 και 8-bit αντίστοιχα. Επιπλέον ορισμένοι αστερισμοί QAM που απαιτούν περιττό αριθμό bits (όπως QAM512), απεικονίζονται χρησιμοποιώντας μια διάταξη σταυρού, έναντι τετραγωνικής [12]. Για κάθε αστερισμό QAM, χρησιμοποιείται κώδικας Gray για την αντιστοίχιση, χρησιμοποιώντας μια συγκεκριμένη διάταξη bits για κάθε σύμβολο, όπου οι διαδοχικές τιμές διαφέρουν σε ένα μόνο bit. Αυτή η τεχνική χρησιμοποιείται προκειμένου να μειωθούν τα σφάλματα αποδιαμόρφωσης λόγω προστιθέμενου θορύβου.




**(i)** QAM4 ($2^2$ bits) **(ii)** QAM16 ($2^4$ bits) **(iii)** QAM256 ($2^8$ bits)

**Σχήμα 2**: QAM αστερισμοί για διαφορετικά bits μετάδοσης

### Soft Αποδιαμόρφωση

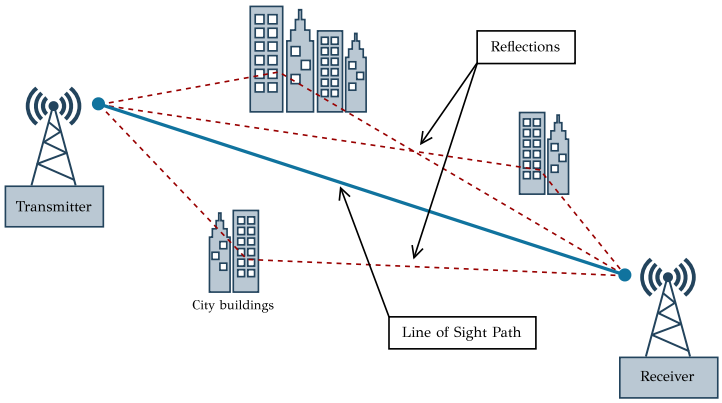
Σε αντίθεση με μέθοδο Hard αποδιαμόρφωσης, όπου το λαμβανόμενο σήμα απεικονίζεται στο πλησιέστερο σύμβολο του αστερισμού, η μέθοδος Soft αποδιαμόρφωσης χρησιμοποιεί μια μετρική πιθανότητας η οποία υποδεικνύει την πιθανότητα ότι κάθε σύμβολο που μεταδίδεται ήταν αρχικά ”1” ή ”0”. Αυτή η μέθοδος, αν και έχει υψηλότερη υπολογιστική πολυπλοκότητα σε σχέση με τη χρήση του αλγορίθμου της Hard αποδιαμόρφωσης, έχει καλύτερες επιδόσεις αναφορικά με το Bit Error Rate (BER). Το Log-Likelihood Ratio (LLR) αντιπροσωπεύει τον λογάριθμο του λόγου των πιθανοτήτων μετάδοσης ενός bit ”0” έναντι της μετάδοσης ενός bit ”1” για ένα λαμβανόμενο σήμα $r$.

Ο υπολογισμός της μετρικής Approximate Log-Likelihood Ratio (LLR) περιλαμβάνει τη χρήση μόνο του πλησιέστερου σημείου αστερισμού στο λαμβανόμενο δείγμα, θεωρώντας ”0” ή ”1” στη συγκεκριμένη θέση bit, σε αντίθεση με τον ακριβή υπολογισμό του LLR, όπου λαμβάνονται υπόψιν όλα τα σημεία του αστερισμού. Όπως ορίζεται στο [13, 14, 15], το Approximate LLR ($ALLR$) δίνεται από την Εξ. 1.

$$ALLR_b = -\frac{1}{\sigma^2}\left\{\max_{s\in S_1}\left[(x-s_x)^2+(y-s_y)^2\right]-\max_{s\in S_0}\left[(x-s_x)^2+(y-s_y)^2\right]\right\} \quad (1)$$

## Μοντελοποίηση Καναλιού

Τα φαινόμενα εξασθένισης και πολλαπλών διαδρομών συναντώνται συνήθως στα συστήματα ραδιοεπικοινωνίας. Στις ασύρματες επικοινωνίες, μπορεί να υπάρχουν διάφορες διαδρομές σήματος μεταξύ των κεραιών πομπού και δέκτη. Αυτά τα πολλαπλά μονοπάτια προκύπτουν από την ατμοσφαιρική σκέδαση, τη διάθλαση ή τις ανακλάσεις από αντικείμενα όπως τα κτήρια, όπως φαίνεται και στο παράδειγμα του Σχ. 3. Σε περιπτώσεις πολλαπλών διαδρομών, τα σήματα που φτάνουν μέσω διαφορετικών διαδρομών παρουσιάζουν εξασθένηση και καθυστερήσεις, οι οποίες μπορούν είτε να ενισχύσουν είτε να μειώσουν το σήμα που




**Σχήμα 3**: Φαινόμενα Fading καναλιού σε αστικό περιβάλλον

μεταδίδεται μεταξύ δύο κεραιών. Για να κατανοήσουμε τα χαρακτηριστικά του φαινομένου αυτού, μπορούμε αρχικά να εξετάσουμε ένα στατικό σενάριο, όπου ένας ακίνητος δέκτης αλληλεπιδρά με ένα μεταδιδόμενο σήμα που αποτελείται από ένα σήμα στενής ζώνης, ενώ κάνουμε την υπόθεση πως δύο εξασθενημένα αντίγραφα του μεταδιδόμενου σήματος φτάνουν στον δέκτη διαδοχικά. Η ύπαρξη καθυστέρησης μεταξύ των σημάτων εισάγει μετατόπιση φάσεις μεταξύ των συνιστωσών του λαμβανόμενου σήματος.

Μια βασική αναπαράσταση για διακριτά κανάλια πολλαπλών διαδρομών περιγράφεται στην Εξ. 2, όπου το $s(t)$ αντιπροσωπεύει το ζωνοπερατό σήμα εισόδου, το $a_n(t)$ είναι η εξασθένηση για το σήμα που λαμβάνεται μέσω του n-οστού μονοπατιού, ενώ το $\tau_n(t)$ είναι η σχετική καθυστέρηση διάδοσης. Συνεπώς το κανάλι εξασθένισης πολλαπλών διαδρομών μπορεί να υλοποιηθεί ως γραμμικό φίλτρο πεπερασμένης κρουστικής απόκρισης (FIR).

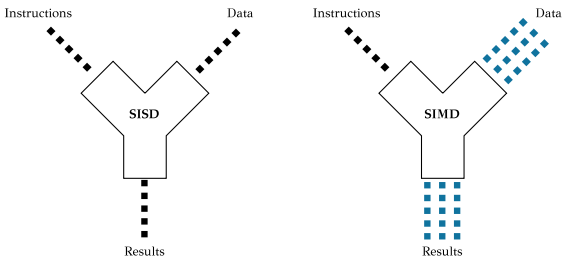
$$y(t) = \sum_n a_n(t)s(t - \tau_n(t)) \tag{2}$$

## SoC-FPGA

Οι πλατφόρμες System-on-Chip (SoC) FPGA αναφέρονται σε διατάξεις ημιαγωγών που συνδυάζουν προγραμματιζόμενη λογική (PL) με πυρήνες επεξεργαστών ενσωματωμένων συστημάτων (π.χ. ARM). Το μοντέλο αυτό συνδυάζει την ευκολία του προγραμματισμού ενός επεξεργαστή με την αποτελεσματικότητα και την απόδοση της προγραμματιζόμενης λογικής.

### FPGA

Ένα Field Programmable Gate Array (FPGA) αποτελείται από τρία θεμελιώδη στοιχεία: μια λογική μονάδα που μπορείνα προγραμματιστεί, ένα στοιχείο εισόδου/εξόδου που μπορεί να προσαρμοστεί για εξωτερικές συνδέσεις, καθώς και




**Σχήμα 4**: Διαφορές μεταξύ SISD και SIMD εντολών

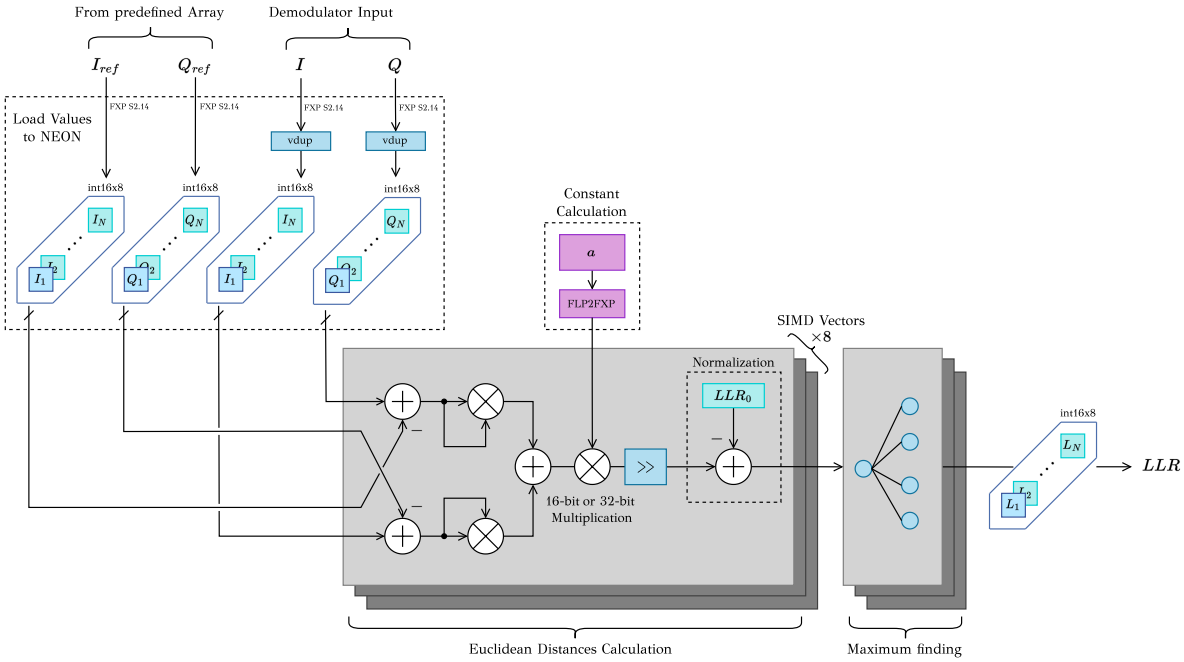
ένα στοιχείο διασύνδεσης που μπορεί να προγραμματιστεί για να συνδέσει διάφορα τμήματα μεταξύ τους. Επιπλέον, υπάρχουν μονάδες Ψηφιακής Επεξεργασίας Σήματος (DSP) και ενσωματωμένη μνήμη, προκειμένου να ενισχυθούν οι υπολογιστικές λειτουργίες. Μεταφέροντας τα δεδομένα σε αυτά τα στοιχεία, ένα FPGA μπορεί να δημιουργήσει το προβλεπόμενο ψηφιακό κύκλωμα.

### ARM NEON

Οι επεξεργαστές ARMv8 μαζί με τους καταχωρητές γενικού σκοπού, συμπληρώνουν 32 καταχωρητές SIMD 128-bit σε κάθε πυρήνα, οι οποίοι χρησιμοποιούνται για την υλοποίηση των εντολών NEON. Οι εντολές NEON υποστηρίζουν 8-bit, 16-bit, 32-bit και 64-bit για προσημασμένους και μη προσημασμένους ακεραίους, 32-bit στοιχεία κινητής υποδιαστολής μονής ακρίβειας, καθώς και 8-bit/16-bit πολυώνυμα. Η τεχνολογία NEON προσφέρει μια εξειδικευμένη επέκταση στην αρχιτεκτονική ARM Intruction Set Architecture, εισάγοντας επιπλέον εντολές ικανές να εκτελούν δεδομένα επεξεργασίας δεδομένων ταυτόχρονα σε πολλαπλές ροές δεδομένων. Οι εντολές Single Instruction Multiple Data (SIMD) ενισχύουν τη συνολική αποτελεσματικότητα της εφαρμογής, επιτρέποντας με μία μόνο εντολή να χειρίζεται ταυτόχρονα πολλαπλά σύνολα δεδομένων, μέσω ειδικά κατασκευασμένης λογικής. Στο Σχ. 4 παρουσιάζεται η διαφορά μεταξύ της αρχιτεκτονικής των εντολών Single Instruction Single Data (SISD) και SIMD.

## SIMD Αποδιαμόρφωση

Η μονάδα QAM αποδιαμόρφωσης βασίζεται στην εξίσωση 1, οποία περιλαμβάνει τον υπολογισμό του προσεγγιστικού LLR για εισερχόμενα σύμβολα μέσα σε έναν αστερισμό QAM. Η υλοποίηση εξυπηρετεί διαφορετικά bits ανά σύμβολο για πληθώρα αστερισμών, όπως QAM4, QAM16, QAM64, QAM256, QAM512 και QAM1024. Η διαδικασία ξεκινάει με τον υπολογισμό της ευκλείδειας απόστασης μεταξύ των εισερχόμενων συντεταγμένων $(I, Q)$ και όλων των πιθανών συμβόλων εντός του επιλεγμένου αστερισμού. Στη συνέχεια, με την εύρεση της μέγιστης τιμής



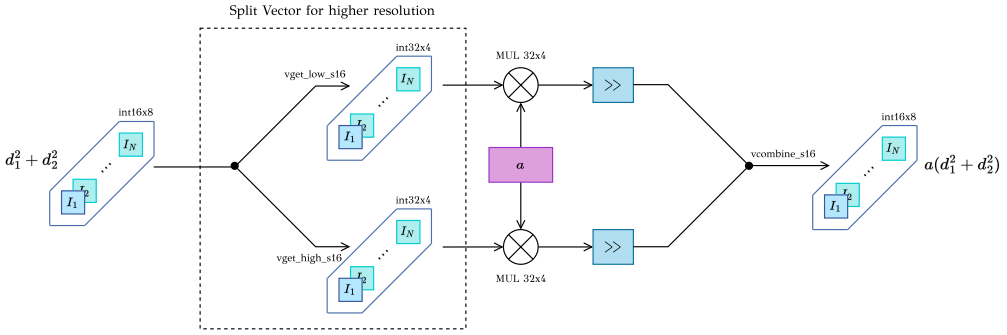

**Σχήμα 5**: Υλοποίηση της Αποδιαμόρφωσης με SIMD εντολές

LLR για τα bit που έχουν ”1” και ”0” αντίστοιχα, προκύπτει το τελικό αποτέλεσμα. Τονίζεται πως οι LLR τιμές κανονικοποιούνται με το πρώτο σύμβολο του αστερισμού, εξασφαλίζονται ίδιο εύρος τιμών για διαφορετικές τιμές στην είσοδο.

Στο Σχ. 5 απεικονίζεται η υλοποίηση της SIMD Αποδιαμόρφωσης, με χρήση NEON Intrinsics. Η υλοποίηση μπορεί να χωριστεί σε τρία κύρια μέρη: τη φόρτωση δεδομένων στη μηχανή NEON, τους μαθηματικούς υπολογισμούς, όπου περιλαμβάνουν τους υπολογισμούς των ευκλείδειων αποστάσεων καθώς και την εύρεση της μέγιστης τιμής, και τελικά την εξαγωγή δεδομένων από τον NEON πίσω στον ARM επεξεργαστή.

Συνολικά υλοποιήθηκαν 4 διαφορετικά μοντέλα για την αποδιαμόφωση, με βάση το μέγεθος των τιμών εισόδου (`int8` και `int16`), καθώς και με βάση την ακρίβεια του πολλαπλασιασμού με τη σταθερά $\alpha$. Η πρώτη προσέγγιση ήταν η χρήση του `vqdmulhq_n_s16()` intrinsic, το οποίο χειρίζεται εσωτερικά τον πολλαπλασιασμό και επιστρέφει ένα αποτέλεσμα 16-bit, ίδιο με αυτό της εισόδου. Η δεύτερη προσέγγιση ήταν να χειριστούμε χειροκίνητα το αποτέλεσμα του πολλαπλασιασμού, αναθέτοντας το σε ένα προσωρινό διάνυσμα, και στη συνέχεια να γίνει η μετατροπή του πίσω σε `int16`. Η διαδικασία αυτή απεικονίζεται στο Σχ. 6.

Στον αρχικό αλγόριθμο της αποδιαμόρφωσης, η τιμή LLR για κάθε bit χρειάζεται τις ίδιες συγκρίσεις, καθώς επαναχρησιμοποιεί τις προηγούμενες μέγιστες τιμές, προκειμένου να βελτιστοποιηθούν οι απαιτούμενες εντολές. Ως εκ τούτου, δημιουργήθηκε μια συνάρτηση για τον υπολογισμό της μέγιστης τιμής με χρήση του NEON, όπως απεικονίζεται στο Σχ. 7. Κατά τη διάρκεια της εύρεσης της μέγιστης




**Σχήμα 6**: SIMD πολλαπλασιασμός με 16-bit είσοδο και 32-bit έξοδο

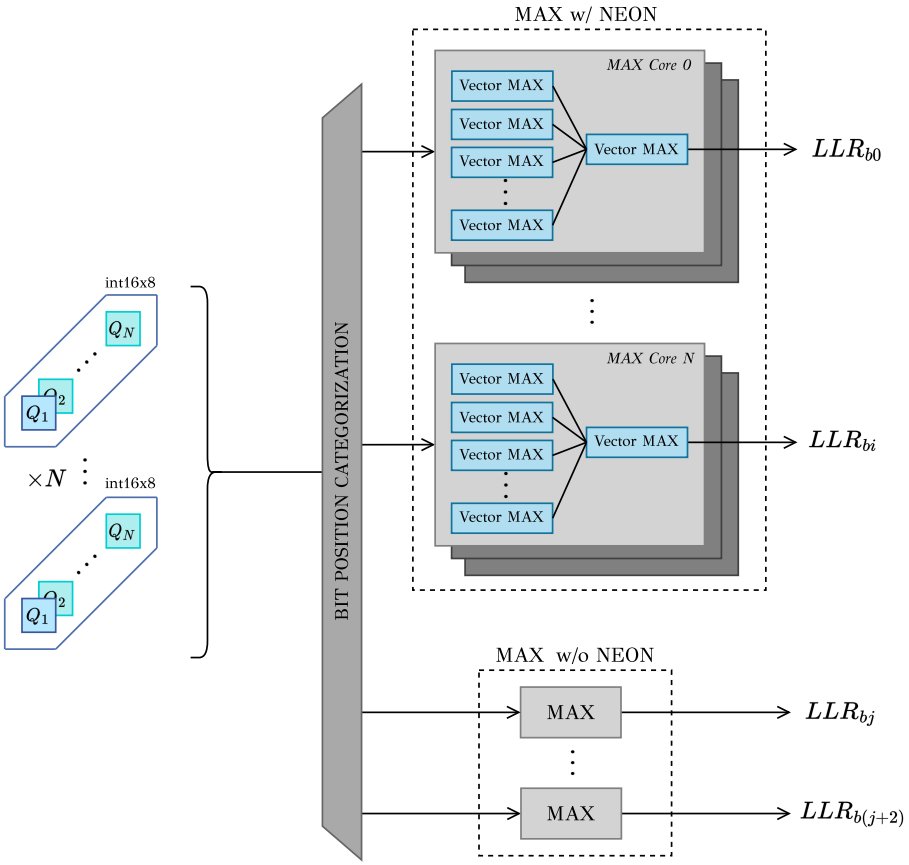
τιμής, δυο διαφορετικά NEON intrinsics χρησιμοποιήθηκαν, ένα για την εύρεση της μέγιστης τιμής σε ένα μόνο διάνυσμα, και ένα άλλο για τον υπολογισμό των μέγιστων τιμών μεταξύ δύο διανυσμάτων.

## Αποδιαμόρφωση με προσεγγιστικές τεχνικές σε QAM16/QAM64

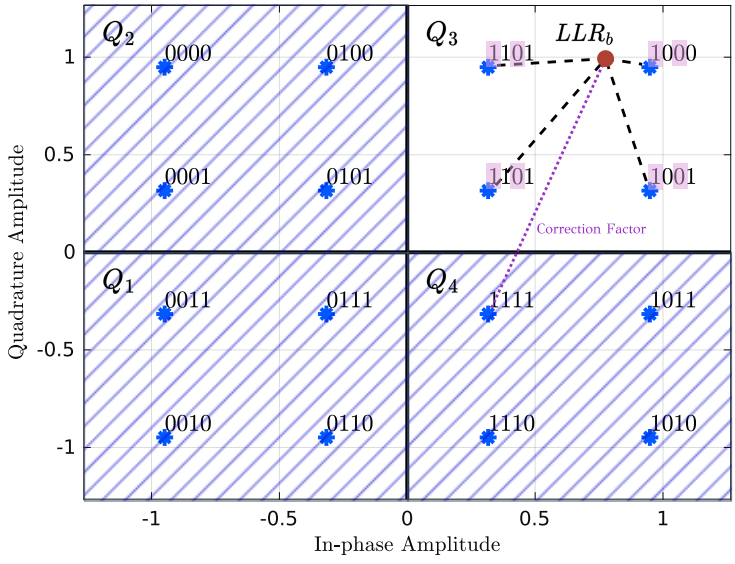
Όπως ήδη έχουμε δει στον αρχικό αλγόριθμό του Approximate LLR, για ένα σύμβολο εισόδου στο QAM16, υπολογίζονται 16 ευκλείδειες αποστάσεις για να βρεθεί η μέγιστη τιμή LLR για κάθε bit. Στη δική μας προσέγγιση, υπολογίζουμε μόνο τις ευκλείδειες αποστάσεις του τεταρτημορίου που ανήκει το σύμβολο εισόδου. Ο υπολογισμός κάθε τιμής LLR εξακολουθεί να χρησιμοποιεί τον αρχικό αλγόριθμο Approximate LLR που αναλύσαμε προηγουμένως.

Όπως απεικονίζεται στο Σχ. 8, μετά τον υπολογισμό των προσεγγιστικών τιμών LLR για ένα τεταρτημόριο, έχοντας συνολικά 4 αποστάσεις, χρειάζεται να βρούμε μόνο 2 μέγιστες αποστάσεις για τις θέσεις που υπάρχει ”1” και ”0” αντίστοιχα. Σημειώνουμε οτι η επιλογή του τεταρτημορίου γίνεται συγκρίνοντας με το 0 τις συντεταγμένες εισόδου $I$ και $Q$. Για τα bits στις θέσεις LSB-1 και MSB σε κάθε τεταρτημόριο που χρησιμοποιείται κώδικας Gray, παρατηρούμε οτι υπάρχει μόνο 1 ή 0 ως πιθανή τιμή bit, επομένως $s \notin S_0$ ή $s \notin S_1$ αντίστοιχα. Η αρχική μας προσέγγιση ήταν να παραβλέψουμε αυτό οτ ζήτημα και βασιστήκαμε αποκλειστικά στις διαθέσιμες αποστάσεις. Ωστόσο αυτή η προσέγγιση οδήγησε σε σημαντικά αυξημένες τιμές σφάλματος των τιμών LLR σε αυτά τα σημεία, με μεγάλες αποκλίσεις στο BER. Επομένως για αυτές τις ειδικές περιπτώσεις βρήκαμε μια εναλλακτική λύση για τον υπολογισμό της μέγιστης τιμής των μη υπαρχόντων στοιχείων. Έτσι υπολογίσαμε δύο πρόσθετες αποστάσεις, μία για κάθε περίπτωση - αν δεν υπάρχει κανένα ”1” ή αν δεν υπάρχει κανένα ”0” αντίστοιχα.

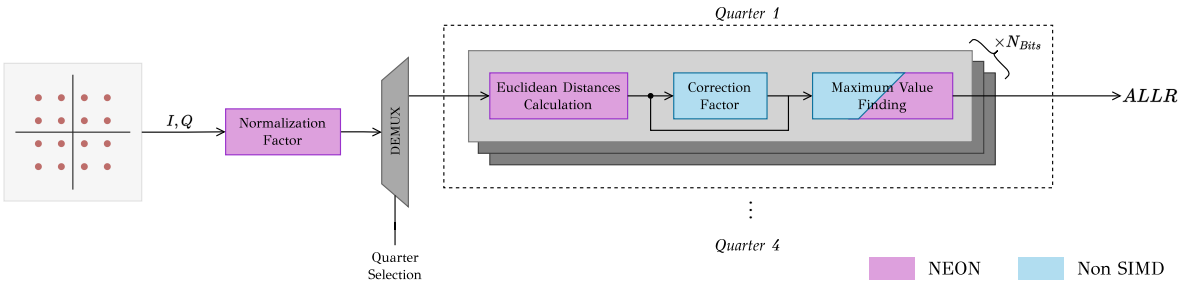
Επιπλέον, όσον αναφορά την κανονικοποίηση κάθε τιμής LLR με την τιμή αναφοράς, οι αρχικές τιμές LLR για κάθε σύμβολο υπολογίστηκαν αρχικά και αποθηκεύτηκαν χρησιμοποιώντας τον NEON. Στο Σχ. 9 παρουσιάζεται η αρχιτεκτονική



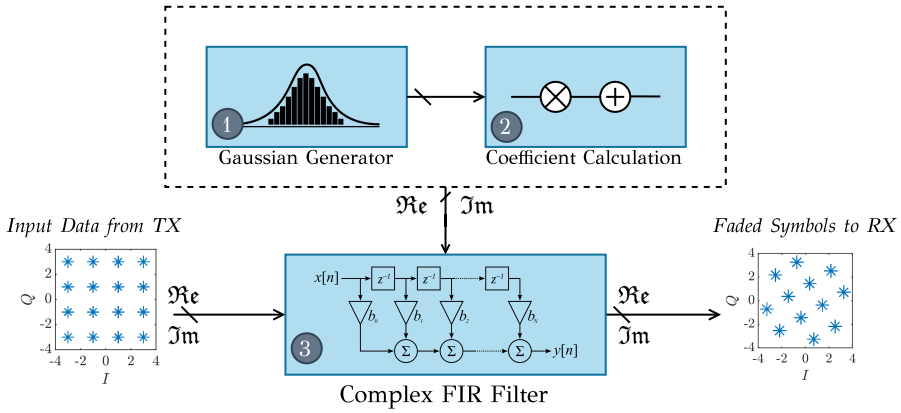

**Σχήμα 7**: Αρχιτεκτονική για τον υπολογισμό της μέγιστης τιμής LLR




**Σχήμα 8**: Προσεγγιστική αποδιαμόρφωση για QAM16




**Σχήμα 9**: NEON υλοποίηση για την προσεγγιστική QAM αποδιαμόρφωση



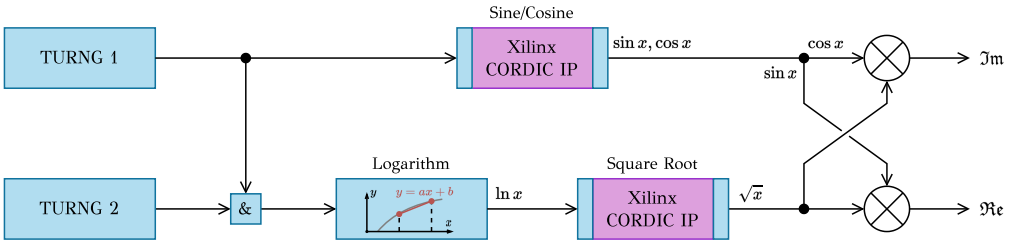

**Σχήμα 10**: Υλοποίηση του εξομοιωτή καναλιού διαλείψεων

για τον προσεγγιστικό αλγόριθμο αποδιαμόρφωσης. Για μεγαλύτερους αστερισμούς (QAM64), για την εύρεση μεγίστου γίνεται χρήση και του NEON.

Ακολουθώντας λοιπόν την ίδια αλγοριθμική προσέγγιση με τον QAM16 αστερισμό, επεκτείναμε την υλοποίηση μας προκειμένου να υποστηρίζει και QAM64 αστερισμούς. Συγκεκριμένα για ένα τεταρτημόριο, υπολογίσαμε τις 16 ευκλείδειες αποστάσεις που απαιτούνται χρησιμοποιώντας τον αλγόριθμο Approximate LLR και στη συνέχεια βρήκαμε για κάθε bit τη μέγιστη τιμή.

# Εξομοιωτής Καναλιών διαλείψεων σε FPGA

Στο Σχ. 3 απεικονίζεται η υλοποίηση του εξομοιωτή καναλιών διαλείψεων, η οποία μπορεί να χωριστεί σε τρία κύρια μέρη. Το *Gaussian Generator* είναι υπεύθυνο για τη δημιουργία τυχαίων μιγαδικών Γκαουσιανών μεταβλητών. Το *Coefficients Calculation* χρησιμοποιώντας τα χαρακτηριστικά του καναλιού (καθυστερήσεις διαδρομών ανάκλασης, μέση ισχύς, K-factor) υπολογίζει τους μιγαδικούς συντελεστές για το φίλτρο. Τέλος το *Complex FIR Filter*, φιλτράρει τα διαμορφωμένα σύμβολα εισόδου με τις μιγαδικές μεταβλητές του καναλιού, παράγοντας τελικά το εξασθενημένο σήμα.




**Σχήμα 11**: Μετασχηματισμός Box-Muller

## Γεννήτρια Γκαουσιανών Μεταβλητών

Οι περισσότερες από τις ψηφιακές τεχνικές για την παραγωγή τυχαίων γκαουσιανών μεταβλητών βασίζονται σε μετασχηματισμούς σε ομοιόμορφες τυχαίες μεταβλητές. Συνήθεις προσεγγίσεις περιλαμβάνουν τη μέθοδο Wallace, τη μέθοδο Inversion καθώς και τη μέθοδο που επιλέξαμε εμείς: τον Box-Muller μετασχηματισμό, ο οποίος χρησιμοποιεί μια ακολουθία συναρτήσεων για τη μετατροπή δύο ομοιόμορφων κατανεμημένων μεταβλητών σε δύο που ακολουθούν κανονική κατανομή. Για να δημιουργήσουμε τα επιθυμητά δείγματα, υποθέτοντας ότι $u_1$ και $u_2$ είναι ανεξέρτητες μεταβλητές στο διάστημα $[0, 1)$, χρησιμοποιούμε τις Εξ. 3 και 4. Στο Σχήμα 11 απεικονίζεται η βασική αρχιτεκτονική της υλοποίησης υλικού της μεθόδου Box-Muller, με βάση τις εξισώσεις που παρατίθενται παραπάνω.

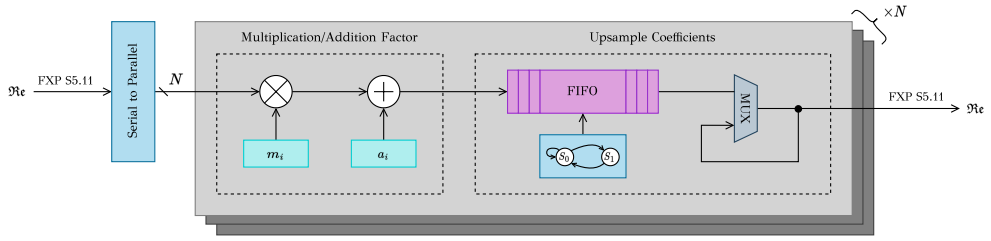
$$x_0 = \sqrt{-2\ln(u_0)}\cos(2\pi u_1) \tag{3}$$

$$x_1 = \sqrt{-2\ln(u_0)}\sin(2\pi u_1) \tag{4}$$

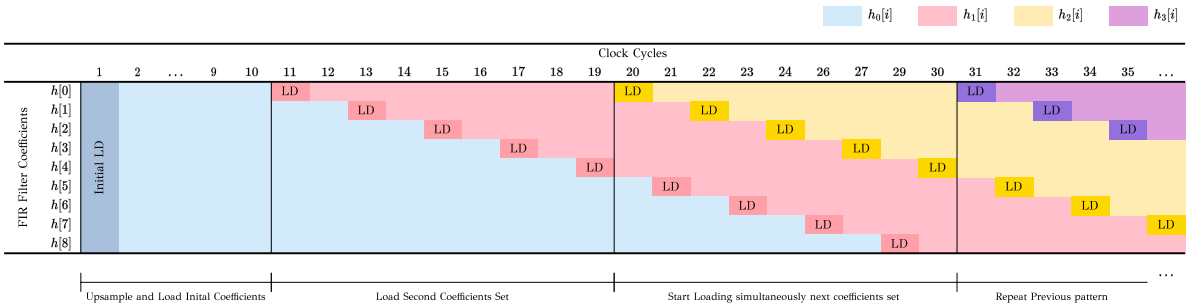
### Υπολογισμός Μιγαδικών Σταθερών

Στην υλοποίηση του Σχ. 12, οι σύνθετοι συντελεστές για το φίλτρο FIR υπολογίζονται παράλληλα. Επομένως μια Serial to Parallel μονάδα χρησιμοποιήθηκε, όπου μετά από $N$ κύκλους, $N$ νέες Γκαουσιανές μεταβλητές φορτώνονται σε τις μονάδες πολλαπλασιασμού/πρόσθεσης. Οι σταθερές $m_i$ και $a_i$ υπολογίζονται με την Εξ. 5, η οποία λαμβάνει υπόψιν τις παραμέτρους του καναλιού. Θέτοντας τον παράγοντα $K$ ίσο με 0, η κατανομή στην έξοδο τους καναλιού ακολουθεί κατανομή Rayleigh, διαφορετικά ακολουθεί Rician κατανομή.

$$m_i = \sqrt{\frac{PK}{K+1}}, \quad a_i = \sqrt{\frac{PK}{2(K+1)}} \tag{5}$$




**Σχήμα 12**: Υπολογισμός σταθερών

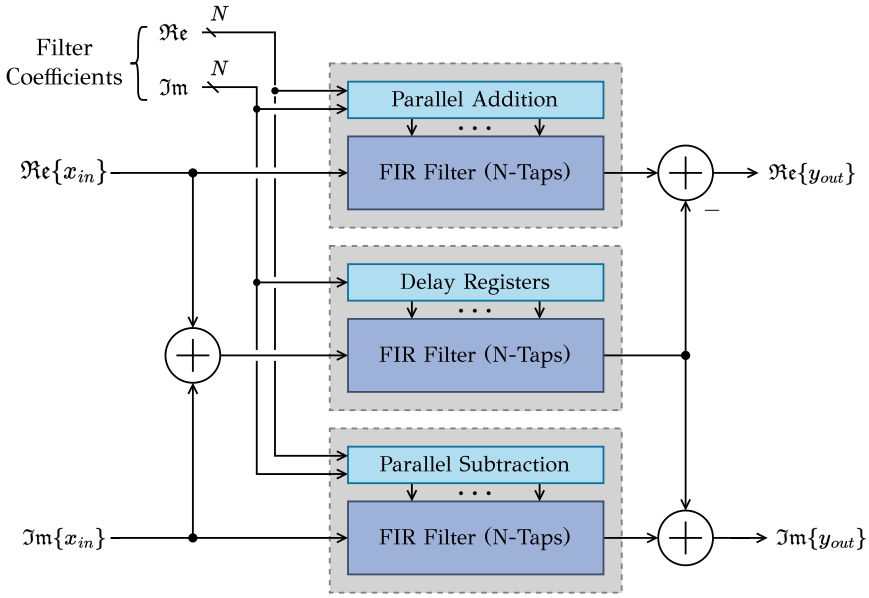



**Σχήμα 13**: FIR Coefficients Loading Pipeline

Επιπλέον, προκειμένου να ενημερώνονται οι συντελεστές του FIR φίλτρου κατά τη διάρκεια εκτέλεσης, πρέπει να ακολουθείται ένα κυκλικό μοτίβο φόρτωσης. Έτσι, χρησιμοποιήθηκε μια FIFO βασισμένη σε BRAM για κάθε αποτέλεσμα συντελεστή όπου με χρήση μιας μηχανής πεπερασμένων καταστάσεων (FSM), ελέγχαμε το σήμα ενεργοποίησης ανάγνωσης και εγγραφής. Οι χρονισμοί για την προαναφερθείσα διαδικασία, απεικονίζονται στο Σχ. 13, ο οποίοι μπορούν να κατηγοριοποιηθούν σε τρία κύρια τμήματα: η αρχική φόρτωση του πρώτου συνόλου συντελεστών, η φόρτωση του δεύτερου συνόλου ενώ δεν συμβαίνει καμία άλλη ενημέρωση συντελεστών και τέλος, το επαναλαμβανόμενο μοτίβο φόρτωσης των συντελεστών κατά τη διάρκεια εκτέλεσης. Κάθε φόρτωση τιμής (LD) συμβαίνει κάθε 2cc λόγω της τοπολογίας των καταχωρητών στο επιλεγμένο FIR.

### FIR Φίλτρο

Στη περίπτωση των γραμμικών φίλτρων, η έξοδος υπολογίζεται ως γραμμικός συνδυασμός της εισόδου και των προηγούμενων δειγμάτων εισόδου και εξόδου. Στην υλοποίηση μας, ο αριθμός των taps του FIR, καθορίζεται από τη μέγιστη καθυστέρηση διαδρομής ανάκλασης. Δεδομένου πως τόσο το σήμα όσο και οι συντελεστές του φίλτρου είναι μιγαδικοί αριθμοί, χρησιμοποιήθηκε μια υλοποίηση ενός μιγαδικού FIR φίλτρου, όπου χρησιμοποιούνται τρία φίλτρα FIR παράλληλα [16] (Σχ. 14).



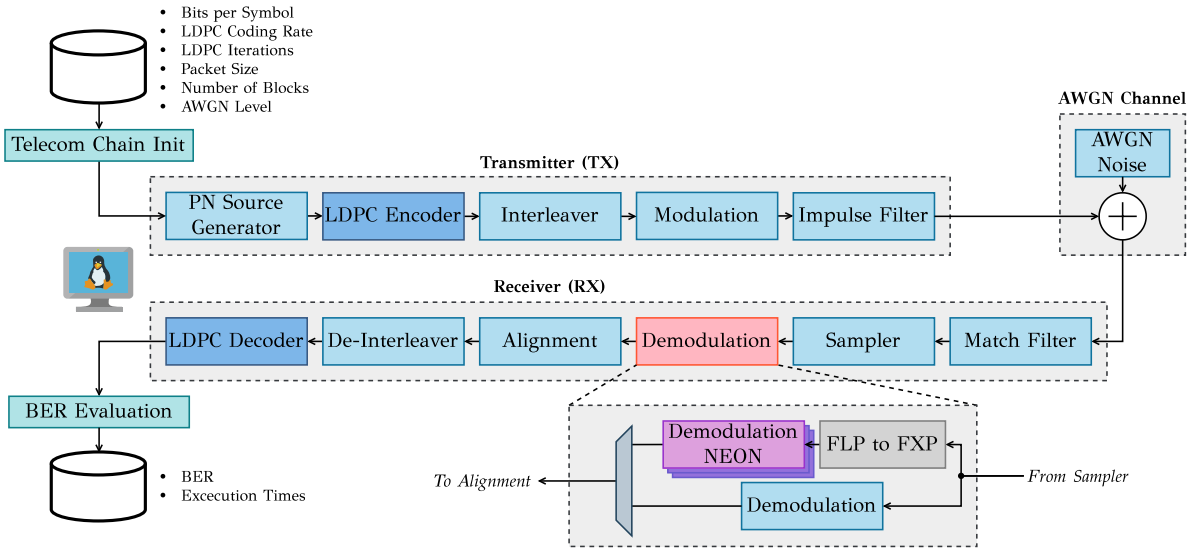

**Σχήμα 14**: Αρχιτεκτονική FIR

# Πειραματικά Αποτελέσματα

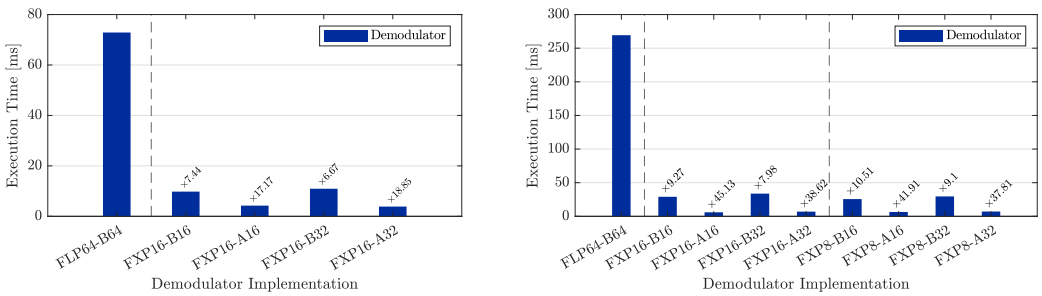
Για την υλοποίηση της τηλεπικοινωνιακής αλυσίδας, χρησιμοποιήσαμε τη βιβλιοθήκη TOPCOM++ η οποία αναπτύχθηκε σε γλώσσα C++. Με βάση την αρχιτεκτονική του DVB-S2 πρωτοκόλλου, ορισάμε τις κατάλληλες μονάδες, όπως φάινεται στο Σχ. 15. Επιπλέον, ως μια δευτερεύουσα πειρματική διάταξη, εξετάσαμε μια τηλεπικοινωνιακή αλυσίδα όπου ο κωδικοποιητής/αποκωδικοποιητής LDPC παρακάμτεται. Όλοι οι παράμετροι προσομοίωσης δίνονται από ένα αρχείο, ενώ τα αποτελέσματα της προσομοίωσης (BER, χρόνος εκτέλεσης, υπολογισμός επιτάχυνσης υλοποίησης) καταγράφονται επίσης σε αρχεία, ώστε να διευκολυνθεί η μεταγενέστερη επεξεργασία τους.

## SIMD Αποδιαμόρφωση

Για τους αστερισμούς QAM16/QAM64 εφαρμόσαμε την προσεγγιστική τεχνική που αναλύθηκε προηγουμένως, παρουσιάζουμε τα αποτελέσματα αναφορικά με τη βελτίωση του χρόνου εκτέλεσης, σε σχέση με τον αρχικό αλγόριθμο αποδιαμόρφωσης με χρήση SIMD λογικής. Εξετάσαμε την απόδοση της μονάδας της αποδιαμόρφωσης καθώς και ολόκληρου του δέκτη (RX) της αλυσίδας, σε δύο διαφορετικά σενάρια: χωρίς/με χρήση κώδικα FEC. Ανατρέχοντας στο Σχ. 16, όπου παρουσιάζεται μόνο ο χρόνος εκτέλεσης της αποδιαμόρφωσης, παρατηρούμε πως έχουμε επιτάχυνση έως $45\times$ σε σύγκριση με την αρχική υλοποίηση. Επιπλέον εφαρμόζοντας την προτεινόμενη τεχνική προσέγγισης, υπάρχει μια επιπλέον βελτίωση απόδοσης $3\times$.

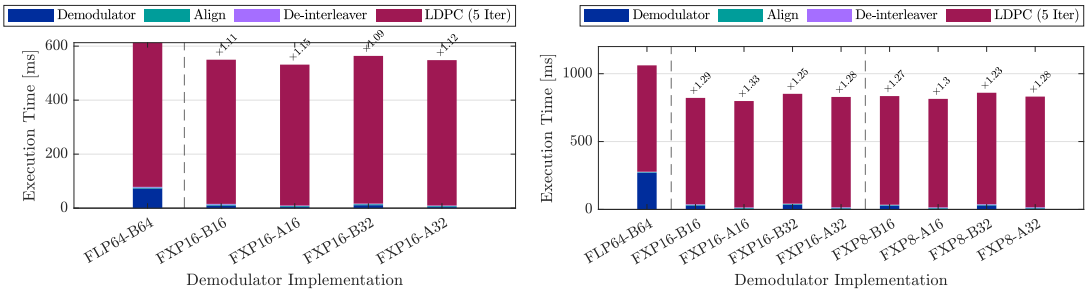



**Σχήμα 15**: Πειραματική διάταξη τηλεπικοινωνιακής αλυσίδας



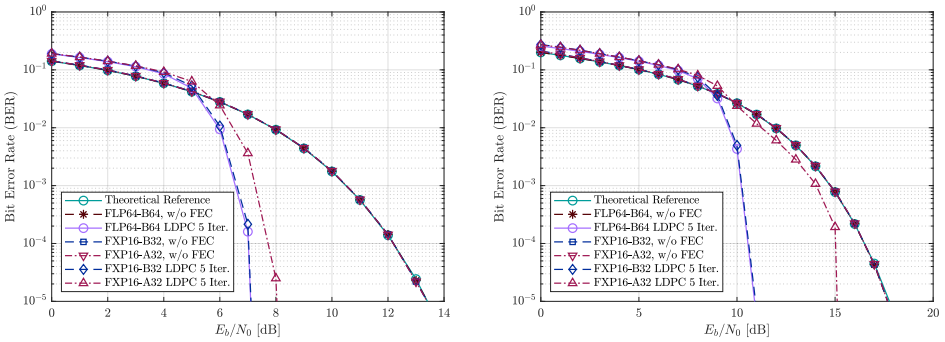

**(i)** QAM16 **(ii)** QAM64

**Σχήμα 16**: Χρόνος εκτέλεσης της SIMD QAM Αποδιαμόρφωσης




**(i)** QAM16, LDPC 5 Επαν. **(ii)** QAM64, LDPC 5 Επαν.

**Σχήμα 17**: Χρόνος εκτέλεσης του δέκτη με χρήση LDPC




**(i)** QAM16 **(ii)** QAM64

**Σχήμα 18**: Επιδόσεις BER για QAM16 και QAM64

Στη συνέχεια, εφαρμόζοντας LDPC κωδικοποίηση όπως φαίνεται στο Σχ. ??, γίνεται φανερό πως η αποκωδικοποίηση του σήματος, λαμβάνει σημαντικό χρόνο εκτέλεσης στη συνολική μονάδα του δέκτη.

Συγκρίνοντας όλες τις διαφορετικές υλοποιήσεις για τους αστερισμούς QAM16 και QAM64, καταλήξαμε στο συμπέρασμα πως η καλύτερη επιλογή μεταξύ ταχύτητας και ακρίβειας είναι το μοντέλο FXP16-A32, καθώς υπερέχει των άλλων όσον αφορά την ακρίβεια BER, ενώ παράλληλα προσφέρει επαρκή κέρδη αναφορικά με το χρόνο εκτέλεσης, σε σχέση με την αρχική υλοποίηση. Στο Σχ. 18 παρουσιάζονται τα αποτελέσματα για την επιλεγμένη υλοποίηση για τους δύο διαφορετικούς αστερισμούς.

## Εξομοιωτής Καναλιών διαλείψεων σε FPGA

### Απαιτούμενοι Πόροι στο FPGA

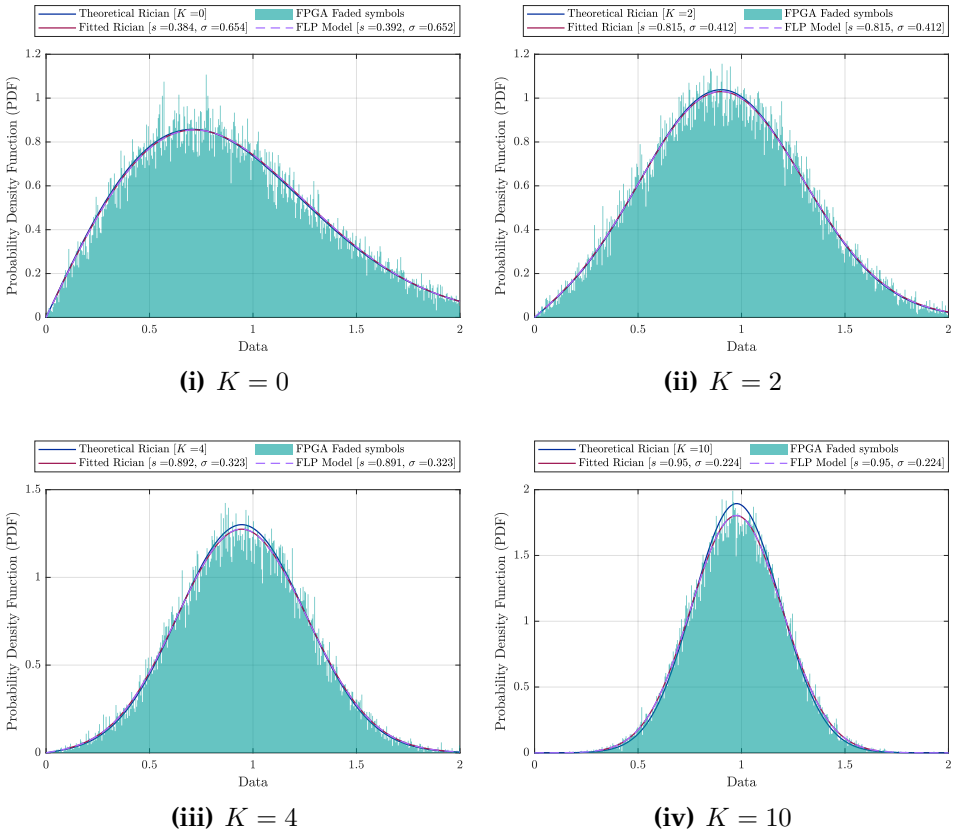
Με την ανάγκη ενός εξομοιωτή καναλιού που προσφέρει υψηλές αποδόσεις, η υλοποίηση έχει αρκετά pipelined στάδια, προκειμένου να μεγιστοποιηθεί η συχνότητα λειτουργίας. Για τη σύνθεση του μοντέλου χρησιμοποιήθηκε το εργαλείο

**Πίνακας 1**: Απαιτούμενοι Πόροι στο FPGA

| Πόροι | Απαιτούμενοι | Διαθέσιμοι | Ποσοστό Χρήσης |
|---|---|---|---|
| LUT | 2504 | 230400 | 1.09 % |
| FF | 3851 | 460800 | 0.84 % |
| BRAM | 9 | 312 | 2.88 % |
| DSP | 49 | 1728 | 2.84 % |

Xilinx Vivado 2022.2, όπου οι πόροι που καταλαμβάνει το FPGA φαίνονται στον Πίνακα 1. Όσον αναφορά την κατανάλωση ενέργειας, το εργαλείο σχεδίασης και σύνθεσης αναφέρει 1.514 Watt.

Για την αξιολόγηση του εξομοιωτή καναλιού, εξετάσαμε τη συνάρτηση πυκνότητας πιθανότητας (PDF) των εξασθενημένων συμβόλων. Στο Σχ. 19 παρουσιάζονται τα ιστογράμματα των συμβόλων στην έξοδο του εξοιμοιωτή, καθώς και 3 καμπύλες PDF. Η πρώτη είναι μια θεωρητική Rician κατανομή με βάση τον αντίστοιχο συντελεστή $K$ που χρησιμοποιείται. Η δεύτερη και τρίτη καμπύλη PDF αναφέρονται στην κατανομή που προκύπτει από τα ιστογράμματα του εξομοιωτή στο FPGA καθώς και ενός μοντέλου σε MATLAB. Από τα παραπάνω διαγράμματα συμπεραίνουμε πως τα παραγόμενα σύμβολα ακολουθούν την θεωρητική καμπύλη PDF, ενώ συγχρόνως σε σύγκριση με το μοντέλο στο MATLAB, έχουν μικρή απώλεια ακρίβειας.




**(i)** $K = 0$ **(ii)** $K = 2$

**(iii)** $K = 4$ **(iv)** $K = 10$

**Σχήμα 19**: Συνάρτηση Πυκνότητας Πιθανότητας για διαφορετικά $K$

# Chapter 1

# Introduction

Digital Communication is employed for signals that exist inherently in an analog and continuous-form, including content like audio, images or other continuous data data streams. Simultaneously, it is utilized for signals that are digital, such as text files and discrete data sets. With the continuous rise of the demand of data transfer, it is well known that digital communication is becoming one of the key technologies of our time [9].

Furthermore, with the continuous rise of chip design technologies, allowing for the integration of entire systems on a single ICs, the integration of System-On-Chip in telecommunication applications is an ideal implementation choice. SoCs refers to a VLSI system that encompasses nearly all the essential components required for a specific application. In other words, they are microchips where the CPU, internal memory, I/O ports, analog input and output, and specialized application-dependent circuit blocks are all intended to be combined within a singular chip [10].

## 1.1 Related Works

The implementation of digital communication protocols in software, is a critical research area for communication stakeholders, presenting substantial benefits such as hardware integration, energy efficiency, scalability, as well as ease of maintenance [17]. Over the past years, different researchers focus on implementing digital communication protocols on software, like for example in paper [18], where the authors have successfully constructed a DVB-S2 receiver achieving a 10 Gb/s throughput using a cluster of server-class CPUs. Other works like [19], focus on low power and smaller scale devices, where an Internet-of-Things (IoT) software-defined radio platform was introduced on ARM-Cortex-M4 processor. It showcases the potential of employing in such platforms software-based implementations of wireless physical layers.

In the realm of wireless communications research different networks are explored, with cutting-edge communication technologies and significant antenna advancements. To facilitate the development and assessment of these technologies, creating extensive fading channels within a laboratory setting using hardware, serves a practical alternative to expensive field-testing [20]. In recent years, the literature has seen a range of hardware approaches (FPGA-based) for simulating fading channels, with some of them being proposed in papers [21, 22].

## 1.2 Thesis Objectives

The target of this thesis was utilize the available software optimizations techniques in an ARM embedded platform/SoC for applications in the field of telecommunications. Initially a soft-demodulation algorithm's performance for digital modulations was enhanced using SIMD logic in ARM processors, following up with the application of Approximate Computing techniques over the original algorithm. Furthermore, in order to have as real-time as feasible channel modeling, an FPGA-based Fading Channel Emulator was implemented, performing at a high frequency with highly accurate output samples.

## 1.3 Thesis Outline

The main part of this thesis is organized in total of 6 Chapters, which after this introductory chapter the structure is as follows:

1. *Theoretical Background*
   Chapter 2 presents a brief mandatory theoretical background for Digital communications and channel modeling. Furthermore a brief introduction is given for the FPGA-SoC, the platform that was used in during the development.

2. *SIMD Demodulation with Approximation Techniques*
   Chapter 3 firstly presents the implementation of a QAM soft-demodulation algorithm on NEON using SIMD commands. Then an approximation approach over the original algorithm is proposed, in order to decrease the required arithmetic operations.

3. *Fading Channel Emulation on FPGA*
   Chapter 4 gives a detailed description of the implementation of all the necessary blocks used in the Fading Channel Emulator on FPGA.

4. *Experimental Results*
   Chapter 5 presents the experimental results for both works. For the SIMD

Demodulation, the execution time as well as the system's Bit Error Rate (BER) is evaluated. Regarding the FPGA Fading Channel emulator, the quality of the Fading effect is analyzed, by comparing with a floating model the Probability Density Function (PDF) and the constellation plot.

5. *Conclusions and Future Work*
   Chapter 6 shows conclusions from the aforementioned experimental results, as well as some future work suggestions for both the SIMD Demodulation and the FPGA Fading Channel Emulator.

# Chapter 2

# Theoretical background

## 2.1 Introduction

In this chapter we will examine the fundamental blocks of a digital communication system, as well as the theory behind Demodulation Algorithms. In addition, modeling techniques for different transmission channels are presented. A brief overview of the SoC-FPGA architecture is presented; the platform of choice during the development of the thesis.

## 2.2 Digital Communication

A fundamental definition of digital communication involves transmitting a message from a defined set of binary digits (bits) or symbols, during a specific time interval. Each symbol or bit is associated with a continuous time waveform, which is then transmitted over the channel. Within any finite period, the waveform at the channel's output belongs to a limited set of potential waveforms, contrary to analog communications, where the channel's output can be any possible waveform. Digital Communications while not being without any trade-offs, offer multiple advantages regarding Design Efficiency, Hardware Flexibility, QoS (Quality of Service), improved security features etc. In a communication system (analog or digital), there are two fundamental resources to take into consideration: transmitted power and channel bandwidth. Moreover, Transmitted Power refers to the average power of the signal being sent out, while the channel bandwidth represents the range of significant frequency components reserved for transferring the input signal [23].

### 2.2.1 Digital Communication Chain

A typical digital communication system is shown in Fig. 2.1, which can be divided into three main parts, regardless of the particular application and configuration; the Transmitter (TX), the channel and the Receiver (RX) [11].

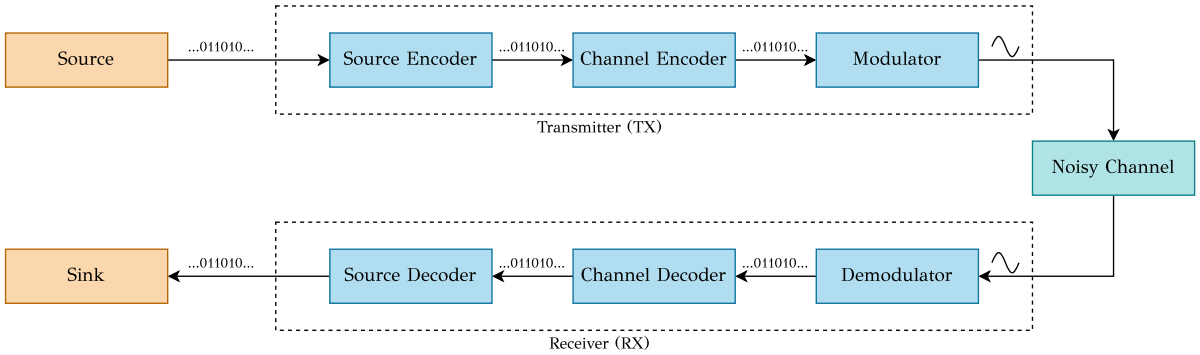



**Figure 2.1**: Digital Telecommunication Chain

In more detail, the information source generates its output, and then an input transducer like a microphone, transforms the source output into a time-varying electrical signal known as the message signal. Then the transmitter is responsible for converting the message into a form suitable for transmission over a physical channel (ex. a cable), where the source encoder's role involves eliminating redundant information from the message, in order to to optimize channel utilization. The Modulation Module takes a series of bits of the Channel Encoder and translates each sequence into a waveform, suitable for transmission.

The physical medium between the transmitter and receiver module is called communication channel, in which the transmitted message is degraded, since no ideal channel exist. Various forms of modifications in the signal can happen, including attenuation, noise, distortion and inference phenomenons.

The Receiver (RX) module of a telecommunication system, is responsible for extracting the received message signal, by performing demodulation, along with some complementary functions like filtering and amplification, since the effects of signal degradation must be encountered. The Demodulation handles the received waveforms and matches each one with the respective bit sequence.

### 2.2.2 Digital Modulation

In digital communications the coded pulses used for the transmission of information signals play an important role, which is essentially the conversion of analog waveforms into coded pulses. In digital modulation, the message signal is expressed with discrete values in both time and amplitude, enabling the transmission of the message as a digital sequence of coded pulses [24]. Some fundamental different Modulation techniques are presented bellow.

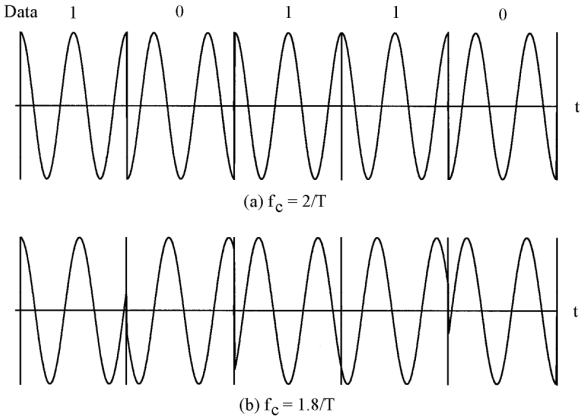



**Figure 2.2**: Example of BPSK waveform [1]

### Phase-Shift Keying (PSK)

Phase shift keying (PSK) Modulation encodes all the data within the phase of the carrier signal and is now used in both military and commercial communications. A series of information symbols form the set $1, 2, \ldots, M$ are transmitted from the M-PSK Modulator. The general analytic expression for PSK is defined in Eq. 2.1, where $\phi_i(t) = {}^{2\pi i}/_{M}$ is defined as the phase term and contains $M$ discrete values.

$$s_i(t) = \sqrt{\frac{2E}{T}} \cos[\omega_0 t + \phi_i(t)], \quad 0 \leq t \leq T, i = 1, 2, \ldots, M \tag{2.1}$$

Each symbol transmitted conveys $k$ bits of information, where $k = \log_2 M$, while the M-PSK mapping is used. The $M$ variable indicates the modulation order and determines the quantity of constellation points in the reference constellation. For example if 3-bits of information are contained in a single transmit symbol, a 8-PSK Modulation scheme is utilized. Furthermore, Binary Phase Shift Keying (BPSK) and Quadrature Phase Shift Keying (QPSK) Modulation is used, for 2 and 4 bits respectively.

### Frequency Shift-Keying (FSK)

In a typical Frequency Shift-Key Modulation, the waveform depicts a slight shift from one frequency to another. The general analytic expression is given in Eq. 2.2, where $\omega_i$ is the frequency term and consists of $M$ discrete values, while in contrast to the PSK Modulation scheme, the phase term $\phi$ is a fixed parameter.

$$s_i(t) = \sqrt{\frac{2E}{T}} \cos[\omega_i t + \phi], \quad 0 \leq t \leq T, i = 1, 2, \ldots, M \tag{2.2}$$

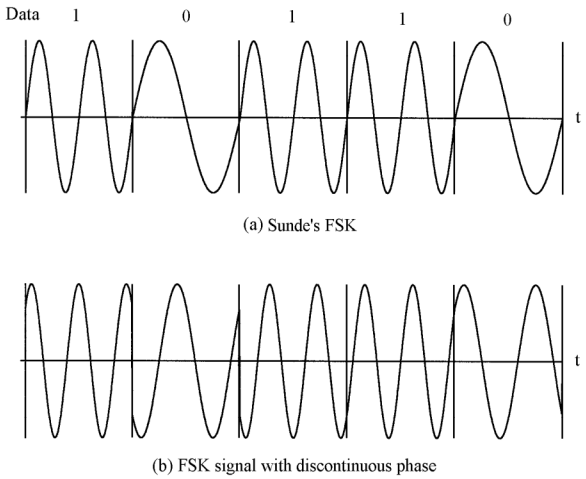



**Figure 2.3**: Examples of different FSK Modulations [1]

### Pulse-Amplitude Modulation (PAM)

Pulse-Amplitude Modulation (PAM) refers as to the most fundamental and straightforward form of modulation. It involves linear modulation, where the amplitudes of evenly spaces pulses are adjusted in relation to the respective sample values of a continuous message signal. These pulses can be formed in either a rectangular or another appropriate shape. During the creation of the PAM signal, the message signal's sampling happens at regular intervals, while taking into consideration the sampling theorem. Meanwhile, the duration of each acquired sampled is extended to a consistent value denoted as $T$. A PAM signal can be expressed as a discrete convolution sum, as denoted in Eq. 2.3 where $T_s$ represents the sampling period and $h(t)$ is a Fourier-transform pulse.

$$s(t) = \sum_{n=-\infty}^{\infty} m(nT_s)h(t - nT_s) \tag{2.3}$$

### Quadrature Amplitude Modulation (QAM)

One of the many different digital modulation techniques is called Quadrature Amplitude Modulation (QAM) or Quadrature Multiplexing, which operates by transmitting two DSB signals using carries of the same frequency but in phase quadrature. M-ary QAM extends the concept of M-ary PAM into two dimensions by utilizing two orthogonal passband basis functions, as denoted in Eq. 2.4. A typical block diagram of a QAM Modulator is illustrated in Fig. 2.4, where $p(t)$ is a smooth pulse defined on $[0, T]$.

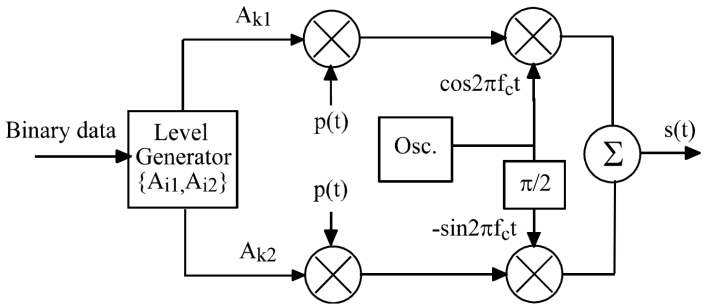
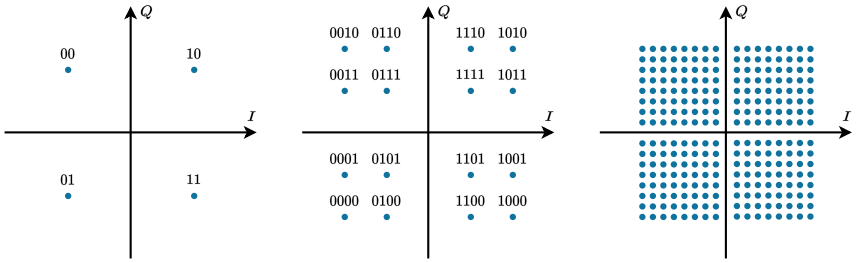



**Figure 2.4**: QAM Modulator [1]




**(i)** QAM4 ($2^2$ bits) **(ii)** QAM16 ($2^4$ bits) **(iii)** QAM256 ($2^8$ bits)

**Figure 2.5**: QAM constellations for different transmission bits

$$\begin{aligned} \phi_1(t) &= \sqrt{\frac{2}{T}}\cos(2\pi f_c t), \quad 0 \leq t \leq T \\ \phi_2(t) &= \sqrt{\frac{2}{T}}\sin(2\pi f_c t), \quad 0 \leq t \leq T \end{aligned} \tag{2.4}$$

This way by combining the amplitude and phase, the Transmitter can send more bits per symbol further. For example some popular variations like QAM4, QAM16, QAM246 etc. transmit as shown in Fig. 2.5 2, 4 and 8 bits respectively. Moreover, certain QAM constellations that require odd number of bits (such as QAM512), are depicted using a cross-like configuration instead of a square arrangement [12].

For each QAM constellation, a Gray Code is utilized for the mapping, employing a specific bit-per-symbol arrangement, where consecutive values differ in only a single bit. This technique is extensively used, in order to reduce errors in demodulation due to added noise.

Rectangular QAM signal constellations offer a notable advantage, as they can be effortlessly created by incorporating two PAM signals onto phase-quadrature carriers, while their demodulation procedure is straightforward. Despite not being the optimal choice for M-ary QAM signal constellations when $M \geq 16$, they exhibit only a slight increase in the average transmitted power needed to achieve a specified minimum distance compared to the best M-ary QAM signal constellation. Due to

these factors, rectangular M-ary QAM signals find widespread practical use [25].

### 2.2.3 Demodulation Algorithms

As we've already seen, in a digital communication system carrying information involves choosing a signal from a predefined set for transmission. The signal received at the other end is inevitably distorted and affected by noise. Consequently, a key challenge in designing the receiver is determining, from the received signal, the specific signal that was originally transmitted from the source. The objective of the link designer is to minimize the probability of error in this decision, adhering to the constraints imposed by the system [2].

#### Soft Demodulation

In contrast to a Hard-Demodulation method, where the Demodulator maps the received signal to the closest symbol in the constellation, Soft-Demodulation Method uses a probability metric, indicating the likelihood that each symbol that transmitted was originally "1" or "0". This method, although having a higher computational complexity than using a hard decision algorithm, it has better performance in terms of Bit Error Rate (BER), while offering numerical stability with a reduced dynamic range. The Log-Likelihood Ration (LLR), represents the logarithm of the ratio of probabilities of a "0" bit being transmitted, versus a "1" bit being transmitted for a received signal $r$. For a bit $b$, the LLR is defined as follows:

$$LLR_b = \log\left[\frac{\Pr(b=0|r)}{\Pr(b=1|r)}\right] \tag{2.5}$$

#### Exact Log-Likelihood Ratio

By assuming all received symbols have equal probability we can express the Exact LLR ($ELLR$) of an AWGN channel with Eq. 2.6, where $\sigma^2$ is the noise variance of baseband signal, $s_x$ represents the In-phase coordinate ($I$) of the ideal symbol or constellation point, while $s_y$ corresponds to the Quadrature coordinate ($Q$) of the same ideal symbol or constellation point [13].

$$ELLR_b = \ln\left\{\frac{\sum_{s\in S_0} e^{-\frac{1}{\sigma^2}[(x-s_x)^2+(y-s_y)^2]}}{\sum_{s\in S_1} e^{-\frac{1}{\sigma^2}[(x-s_x)^2+(y-s_y)^2]}}\right\} \tag{2.6}$$

#### Approximate Log-Likelihood Ratio

The computation of the Approximate Log-Likelihood Ratio (LLR) involves using only the closest constellation point to the received sample, considering "0" or "1"

at that specific bit position, as opposed to the Exact LLR Calculation, where all the constellation points are taken into consideration. Essentially, each sum in Eq. 2.6 is replaced by it's largest term, by using only the nearest constellation point that has "0" in the numerator and "1" in the denominator. As defined in [13, 14, 15], the Approximate LLR ($ALLR$) is given by Eq. 2.7:

$$ALLR_b = -\frac{1}{\sigma^2}\left\{\max_{s\in S_1}\left[(x-s_x)^2+(y-s_y)^2\right]-\max_{s\in S_0}\left[(x-s_x)^2+(y-s_y)^2\right]\right\} \tag{2.7}$$

### 2.2.4 Forward Error Correction Codes

To ensure data integrity and minimize errors, Forward Error Correction (FEC) can be employed. The primary objective of the channel encoder and decoder is to mitigate the impact of channel noise, aiming to minimize the number of errors between the channel encoder input and the channel decoder output that is delivered to the user. When implementing a predefined modulation scheme, incorporating redundancy into the coded messages necessitates a higher transmission bandwidth. Additionally, the integration of error-control coding increases the complexity of the system. Consequently, the design trade-offs associated with employing error-control coding to attain sufficient error performance, encompass the balance of bandwidth utilization and system complexity.

#### Low-Density Parity Check Codes

A type of Error Correction Code that is widely used is the Low-Density Parity Check (LDPC) Codes. These codes are defined by a parity-check matrix designated as $\mathbf{A}$, deliberately designed to have a sparse structure. This means that the code primarily comprises 0s with only a few instances of 1s. Specifically, we refer to them as $n, t_c, t_r$ LDPC codes, where $n$ represents the block length, $t_c$ signifies the number of 1s in each column of matrix $\mathbf{A}$, and $t_r$ indicates the weight of each row, with $t_r > t_c$. The rate of an LDPC code is defined in Eq. 2.8 [3].

$$r = 1 - \frac{t_c}{t_r} \tag{2.8}$$

In matrix $\mathbf{A}$ we consider that $n-k$ represents the number of rows and $n$ is the number of columns. Hence, dividing $t_c$ by $t_r$ we get the Eq. 2.9, where $k/n$ is essentially the block code rate. For this statement to hold true, it is necessary for the rows in matrix $\mathbf{A}$ to be linearly independent.

$$\frac{t_c}{t_r} = 1 - \frac{k}{n} \tag{2.9}$$

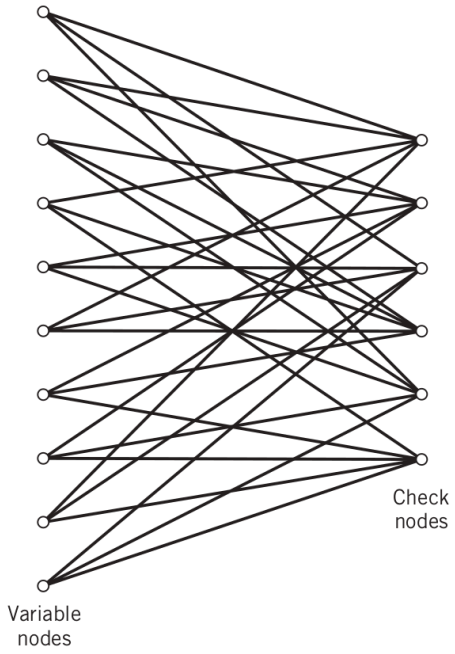



**Figure 2.6**: LDPC code example graph

The composition of LDPC codes is effectively illustrated by bipartite graphs, originally introduced by Tanner in 1981, and consequently referred to as Tanner graphs. Fig. 2.6 illustrates an example for a graph of $n = 10$, $t_c = 3$ and $t_r = 5$. The nodes on the left represent variable nodes, corresponding to elements of the codeword. On the right, check nodes are depicted, representing the set of parity-check constraints satisfied by codewords in the code. The graph showcases LDPC codes categorized as regular, where nodes of the same kind possess identical degrees. With the block length approaching infinity, the proportion of variable node connected to each check node diminishes significantly, resulting in what is referred to as ”low density” scenario.

### 2.2.5 Interleaver

To fully leverage the advantages of FEC coding in wireless communications, the incorporation of an additional technique called interleaving is crucial, since wireless channels possess memory, due to multipath fading caused by signals reaching the receiver through various propagation paths of various lengths. Fast Fading specifically, a phenomenon that arises from signal reflections off nearby objects near the transmitter, the receiver or both. Interleaving servers to effectively disperse any error bursts that might occur during data transmission over the wireless channel, distributing them across the duration of the Interleaver’s operation. This process substantially enhances the probability of receiving a correctable sequence. During

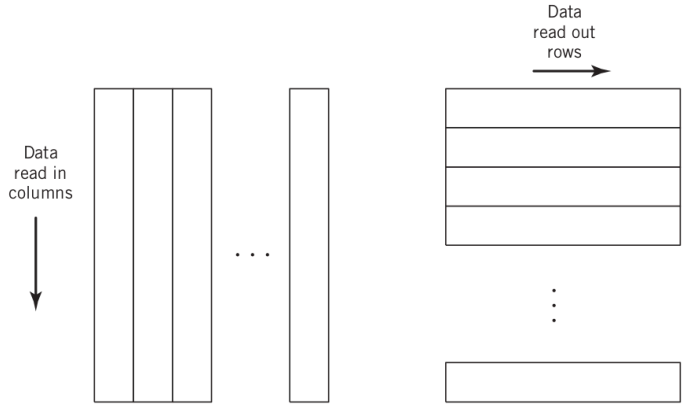



**Figure 2.7**: Structure of a Block Interleaver

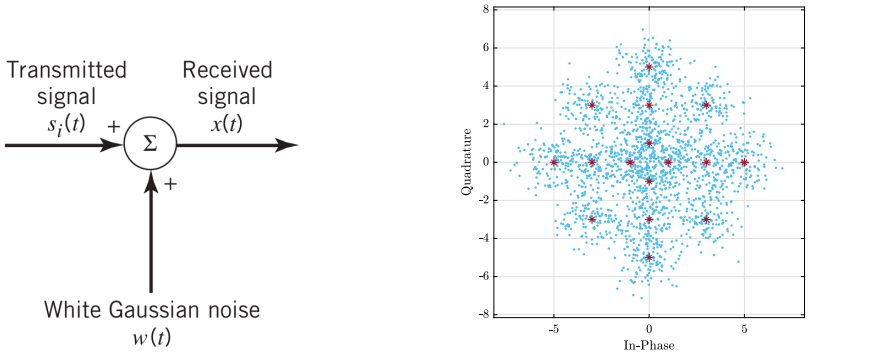
the transmission phase, the Interleaver is positioned after the channel encoder, while in the reception phase, the de-interleaver is positioned prior to the channel decoder.

Fig. 2.7 illustrates a classic Interleaver model, known as Block Interleaver, that functions as a memory buffer. Input data is written into the rectangular array in column fashion. After the array is full, it's contents are read sequentially in a row-wise manner and delivered to the transmitter. The opposite process is executed at the receiver.

## 2.3 Channel Modeling

### 2.3.1 Additive White Gaussian Noise

Additive White Gaussian Noise (AWGN) serves as a straightforward noise model, representing the movement of electors within the RF front end of a receiver. As implied by its name, the noise is added to the signal, while it's called "white" because the noise maintains a uniform spectral distribution across the sampling bandwidth.




**(i)** Channel model [3] **(ii)** Effect over General QAM Modulation

**Figure 2.8**: AWGN Channel and effects

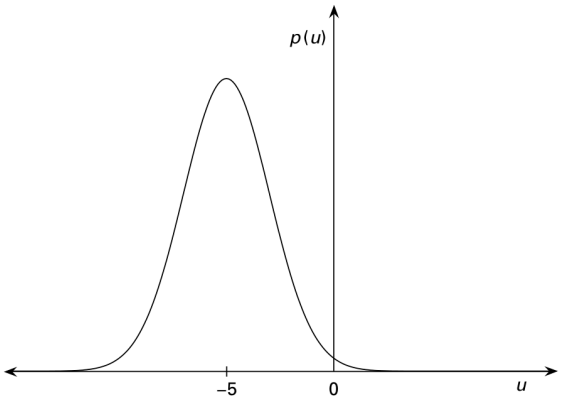



**Figure 2.9**: PDF of Gaussian Variable with $\mathcal{N}(-5, 4)$ density [2]

It's termed Gaussian due to its amplitude conforming to a normal probability distribution. Precisely, a random variable $X$ is following a Normal or Gaussian distribution, if its density follows the Eq. 2.10., where $m$ is the mean value of $X$ and $v^2$ represents the variance of $X$.

$$p(x) = \frac{1}{\sqrt{2\pi v^2}} \exp\left[-\frac{(x-m)^2}{2v^2}\right] \tag{2.10}$$

For an AWGN channel model, the received signal $x(t)$ is defined by Eq. 2.11, where $w(t)$ is the channel noise and $s_i(t)$ the transmitted signal. The receiver is responsible for observing the received signal for a duration of $T_b$ seconds, and subsequently forming an approximation of the original transmitted signal $s_i(t)$.

$$x(t) = s_i(t) + w(t),\ 0 \leq t \leq T_b \tag{2.11}$$

The AWGN channel is commonly employed to emulate a satellite communications channel, as such channels generally avoid typical terrestrial impairments like fading, multipath and interference. Utilizing an AWGN channel provides a solid initial basis for analyzing terrestrial wireless links, as it sets a favorable benchmark for the Bit-Error-Rate (BER) performance of these links.

For a transmitted signal over a AWGN channel, we can theoretically estimate the BER, using a set of predefined equations. Moreover, to determine the probability of error for a M-QAM Modulation, as stated in [26] the Eq. 2.12 can be used, where $k = \log_2 M$ and it's an even number.

$$P = 4\frac{\sqrt{M}-1}{\sqrt{M}} Q\left(\sqrt{\frac{3kE_b}{(M-1)N_0}}\right) - 4\left(\frac{\sqrt{M}-1}{\sqrt{M}}\right)^2 Q^2\left(\sqrt{\frac{3kE_b}{(M-1)N_0}}\right) \tag{2.12}$$

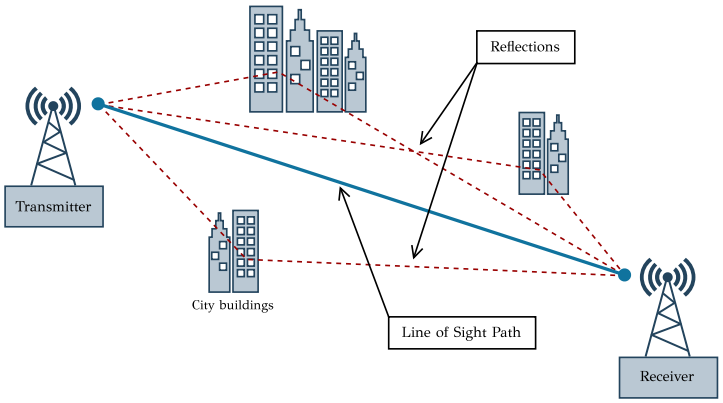



**Figure 2.10**: Illustration of fading channel in urban environment

### 2.3.2 Fading Channel

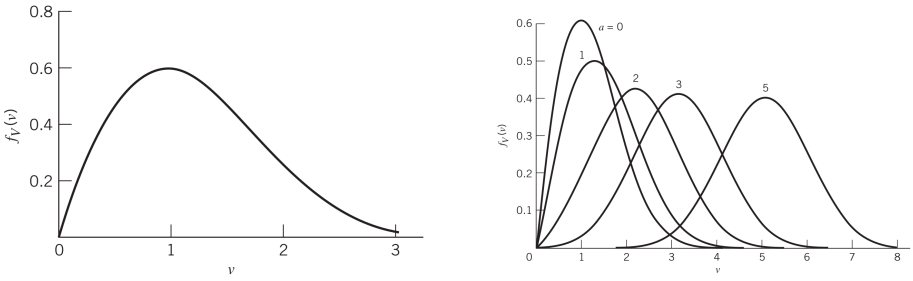
Fading and Multipath effects are commonly encountered in radio communication systems. In wireless communication setups, various signal paths can exist between transmitter and receiver antennas. These multiple paths emerge from atmospheric scattering, refraction, or reflections from objects like buildings, as illustrated in Fig. 2.10. In cases of multipath, signals arriving via different paths exhibit distinct attenuation and delays, which can either enhance or diminish each other at the receiving antenna. To comprehend the characteristics of the multipath phenomenon, we can initially examine a static, multipath scenario; a stationary receiver interacts with a transmitted signal comprising a narrowband signal, such as an unmodulated sinusoidal carrier, while we make the assumption that two attenuated copies of the transmitted signal reach the receiver sequentially. The consequence of the time delay disparity is the introduction of a relative phase shift between any two components of the received signal.

A basic representation for discrete multipath channels is shown in Eq. 2.13, where $s(t)$ represents the bandpass input signal, $a_n(t)$ is the attenuation factor for the signal received through the nth path, and $\tau_n(t)$ represents the associated propagation delay. Hence, the multipath fading channel is implemented as a linear finite impulse-response (FIR) filter [27].

$$y(t) = \sum_n a_n(t)s(t - \tau_n(t)) \tag{2.13}$$

When path lengths or geometry change due to factors such as transmission medium alterations or antenna movement, signal strength can undergo unpredictable fluctuations. Based on signal path, the fading distribution can be divided into two categories:

1. *Rician*, where the transmitter has direct Line-of-Sight (LoS) path to the receiver.




**(i)** Rayleigh **(ii)** Rician

**Figure 2.11**: Normalized Distributions PDFs [3]

2. *Rayleigh*, where one or more major reflected paths from transmitter to receiver.

In general, a probability distribution is a theoretical model that is based on assumptions about a source population. Probabilities are allocated through these model to the occurrence of a random variable. A Rician Distribution is described from the Eq. 2.14, where the first kind zero-order modified Bessel function is denoted as $I_0$. The non centrality parameter $s$ is non zero, while for $x > 0$ the scale parameter $\sigma > 0$.

$$p(x) = I_0\left(\frac{xs}{\sigma^2}\right)\frac{x}{\sigma^2}\exp\left(-\frac{x^2+s^2}{2\sigma^2}\right) \tag{2.14}$$

Utilizing the Central Limit Theorem, the real and imaginary parts of the complex faded signal can be represented as zero-mean Gaussian random variables, denoted as $\mathcal{N}(0,\sigma^2)$, with mean value $\mu = 0$ and variance $\sigma^2$. The probability Density Function (PDF) of the Rayleigh distribution is shown in Fig. 2.11 and follows the Eq. 2.15.

$$p(x) = \frac{x}{b^2}\exp\left(-\frac{x^2}{2b^2}\right) \tag{2.15}$$

## 2.4 SoC FPGA

System-on-Chip Field Programmable Gate Arrays (SoC FPGAs) consist of semiconductor devices that combine programmable logic (PL) and embedded processor cores, typically sources from companies like ARM. This design merges the convenience of programming a processor with the adaptability and efficiency of a programmable logic structure.

### 2.4.1 FPGAs

Field Programmable Gate Arrays are flexible hardware reconfigurable hardware chips, designed for digital logic. FPGAs consist of an arrangement of logic gates that

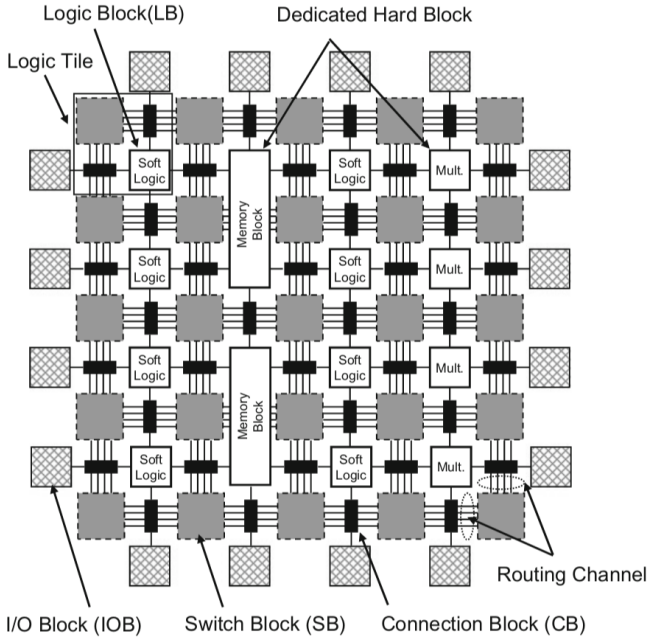



**Figure 2.12**: Overview of an Island-style FPGA [4]

can be programmed to create custom digital circuits. These circuits can be defined using Hardware Description Languages (HDL), like Verilog or VHDL [28].

An FPGA comprises three fundamental elements: a logic element that can be programmed, an I/O element that can be customized for external connections, and an interconnect element that can be programmed to link various sections together. Additionally there are Digital Signal Processing (DSP) units and integrated memory, in order to enhance computational capacity. By transferring the data to these components, an FPGA can execute the intended digital circuit. The structure of an island-style FPGA is illustrated in Fig. 2.12, which comprises logic elements (logic block), peripheral I/O elements (I/O block), routing elements (switch block, connection block and routing channel), embedded memory and multiplier blocks. A collection of adjacent logic blocks along with a connection and switch block, is refereed to as a logic tile [4].

In the initial stages of FPGA development, LUTs were the only components comprising a logic block. Nowadays however, FPGAs there are fundamental logic elements known as BLEs, which contain a LUT, a flip-flop (FF) and a selector. Depending on a memory bit dedicated for configuration, the selector controls whether the value of the LUT or the one stored in the FF is emitted. When designing the architecture of a logic block, there are various trade-offs between area, efficiency and delay.

In order to enhance arithmetic operation performance, modern FPGAs incorporate a specialized carry logic circuit within the logic block. While arithmetic operations can technically be executed using LUTs, employing dedicated carry logic

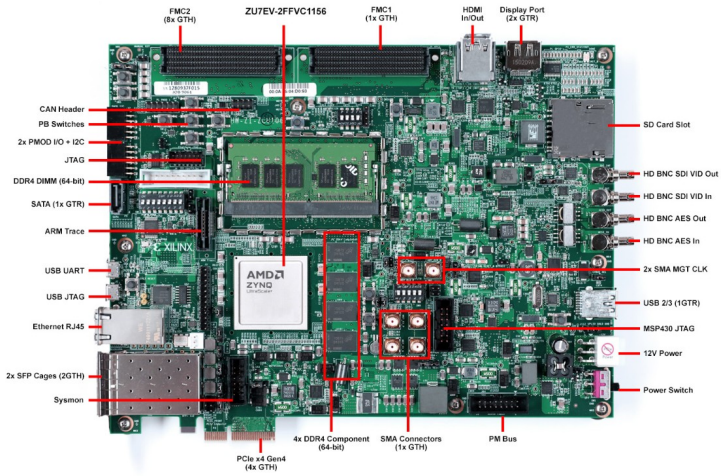



**Figure 2.13**: Xilinx Zynq UltraScale+ ZCU106 Evaluation board

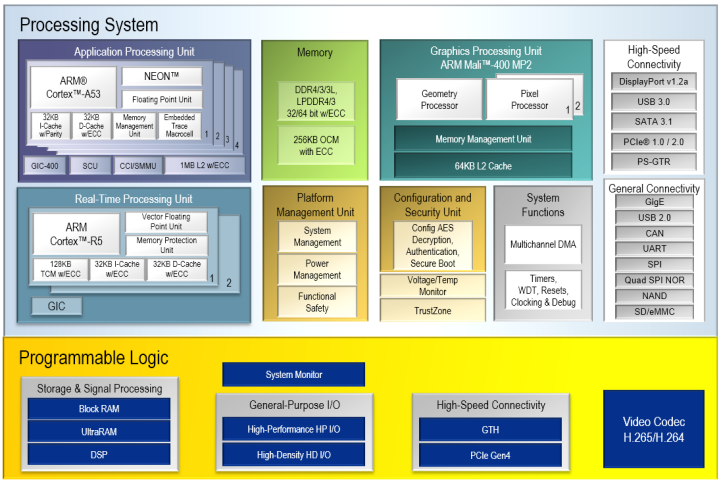



**Figure 2.14**: Zynq UltraScale+ MPSoC Block Diagram [5]

proves more efficient in terms of both integration level and operational speed.

### 2.4.2 Zynq UltraScale+ MPSoC

In this thesis we used the ZCU106, a general purpose evaluation board for rapid prototyping from Xilinx, packaged in the 16nm FinFET Zynq UltraScale+ MPSoC. The Zynq UltraScale+ MPSoC is a diverse SoC that integrates multiple processing engines, a variety of high-speed peripherals, advances I/O capabilities and a PL component. This SoC consists of a quad-core ARM Cortex A53-based APU, a dual-core ARM Cortex R5-based RPU, a Mali graphics processing unit, a platform management unit, and a video codec unit (VCU). Additionally, it features power islands that allow turning specific blocks on and off to conserve power. The generic block diagram of the Zynq's UltraScale+ MPSoC architecture of the PS and PL, is shown in Fig. 2.14 [5].

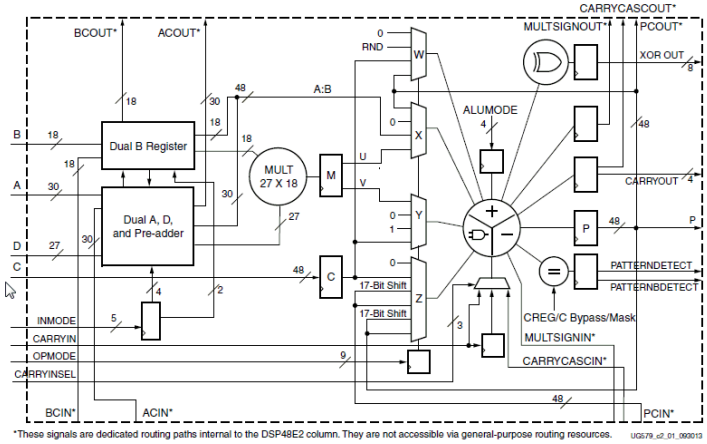



**Figure 2.15**: Zynq UltraScale+ DSP48E2 Slice [6]

### DSP Blocks

In DSP applications, programmable logic devices have been proved to be highly effective, due to their capability to execute tailored, fully parallel algorithms. Within DSP applications, numerous binary multipliers and accumulators are utilized and with dedicating specific DSP resources for these functions, leads to optimal implementation [29]. The UltraScale+ devices contain numerous specialized, low-power DSP slices, offering a blend of high speed and compact size. The resources significantly boost the speed and efficiency of various applications beyond DSP, spanning from wide dynamic bus shifters, memory address generators, wide bus multiplexers and memory-mapped I/O registers. The fundamental operation of the DSP48E2 slice contained in the UltraScale+ devices is depicted in Fig. 2.15.

The DSP48E2 slice provides versatile functionality, encompassing various independent operations, such as multiplication, multiply-accumulate, multiply-add, four-input addition, barrel shifting, wide-bus multiplexing, magnitude comparison, bitwise logic operations, wide XOR, pattern detection and wide counter. Additionally, this design allows for the integration of multiple DSP28E2 slices, in order to create DSP Filter, extensive math function and complex arithmetic operations, without the necessity of additional logic units.

## 2.4.3 ARM Cortex-A53 Processor

The Cortex-A53 processor falls within the mid-range category, offering low-power consumption and the capability of handling both 32-bit and 64-bit code. It features one to four cores in a single cluster, each equipped with its own L1 cache subsystem. There's also an optional integrated GICv3/4 interface an optional L2 cache controller. The Cortex-A53 processor features an in-order, eight stage execution pipeline, reduced power consumption through hierarchical clock gating, power

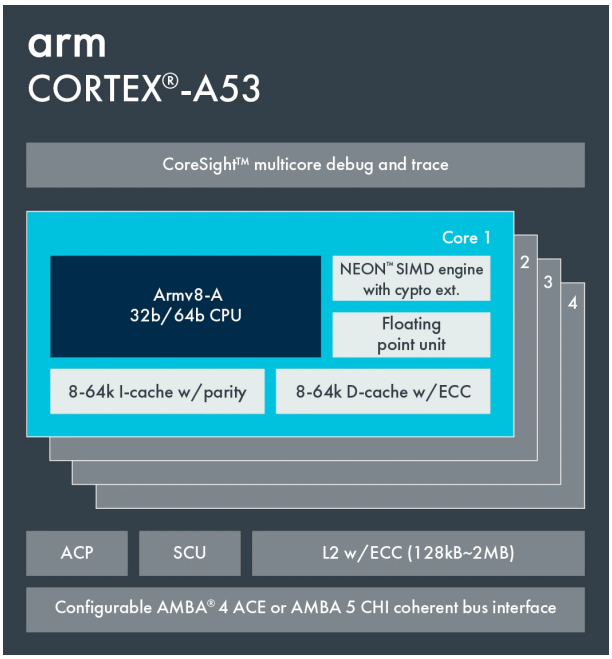



**Figure 2.16**: Cortex-A53 processor block diagram [7]

domains and advanced retention modes. Also it has enhanced dual-issue capability, from incorporating dual instruction decoders and execution resources [30]. Fig. 2.16 illustrates the basic block diagram of the Cortex-A53.

The Armv8-A architecture introduces the potential of utilizing two distinct execution modes: AArch64 which is 64-bit and AArch32, which is 32-bit. Within the AArch64 Execution state, the A64 instruction set is enabled, employing 64-bit registers to handle addresses and permitting instructions in the base Instruction Set, in order to utilize these 64-bit registers for computation. Conversely, the AArch32 Execution state represents a 32-bit execution mode aimed at preserving compatibility with the prior Armv7-A architecture. It extends this profile to encompass certain features from the AArch64. Furthermore, AArch32 supports both the T32 and A32 Instruction sets [31].

### Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) instructions, enhance the overall effectiveness of the intended application, by enabling a single instruction to simultaneously handle multiple sets of data, through purpose-built hardware logic. Fig. 2.17 shows the difference between SISD and SIMD processor architecture.

### NEON Engine

ARMv8 processors along with the general purpose registers, they populate 32 128-bit SIMD registers on each core, that are used to implement NEON instructions.

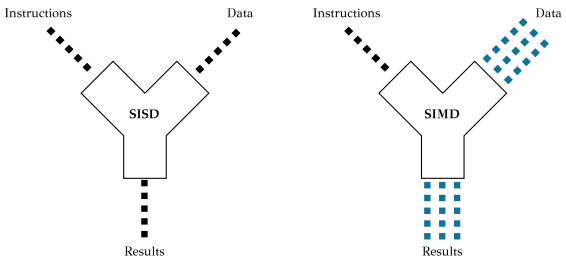



**Figure 2.17**: SISD versus SIMD architecture

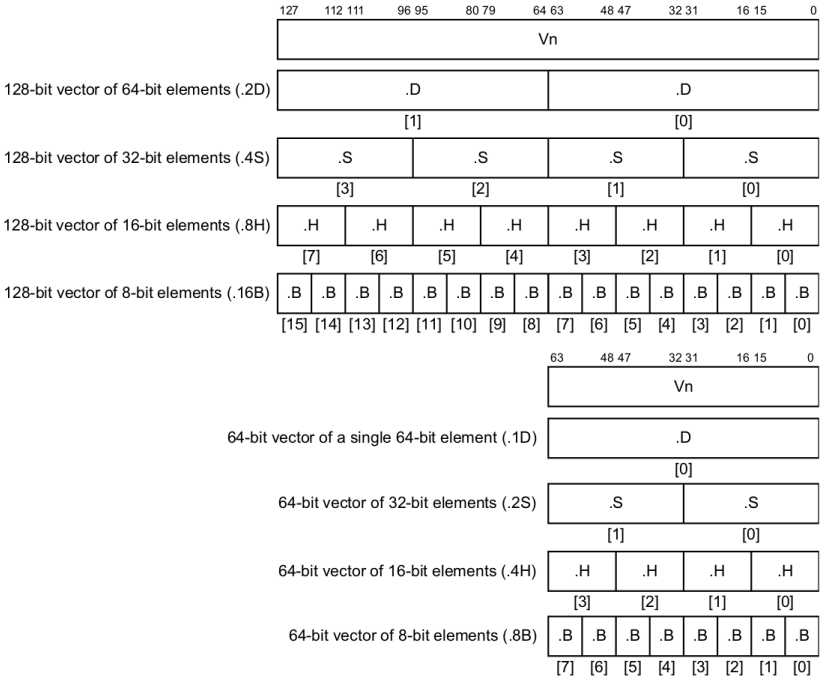



**Figure 2.18**: NEON SIMD vectors in AArch64 [8]

The NEON instructions support 8-bit, 16-bit, 32-bit and 64-bit signed and unsigned integers, 32-bit single-precision floating point elements and 8-bit, 16-bit polynomials. NEON technology offers a specialized expansion to the ARM Instruction Set Architecture, introducing extra instructions capable of executing data processing operations concurrently across numerous data streams. In Fig. 2.18 the possible scenarios for splitting the NEON registers are illustrated, in order to fit the desired data widths of vector elements [32].

### NEON Intrinsics

NEON Intrinsics refer to functions with well-defined implementations known to a compiler. Specifically NEON Intrinsics encompass a collection of C/C++ functions outlined in `arm_neon.h` header file, compatible with Arm compilers and GCC. These functions enable the utilization of NEON functionality, without the necessity

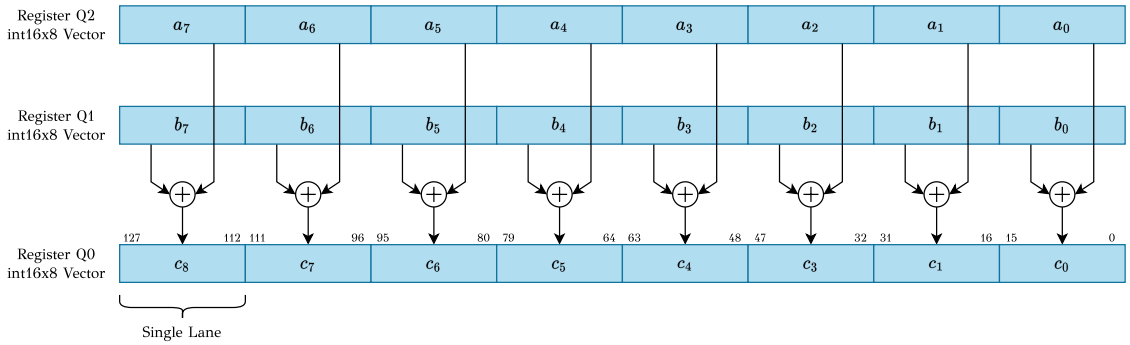



**Figure 2.19**: Example of NEON intrinsic addition instruction

of directly coding in Assembly. They encapsulate short assembly kernels, inlined into the calling code. Furthermore, the compiler efficiently managers register allocation and pipeline optimization, alleviating challenges commonly encountered by assembly programmers.

Fig. 2.19 illustrates how a NEON intrinsic accomplishes a simultaneous addition of eight lanes comprising 16-bit elements extracted from Q1 and Q2, while storing the outcome in Q0.

# Chapter 3

# SIMD Demodulation with Approximation Techniques

## 3.1 Introduction

In this chapter the implementation of the Demodulation block for QAM Modulation will be analyzed. Firstly, we're going to analyze the initial implementation on the NEON Engine based on a known algorithm. Afterwards we will present an Approximation Technique for QAM Demodulation and it's implementation.

## 3.2 Approximate LLR Algorithm

The Demodulation Block depends upon Eq. 2.7, which involves calculating the Approximate LLR for individual incoming symbols within a QAM constellation. This implementation provides support to varying bits per symbol, accommodating a range of constellations such as QAM4, QAM16, QAM64, QAM256, QAM512 (Cross Constellation) and QAM1024.

The process begins with calculating the Euclidean distance between each incoming coordinate $(I, Q)$ and all conceivable symbols within the chosen constellation, accounting for both 1s and 0s. Subsequently, by finding the maximum LLR value within the relevant range, the final result is derived. It is important to emphasize that the LLR values are normalized with the first-reference symbol, ensuring a consistent span across diverse input values.

## 3.3 Loop transformations

In our initial C++ function, we applied fundamental loop transformations: loop unrolling and loop tiling. We experimented with different factors for loop unrolling and loop tiling. In both scenarios, we observed enhancement in the execution time of the Demodulation process, achieving a performance improvement ranging from 5%

**Algorithm 1** Demodulation using Approximate LLR

**function** SOFT DEMODULATION
  $a \leftarrow {}^{1}/_{2\sigma^2}$ ▷ Calculate multiplication factor
  **for all** Input Symbols **do** ▷ Block input data
    **for all** Reference QAM Symbols **do** ▷ Find all Euclidean distances
      $d_1 \leftarrow I - I_{ref}$ ▷ $I_{ref}, Q_{ref}$ reference constellation coordinates
      $d_2 \leftarrow Q - Q_{ref}$
      $d \leftarrow -(d_1^2 + d_2^2)$
      **if** i=0 **then**
        $LF_0 \leftarrow a \cdot d_0$ ▷ Normalize LLR using the first symbol
      **else**
        $LF_i \leftarrow a \cdot d_i - LF_0$ ▷ Normalize results
      **end if**
    **end for**
    **for all** QAM Symbol Bits **do** ▷ Maximum LLR value ∀ Bits
      $LLR_i \leftarrow \max_{s \in S_1}(LF_s) - \max_{s \in S_0}(LF_s)$
    **end for**
  **end for**
**end function**

to 10%. It's worth noting that similar improvements in percentages were observed when applying the same loop transformations to the SIMD Demodulator Block, which is outlined in the subsequent section.

## 3.4 SIMD Demodulation

Fig. 3.1 illustrates the calculation of Approximate LLR values using NEON Intrinsics in C++, where as it is shown, we can divide the procedure into three main parts: the data loading to NEON engine, the mathematical calculations (including the Euclidean Distances calculation and the Maximum Distance finding, and ultimately the data extraction from the NEON engine.

### 3.4.1 Data management

Through benchmarks we identified a notable portion of the execution time attributed to data transfers to and from the NEON engine. Consequently, optimizing these operations became essential. To enhance efficiency, we adopted a strategy as proposed by ARM, of loading 8 or 16 values in a single cycle.

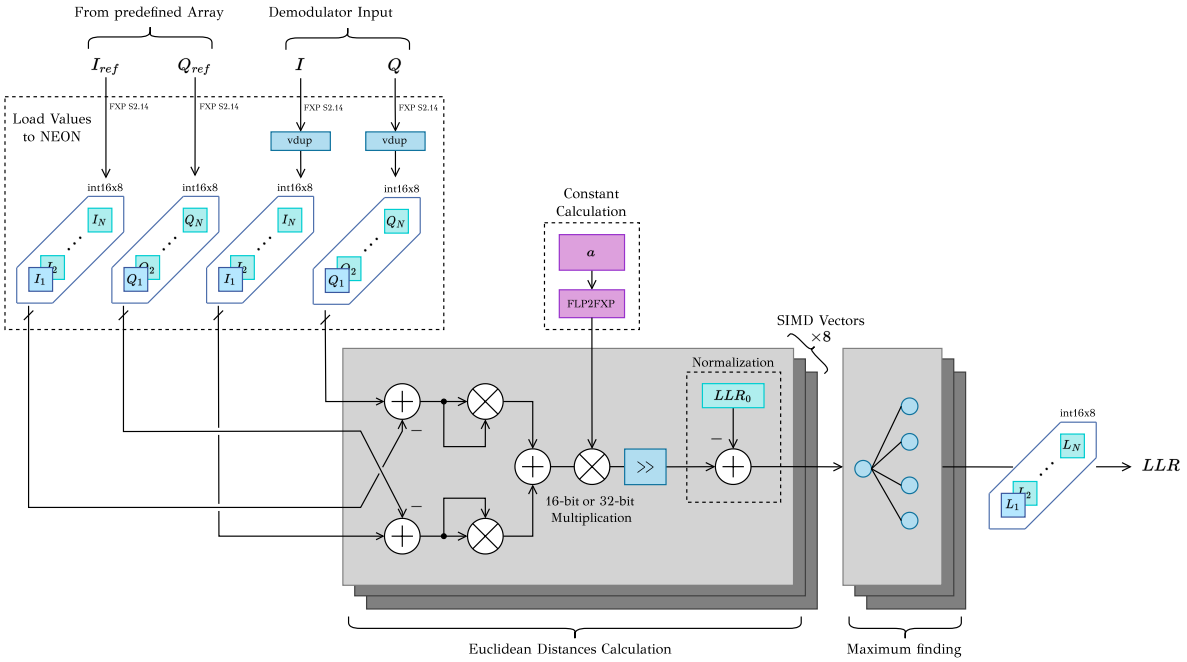



**Figure 3.1**: Block diagram of SIMD Demodulation

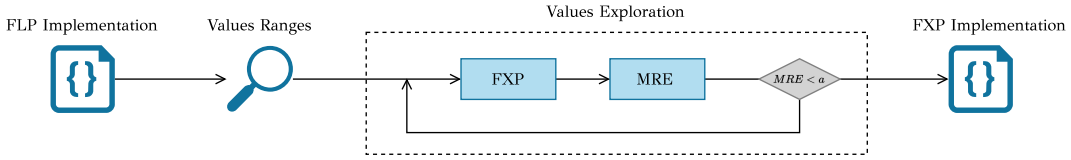



**Figure 3.2**: Workflow for the transition from floating-point to fixed-point arithmetic

### 3.4.2 Fixed point arithmetic

Although various software implementations use floating-point arithmetic, we opted for fixed-point arithmetic [33], expressing all of our data in vectors of `int16` or `int8` for 16-bit or 8-bit fixed-point number respectively. For our fixed-point representation we use the notation XA.B, where X denotes if it's signed number (S), or unsigned (U), A is the integer part with the sign bit, and finally B is the fractional part.

For the transition from floating-point arithmetic to fixed-point arithmetic, we followed the workflow illustrated in Fig. 3.2. Starting with the base model, we analyzed the size of each variable in order to find appropriate fixed-point representation. During the development in fixed-point arithmetic, we evaluated the error in comparison to the floating-point model in order to optimize our design.

For the evaluation of the precision loss due to the fixed point arithmetic, as en error evaluation metric we utilized Mean Relative Error ($MRE$), which is calculated using the floating ($a_{\text{FLP}}$) and the fixed point value ($a_{\text{FXP}}$) over a number of $N$ values,

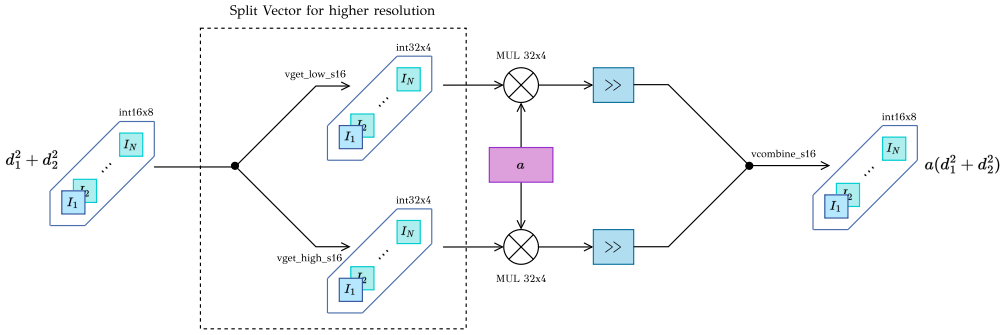



**Figure 3.3**: SIMD Multiplication of 16-bit input and 32-bit output

as stated in Eq. 3.1.

$$MRE = \frac{1}{N} \sum_{i=0}^{N} \left| \frac{a_{\text{FLP}} - a_{\text{FXP}}}{a_{\text{FLP}}} \right| \cdot 100\% \tag{3.1}$$

After data analysis of all the possible input values of the Demodulation Block, we found out that we only need 2 integer bits, thus for our input we used the formats S2.14 and S2.6 without losing accuracy. The final result of the LLR value is integer (S16.0), while it is important to keep track of the decimal position while we perform any mathematical operation. A thorough data analysis was conducted at each step to ensure the presence of sufficient integer and fractional bits and to prevent any possibility of overflow.

### Mathematical operations

We opted for a total of four implementations of the Demodulation Block, based on the input values width (`int8` and `int16`), and based on the precision of the multiplication with constant $\alpha$. The first approach, was to use `vqdmulhq_n_s16()` intrinsic which performs *Vector saturating doubling multiply high with scalar*, as referred in ARMs Manual, which conveniently handles the multiplication doubling size internally and returns a 16-bit result, same as the input. While this instruction is rather fast, the downside is that we have limited accuracy and is required to have only 6 fractional bits for $a$. Thus the second approach was to manually handle the multiplication result; assigning it to a temporary `int32` vector and then converting it back to an `int16`. The aforementioned described procedure is illustrated in Fig. 3.3. Additionally in order to gain execution time, we pre-calculated the reference constellation symbol coordinates in fixed point format and stored them in an array, rather than converting them in real time.

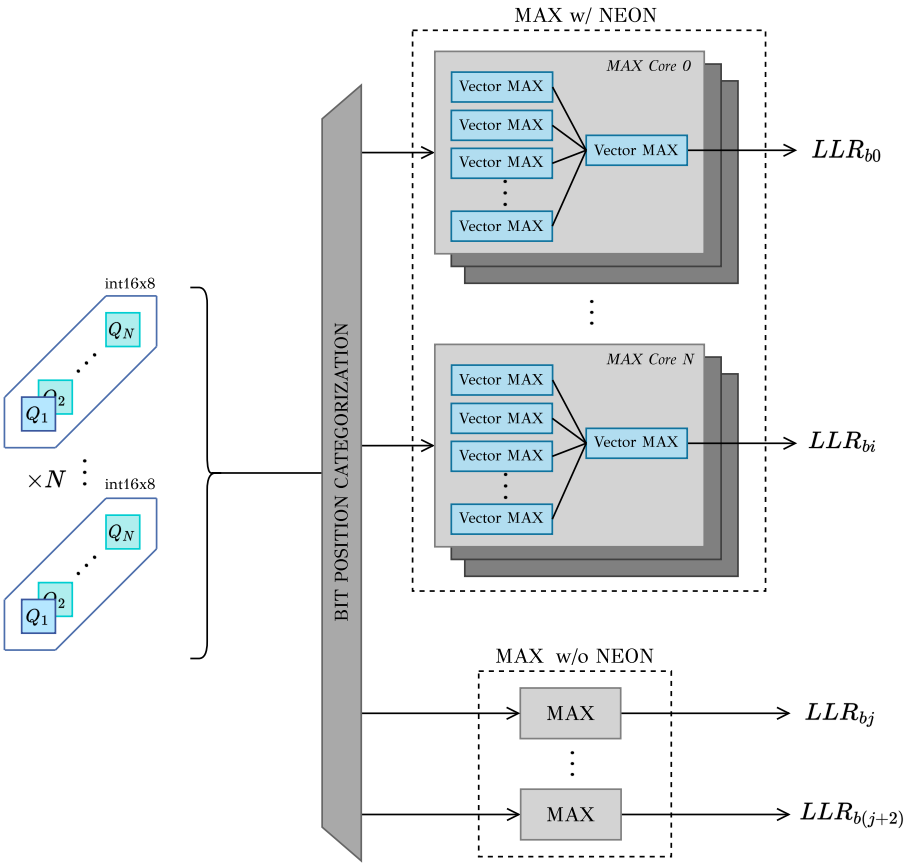



**Figure 3.4**: Architecture for computing the Maximum LLR value for different input sizes

### Finding maximum Euclidean distance

In the original implementation of the Demodulation, the LLR value of each bit doesn't need the same comparisons; it re-uses previous calculated max values, in order to optimize the required instructions. Therefore, we created a custom max finding function for calculating the maximum value each time, by taking into consideration the efficient use of the NEON engine.

When porting this algorithm to NEON using SIMD, vectorized instructions can't be utilized for each LLR computation. Therefore as illustrated in Fig. 3.4, the first values of the LLRs are calculated using NEON intrinsics, while the remaining 3 LLR calculations, since they require negligible number of comparisons, no SIMD logic is required. Furthermore with the calculated values from the $N-1$ digits, LSB bit required only a subtraction for the desired result. During the maximum finding, two different NEON intrinsics were utilized, one for finding the maximum value across a single vector and another one for calculating the maximum values between two vectors.

It's obvious to point out, that for small QAM constellations (QAM16), the NEON

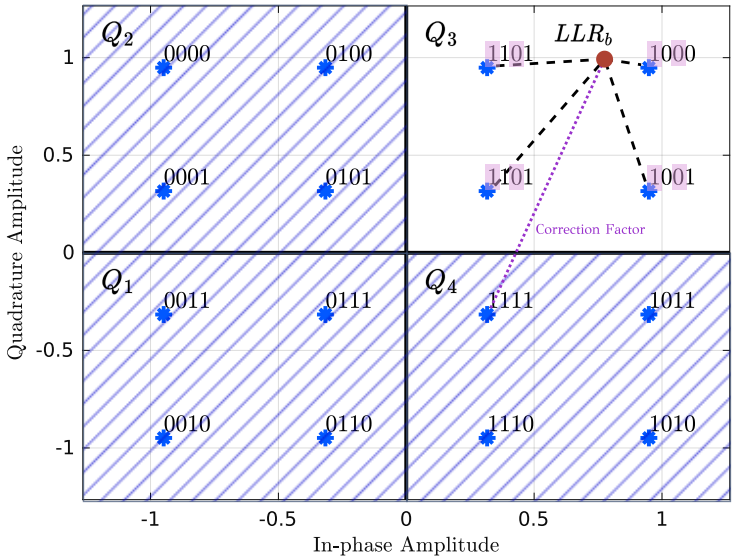



**Figure 3.5**: Soft Demodulation proposed approximation for QAM16

engine for the Demodulation algorithm is utilized only during the Euclidean Distances calculation.

## 3.5 SIMD Approximate Demodulation

Having as a base model the four previous implementations of the SIMD Demodulation, we opted for further optimizations, by applying Approximate Computing techniques [34] to the original LLR calculation in order to reduce the required arithmetic operations.

### 3.5.1 Approximation on QAM16

As we've already seen in the original Approximate LLR algorithm, for a given symbol in QAM16, 16 Euclidean distances are calculated in order to find the maximum LLR value for each bit. For our approximation technique, we propose that we only calculate the Euclidean distances on the quadrature the input symbol belongs. A similar technique of calculating less euclidean distances, but with a different grouping and for PSK/APSK modulation schemes was presented in paper [35]. The computation of each LLR value, still employs the original Approximate LLR algorithm from previous section.

As illustrated in Fig. 3.5, after calculating the Approximate LLR values for a given quadrature, knowing the Gray Mapping for our constellation, we can define the 1s and 0s in each bit position, in order to find the maximum Euclidean Distance in each case. Moreover, having a total for 4 distances, we only need to find the maximum of

2 distances for 1s and 0s respectively. Note that the quadrature selection is simply done by comparing with 0 the $I$ and $Q$ input coordinates, which we found to be the fastest solution. The aforementioned procedure in described by Eq. 3.2.

$$LLR_{approx} = \begin{cases} LLR_{s \in Quadr_3}, & \text{if } I \geq 0 \text{ and } Q \geq 0 \\ LLR_{s \in Quadr_2}, & \text{if } I \leq 0 \text{ and } Q \geq 0 \\ LLR_{s \in Quadr_4}, & \text{if } I \geq 0 \text{ and } Q \leq 0 \\ LLR_{s \in Quadr_1}, & \text{if } I \leq 0 \text{ and } Q \leq 0 \end{cases} \tag{3.2}$$

For each quadrature, we stored all the reference constellation coordinates in fixed-point format in a different array with a specific arrangement, in order to optimize the access pattern to the elements when loading the data to NEON engine. Specifically we could load to a 8 element vector both values for $I$ and $Q$ coordinates in order to calculate both distances with a single instruction.

For bits in places LSB-1 and MSB in any quadrature mapped with Gray Code, we observe that there is only 1 or 0 as a possible bit value, thus $s \notin S_0$ or $s \notin S_1$ respectively. In our preliminary attempt, we initially overlooked this issue and relied solely on the available distances. However, this approach resulted in significantly elevated LLR MRE values and a noticeably deviated BER. Therefore for these special cases, where we had to find an alternative to compute the maximum value of non-existing elements, thus we calculated two additional distances (one for each case; if there is none 1s present or if there are none 0s present), denoted as *Correction Factor* in the Eq. 3.3.

$$LLR_{approx} = LLR_{s \in Quadr_i} + \underbrace{LLR_{s \notin Quadr_i}}_{\text{Correction Factor}} \tag{3.3}$$

After conducting tests with various possible combinations, we identified that the symbol closest to the center of the axis exhibited the least LLR Relative Error. As a result, we selected this particular symbol of our calculations, although different methods could possibly be applied, for example selecting each time randomly or in a cyclic pattern the symbol, for the correction factor computation.

Furthermore, regarding the normalization of each LLR value with the reference one, the initial LLR values for each symbol was initially calculated and stored in an array using NEON. In Fig. 3.6 an abstract block diagram of the proposed Approximate Demodulation is presented. The *Maximum Value Finding* block is drawn with both colors, since as we're going to analyze below, for larger constellations (ex. QAM64) it can also be accelerated with the use of NEON Intrinsics.

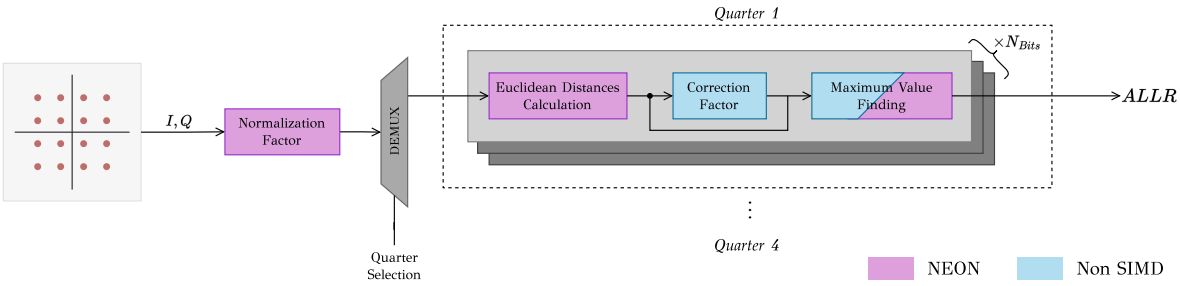



**Figure 3.6**: NEON Implementation for proposed approximated QAM demodulation

**Table 3.1**: Computational complexity of LLR Calculation per Bit

| | QAM16 | | QAM64 | |
|---|---|---|---|---|
| Operation | Original | Proposed | Original | Proposed |
| Euclidean Distances | 16 | 6 | 64 | 18 |
| Comparisons | 45 | 16 | 193 | 64 |

### 3.5.2 Extending the Approximation on QAM64

Following the same algorithmic approach with the QAM16, we extended our implementation design in order to support also QAM64 Demodulation. Specifically for the given quadrature, we calculated the 16 Euclidean distances needed using the Approximate LLR algorithm, and then found for each bit the maximum value. A different access pattern that the one in QAM16 was implemented, where there wasn't a need for combining $I$ and $Q$ coordinates.

From Table 3.1, we can observe the gain on additions, multiplications, and comparisons over the original Approximate LLR calculation and our approximate approach, for each Constellation. With our proposed method we need almost $1/4$ of each operation over the initial algorithm. For the calculation of each Euclidean Distance, a fixed number of Multiplication and Additions/Subtractions is needed; for a single computation 3 Addition/Subtraction and 3 Multiplications are required. Furthermore as we already mentioned, a small set of subtractions is also necessary during the finding of the Maximum Value.

Furthermore, we were able to utilize more the NEON engine, for calculating the maximum Euclidean distance (In QAM16 it wasn't possible due to the fact we only had to find the maximum value between two numbers). Moreover, we opted for a custom architecture for finding the max value of an array similar to the one used in Section 3.4, where the NEON engine is also utilized.

As we analyzed, we created for Approximate Demodulation different implementations for each QAM: 2 different blocks for QAM16; regarding the multiplication precision (16-bits or 32 bits), as well as 4 different blocks for QAM64; regarding the multiplication precision (16-bits or 32-bits) as well as having two input formats

**Table 3.2**: NEON Demodulator Implementations QAM support

| | Implementation† | QAM16 | QAM64 | QAM256 | QAM512 | QAM1024 |
|---|---|---|---|---|---|---|
| Original | FLP64-B64<br>FXP16-B16<br>FXP16-B32<br>FXP8-B16<br>FXP8-B32 | ✓ | ✓ | ✓ | ✓ | ✓ |
| Proposed | FXP16-A16 | ✓ | ✓ | ✗ | ✗ | ✗ |
| | FXP16-A32 | | | | | |
| | FXP8-A16 | ✗ | ✓ | ✗ | ✗ | ✗ |
| | FXP8-A32 | | | | | |

† For the implementations naming, first three letters denote the input format and next number refers input bits. Letter B is for original Approximate LLR Algorithm, while letter A is for our proposed approximation method. Final number is the precision bits used in the multiplication.

(S2.14 or S2.6). In Table 3.2 there is a complete list of all the NEON Demodulation implementations, with the according QAM constellation support.

# Chapter 4

# Fading Channel Emulation on FPGA

## 4.1 Introduction

This chapter outlines the integration on FPGA of a Fading Channel Emulator. Moreover we describe the techniques used and the custom hardware blocks used for the implementations. The chosen technique for the applying the fading effect to the transmitted symbols, is filtered Gaussian noise.

## 4.2 Model Architecture

Fig. 4.1 illustrates the overall structure of the Fading Channel implementation, where it's apparent that we can split the implementation into three main blocks. The *Gaussian Generator* is responsible for generating random Gaussian variables with real and imaginary part. The process of *Coefficients Calculation*, uses the channel properties (discrete path delays, average power gains, K-factor) computes the complex coefficients for the filter. Finally the *Complex FIR Filter*, filters the modulated input symbols with the complex channel coefficients, ultimately producing the faded signal.

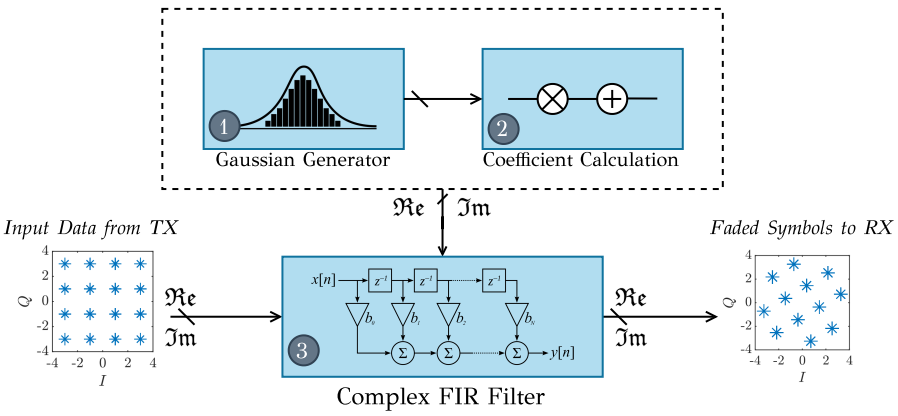



**Figure 4.1**: Fading Channel implementation block diagram

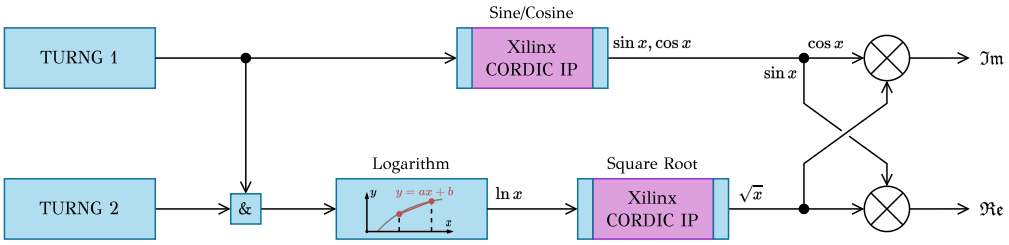



**Figure 4.2**: Box-Muller Method Architecture

## 4.3 Gaussian Random Variable Generator

Having access to random samples that follow a normal distribution is crucial in numerous computationally intensive modeling and simulation applications, with this necessity extending to evaluations of channel codes [36].

### 4.3.1 Box-Muller Transform

Most of the digital techniques for producing Gaussian random variables rely on transformations on uniform random variables. Common approaches include the Inversion method, the Wallace method, as well as our preferred method: the Box-Muller, that utilizes a sequence of evaluations of elementary functions, to convert two uniformly distributed variables into two variables with a normal distribution. In order to generate our desired samples, supposing $u_1$ and $u_2$ are independent samples on the interval $[0, 1)$, $x_0$ and $x_1$ from Eq. 4.1 and 4.2 accordingly, are independent variables of a Gaussian distribution.

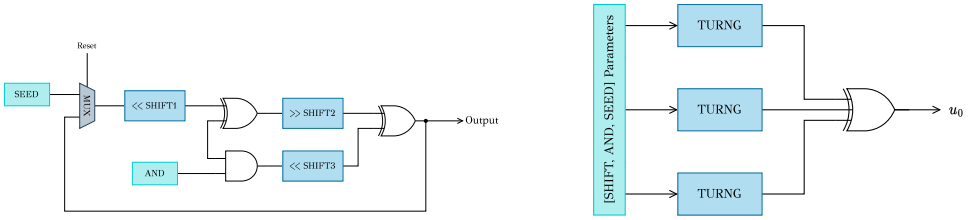
$$x_0 = \sqrt{-2\ln(u_0)}\cos(2\pi u_1) \tag{4.1}$$

$$x_1 = \sqrt{-2\ln(u_0)}\sin(2\pi u_1) \tag{4.2}$$

Fig. 4.2 illustrates the basic architecture of the hardware implementation of the Box-Muller Method, based on the equations listed above. With blocks colored blue, we refer to our custom made modules, while the purple blocks refer to Xilinx Soft IPs. For our uniform random variables in order to provide us with sufficient resolution, we chose 48-bits and 16-bits, for $u_0$ and $u_1$ respectively.

### 4.3.2 Tausworthe URNG

Each Tausworthe Uniform Number Generator (TURNG), offers remarkable randomness with a large number period of $2^{88}$. It provides a 32-bit uniform random number per clock cycle and it was conducted by applying the algorithm presented




**(i)** Single TURNG Module **(ii)** Combined 3 TURNG Modules

**Figure 4.3**: TURNG Implementation

in paper [37]. As shown in Fig. 4.3, the Algorithm 2 was implemented using XOR gates, AND gates and left or right shifts. For each module, we specified a starting seed needed for initialization, that is loaded with a MUX.

**Algorithm 2** Tausworthe URNG

**function** TURNG ▷ Variables $s_0$, $s_1$, $s_2$, $b$ are 32-bit
  $b \leftarrow ((s_1 << 13) \oplus s_1) >> 19$
  $s1 \leftarrow ((s_1 \wedge 4294967294) << 12) \oplus b$
  $b \leftarrow ((s_2 << 2) \oplus s_2) >> 25$
  $s_2 \leftarrow ((s_2 \wedge 4294967288) << 4) \oplus b$
  $b \leftarrow ((s_3 << 3) \oplus s_3) >> 11$
  $s_3 \leftarrow ((s_3 \wedge 4294967280) << 17) \oplus b$
**end function**

Each TURNG module was obtained by combining three modules, essentially combining with an XOR gate, three linear feedback shift registers (LFSRs) with different initial parameters (for the AND gate, the shift registers and the starting seed), in order to improve the statistical properties of the generator.

### 4.3.3 Sine/Cosine Implementation

For the calculation of sine and cosine functions, the CORDIC IP from Xilinx was utilized. This core implements a generalized coordinate rotational digital computer (CORDIC) algorithm as presented in [38]. A vector rotation is performed, as a sequence of successively smaller rotations, as it is shown in Eq. 4.3, were $i$ is the iteration index and $a_i = \pm 1$ is the direction of rotation.

$$\begin{aligned} x_{i+1} &= x_i - a_i y_i 2^{-i} \\ y_{i+1} &= y_i + a_i x_i 2^{-i} \\ \theta_{i+1} &= \theta_i + a_i \arctan(2^{-i}) \end{aligned} \tag{4.3}$$

Given that the input range of the CORDIC IP falls within the interval of $[-\pi, \pi]$, we were required to perform a range reduction process following the conversion of

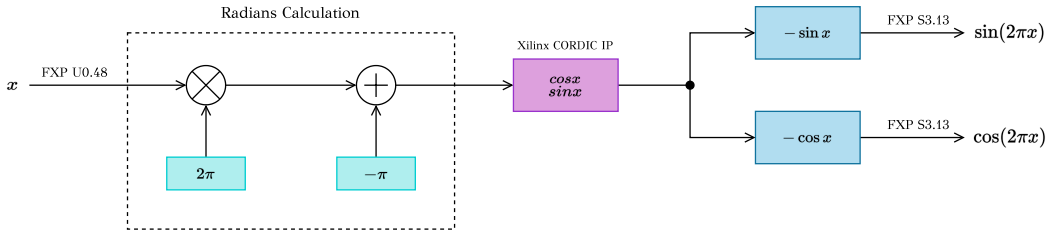



**Figure 4.4**: Hardware implementation of Sine/Cosine

our random input variable into radians. This adjustment was necessary to adhere to Xilinx's IP specifications, since our radians were in the range $[0, 2\pi]$. Afterwards, by following the trigonometric identities of shifting $\sin x$ and $\cos x$ by one half period (Eq. 4.4), we changed the output sign in order to obtain the correct result. A block diagram illustrating the Sine/Cosine calculation is presented in Fig. 4.4.

$$\begin{aligned} \sin(2\pi x - \pi) &= -\sin(2\pi x) \\ \sin(2\pi x - \pi) &= -\cos(2\pi x) \end{aligned} \tag{4.4}$$

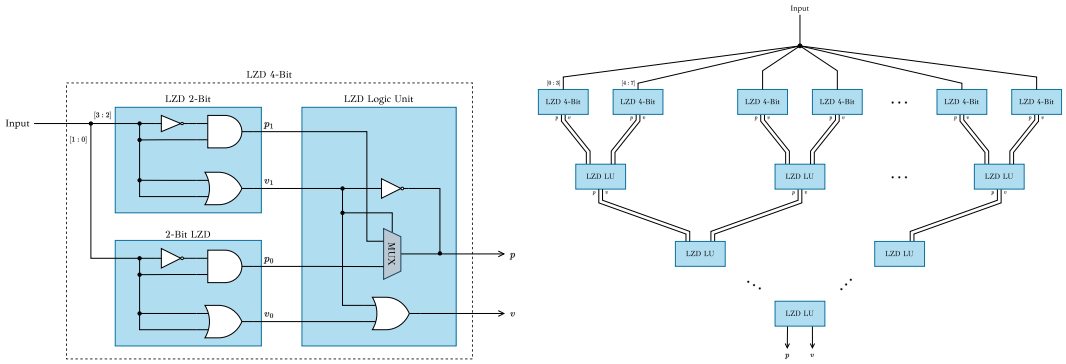
### 4.3.4 Logarithm Implementation

For the evaluation of an elementary function, we adhered these three following steps, as stated in [39]:

1. Range reduction of the input variable, over a more appropriate interval.
2. Approximation of the function on the reduced interval.
3. Range reconstruction of the calculated value to the original input range.

#### Leading Zero Detector

A Leading Zero Detector (LZD) is a module that counts the number of consecutive leading zero digits in a binary number representation, until the first nonzero digit is encountered. It's purpose is to determine the appropriate shift needed for input range reduction and was used both in logarithm and square root calculation. The LZD was implemented with the combinational circuit of the Fig. 4.5, as presented in [40]. The main idea is to implement a base 4-bit LDZ, and then expand in a tree topology for the required number of bits. Each module at the top layer, read's 4 different bits from the input with a 4-bit LDZ, while the following layers are implemented using only the LZD Logic Unit (LU). The final result includes the variable $p$, with the number of the leading 0s and $v$, a valid signal whether the input number contains 1s.




**(i)** 4-bit LZD **(ii)** 48-bit LZD tree layout

**Figure 4.5**: Combinational LZD implementation

### Logarithm

Following the block diagram in Fig. 4.6, we can see that the logarithm input signal $x$ is $U0.48$, which corresponds to $x \in [0, 1 - 2^{-48})$. Thus the function $-2\ln(x)$ is the range $[0, 66.54]$, so with an signed format, the output signal is S8.24. For the range reduction of the variable $x$, based on Eq. 4.5 from [41], were $M_x = [1, 2)$ and $E_x$ is an exponent, we can define the calculation of our natural logarithm, by approximating the logarithm over $[1, 2)$ and then performing range reconstruction.

$$\ln(x) = \ln(M_x \cdot 2^{E_x}) = \ln(M_x) + E_x \cdot \ln(2) \tag{4.5}$$

The initial range reduction was performed using the aforementioned LZD circuit, followed by a left shift depending on the leading zeros. For the calculation of the natural logarithm over $[1, 2)$, we opted for a piece wise linear interpolation, with a first order polynomial. Coefficients for the first order polynomial were calculated using MATLAB Curve Fitting Toolbox, after diving our interval in 256 segments. Fig. 4.7 depicts the accuracy of the Piecewise Linear Approximation of the logarithm over the specified interval. These coefficients after converted to fixed point format, there were stored in a 12Kb ROM, since the first coefficient was 16-bit while the second one in 32-bit. By choosing high resolution for the polynomial constants, we can avoid using a second order polynomial approximation, that will increase the complexity, although offering better performance regarding the accuracy. ROM contents were accessed using as address the 8 most significant fraction bits of the reduced variable. With the result of the linear approximation, we perform based on Eq. 4.5 the according range reconstruction using the LZD output. By utilizing the valid out signal '$v$' from the LZD, we can find if the input is zero. In this case, since $\lim_{x\to 0}(-2\ln x) = \infty$, we map the output signal to the largest FXP S8.24 number, $2^7 - 1$.

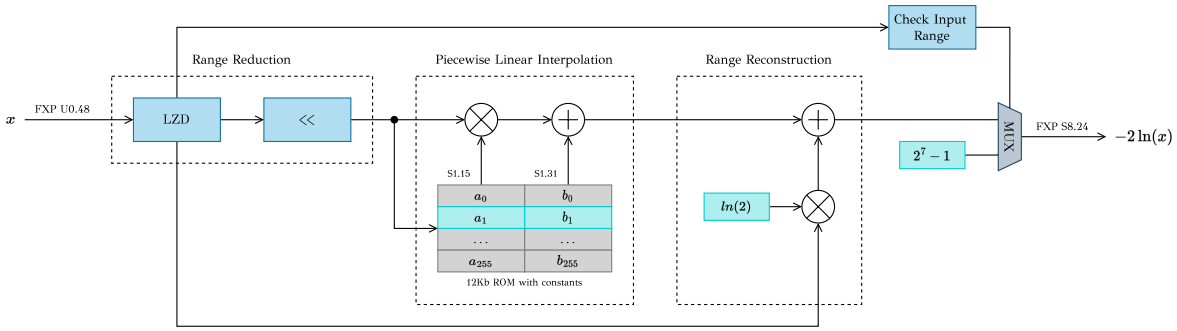



**Figure 4.6**: Hardware implementation of natural logarithm

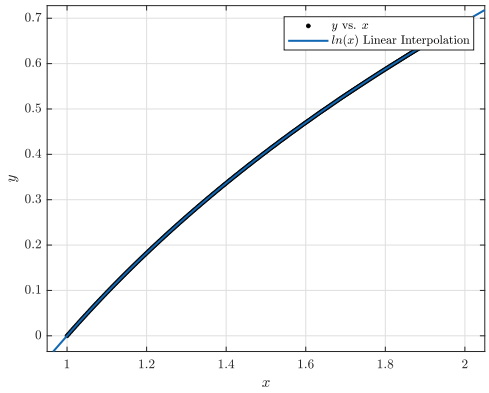



**Figure 4.7**: Logarithm Piecewise Linear Approximation versus floating point model in MATLAB, with an expected $R^2 \approx 1$

### 4.3.5 Square Root Implementation

The Square Root was calculated with the CORDIC IP from Xilinx in square root functional configuration. The input value of the core, considering it's in fixed point point format, must be in the interval $[0, 2)$, thus according to input $x \in [0, 66.54]$ as the $-2\ln(x)$ result, we had to perform corresponding range reduction and reconstruction. In Eq. 4.6 we can see the correlation between the input variable $x$ and the calculation of the square root after performing range reduction, based on [41]. The final implementation is depicted in Fig. 4.8.

$$\sqrt{x} = \sqrt{M_x \cdot 2^{E_x}} = \begin{cases} \sqrt{M_x} \cdot 2^{E_x/2}, & \text{if } E_x \bmod 2 = 0 \\ \sqrt{2M_x} \cdot 2^{(E_x-1)/2}, & \text{if } E_x \bmod 2 = 1 \end{cases} \tag{4.6}$$

The range reduction was performed using the LZD of Fig. 4.5 for the responding input bits, followed by the according right shift. For our proposed architecture, if the input variable is in range $[0, 2)$ the input is going directly into the CORDIC IP and no range reconstruction is required. The two calculations (with/without range reduction) are perform in parallel and a MUX at the output selects the correct result based on the x value ($x < 2$ or $x \geq 2$). The output result of the square root is in

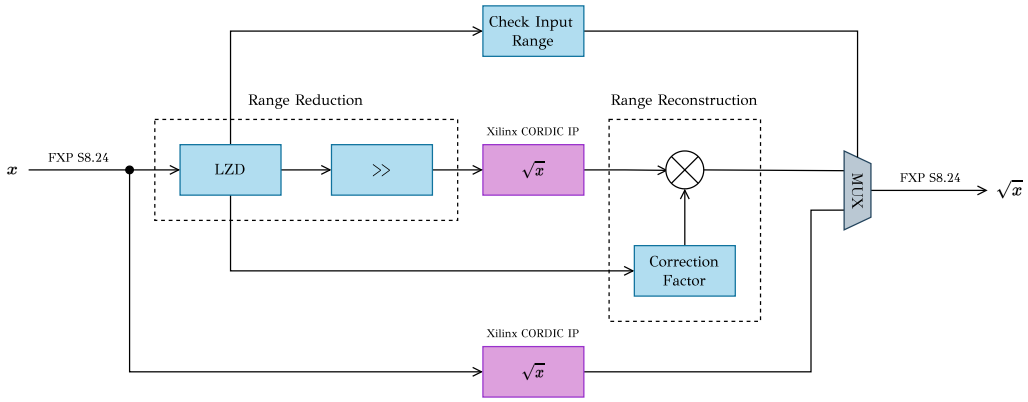



**Figure 4.8**: Hardware implementation of square root

$[0, 8.15]$, so with a fixed point representation we select S5.13.

## 4.4 Complex Channel Coefficients

In the chosen implementation as shown in Fig. 4.9, the complex coefficients for the FIR Filter are calculated in parallel. Therefore, a Serial to Parallel Module was used, were after $N$ cycles, $N$ new Gaussian variables were loaded to the Multiplication/Addition units. Each calculation is based on Eq. 4.7, were a Multiplication and Addition unit was used for the real part, and only a Multiplication unit for the imaginary part.

$$\begin{aligned} \mathfrak{Re}(c) &= m_i \cdot \mathfrak{Re}(g) + a_i \\ \mathfrak{Im}(c) &= m_i \cdot \mathfrak{Im}(g) \end{aligned} \tag{4.7}$$

The constants $m_i$ and $a_i$ are pre-calculated with the Eq. 4.8, that takes into consideration the Fading Channel parameters; the average path gain $P$ and $K$ factor. By setting the $K$ factor equal to 0, the fading distribution of the channel follows a Rayleigh distribution; otherwise it adheres to a Rician distribution.

$$m_i = \sqrt{\frac{PK}{K+1}}, \quad a_i = \sqrt{\frac{PK}{2(K+1)}} \tag{4.8}$$

Furthermore, in order to update the Coefficients of the FIR Filter in run-time, a cyclic loading pattern must be followed. Thus, a BRAM based FIFO was utilized for each coefficient result, and by controlling the Read and Write Enable signal of each FIFO with a Finite State Machine (FSM), the desired pipeline was implemented. The required timings for the aforementioned procedure, are illustrated in Fig. 4.10,

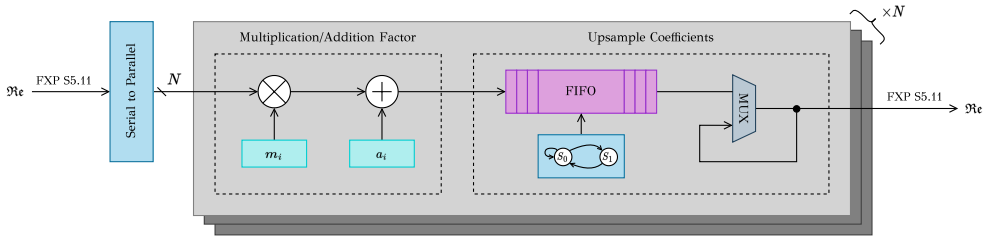



**Figure 4.9**: Complex Coefficients Calculation

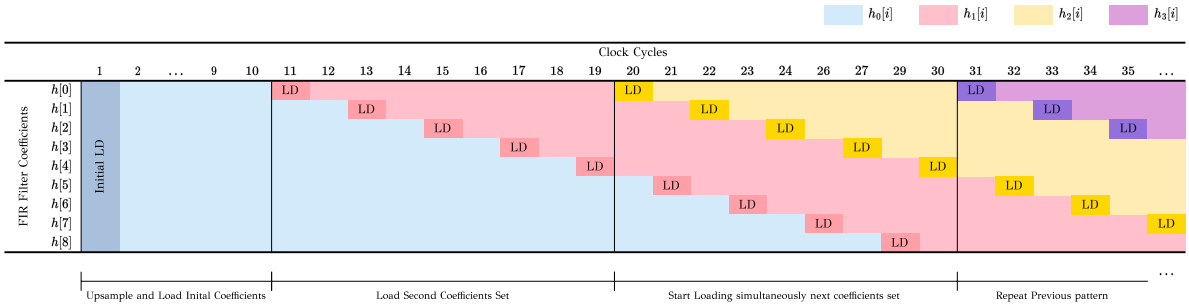



**Figure 4.10**: FIR Coefficients Loading Pipeline

where the timings can be categorized into three primary segments; the initial load of the first coefficients set, the load of the second set while no other coefficient update is happening, and finally the recurring pattern of run-time coefficient loading. Each value load (LD) happens every 2cc because of the registers topology in the chosen FIR Implementation.

## 4.5 Finite Impulse Response Filter

The FIR Filtering process, is employed to filter and modify a signal based on a finite sequence of input data. In case of linear digital filters, the output is computed as a linear combination of the input and the previous samples of input and output signals. In the fading channel emulator, the FIR Filter Tap number is defined by the maximum discrete path delay in the emulator specifications [42]. This block filters the input data, essentially the output of the Transmitter (TX) of the Telecom chain, with the channel coefficients, generating the faded output symbols. Since both the signal and the coefficients are complex, a complex implementation of FIR was utilized, were three FIR filters are used in parallel [16] (Fig. 4.11). Furthermore, one block for parallel addition and one block for parallel subtraction were implemented, for simultaneous calculation of the coefficients for the Complex FIR Filter. One module with 2cc delay was utilized for the middle FIR Module, for coefficient synchronization.

In order to efficiently use the DSP blocks of the FPGA, we followed the direct Systolic architecture illustrated in Fig. 4.12 [43]. Although having a higher latency

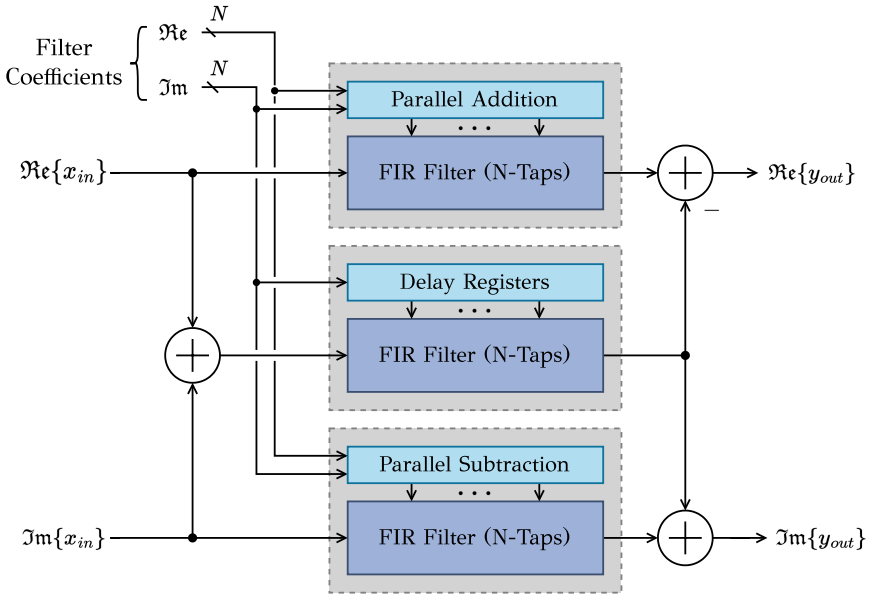



**Figure 4.11**: Complex FIR Architecture

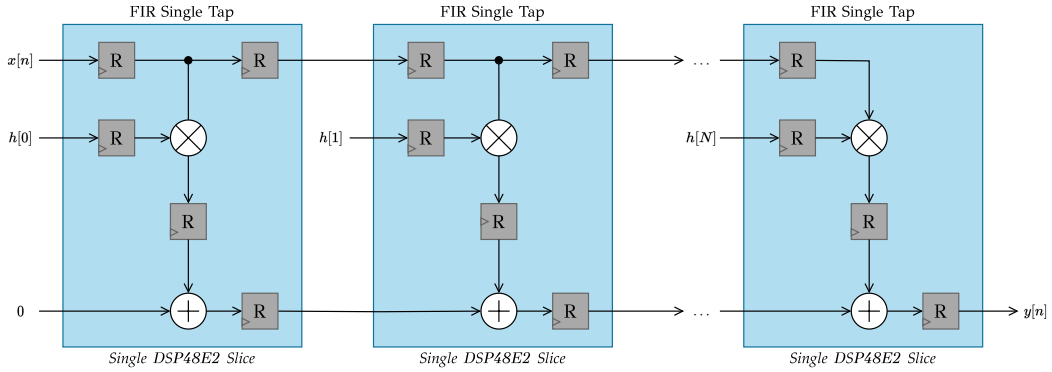



**Figure 4.12**: FIR Direct Systolic

opposed to a transposed FIR Filter, if offers superior performance due to the absence of a high fanout input signal. Furthermore, in order to avoid potential overflow, the total bit growth for a N-Tap FIR filter is calculated using the Eq. 4.9. Subsequently for a 9-Tap FIR filter that we're using, the total bit growth of the 16-bit input $x_{in}$ is 19 bits.

$$BG = h_w + \lceil \log_2 N \rceil \tag{4.9}$$

# Chapter 5

# Experimental Results

## 5.1 Introduction

In this chapter experimental results for each implemented block are presented. Moreover, for the SIMD Demodulation (original and proposed approximate algorithm), we evaluate our designs regarding the Execution Time and the BER Performance. As to the FPGA based Fading Channel Emulator, we demonstrate the Resources occupied for our implementation, and we evaluate the design based on the Fading Effect PDF, while we also present the faded symbols I-Q plots.

## 5.2 SIMD Demodulation Setup

For our Telecommunication Chain simulation, we used the TOPCOM++ Library developed in C++. Based on some examples of a DVB-S2 (Digital Video Broadcasting - Satellite - Second Generation) chain, we defined all the required blocks for our setup, as shown in Fig. 5.1. Furthermore as a secondary test case, we examined a telecommunication chain in which the LDPC Encoder/Decoder was bypassed.

The simulation parameters are given from a text file, that can be configured to accept multiple values, and traverse along them in multiple simulation runs accordingly. This feature was particularly useful for running batches of implementation arrangements without additional user input. Furthermore, the simulation results (BER, Execution time, Speedup calculation) were logged in files, to facilitate post processing and derive all the necessary results.

In the platform of choice (Xilinx ZCU106), we opted for Ubuntu 20.04.6 LTS aarch64 as our operating system of choice, with kernel 5.4.0-1015-xilinx-zynqmp. For code compilation we relied on GCC (g++ Version 9.4.0) and on CMake for building the source files. All the subsequent implementations were compiled using the (`-O3`) flag, representing the highest and most aggressive optimization level, that enables all optimizations outlined in (`-O2`) and additionally, it triggers optimizations flags regarding loop transformations, common sub expression elimination and loop

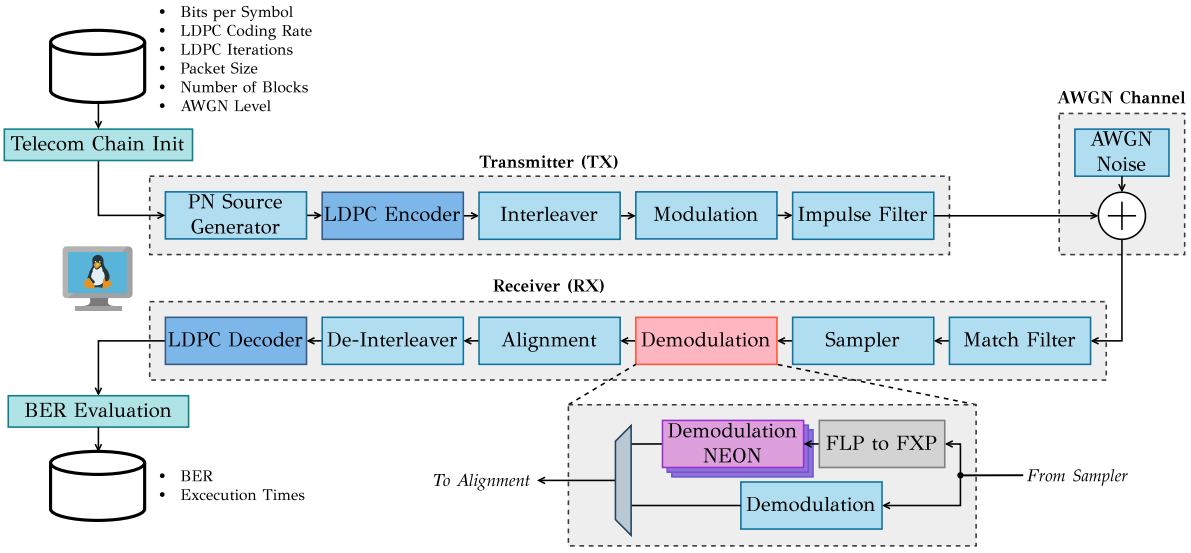



**Figure 5.1**: Telecommunication Chain setup for Simulation

vectorization. Furthermore, for the implemented C++ functions, the `inline` keyword was used, that instructs the compiler to replace the code within the function definition each time it's called [44]. By employing inline functions, program execution time can be enhanced by removing the additional steps involved in making function calls. Additionally the `__restrict__` keyword was utilized for each function arguments.

### 5.2.1 Original Approximate Demodulation Performance

Without our setup being the Telecom Chain of Fig. 5.1 with the same input simulation parameters, we benchmarked each Demodulation implementation for different QAM Modulations, in order to examine the execution time. From Fig. 5.2, we conclude that for a larger QAM Modulation (for larger input bits), the more efficient the use of the NEON engine is with an exponential rate, until a saturation point is reached. Between the different implementations, the FXP8-B16 is the faster for the large QAMs, since the int16x8 vectors are kept constant through all the calculations, while we benefit from some 8-bit arithmetic operations for the initial distances calculation. In order to maintain consistent results through all simulations, we kept packet size at 64800 symbols, a fixed LDPC Coding Rate at 3/4 and 32 blocks.

### 5.2.2 Demodulation BER Performance

One of the key metrics in telecommunications for the system's accuracy is the Bit Error Rate (BER), that is calculated by comparing two binary streams; essentially

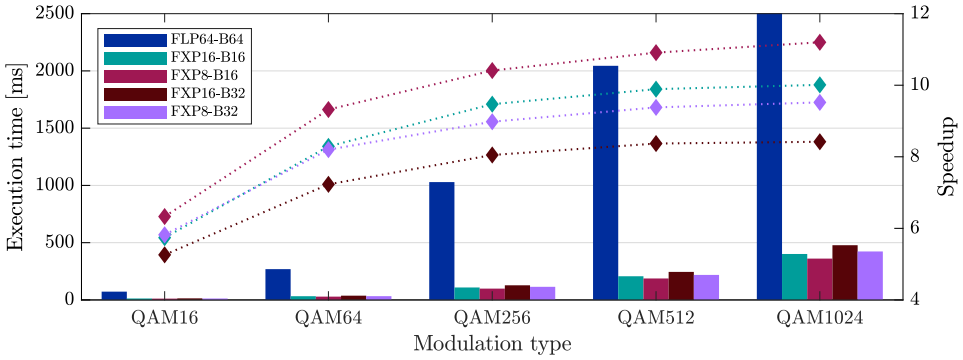



**Figure 5.2**: SIMD Demodulation Execution times

by dividing the number of error bits, by the total number of transmitted bits. For the following BER Analysis, as a baseline line model in each we used the default Demodulation Block floating model. For additional evaluation, we compared our results with the theoretical BER values in an AWGN channel for uncoded data, that is calculated using equations as stated in Section 2.3.

For the base NEON Demodulation with LDPC Coding, we derive from Figs. 5.3 and 5.4 that across various LDPC iterations, there is minimal loss in BER performance. Particularly, the FXP16-B32 configuration has superior performance, while the FXP8-B16 model has the least favorable performance among the evaluated setups.

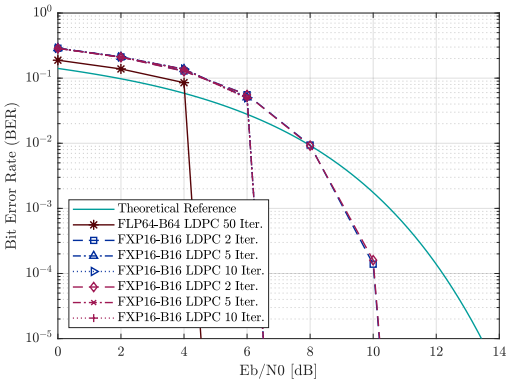
### 5.2.3 Demodulation with Approximation Techniques Performance

For the QAM constellations we implemented the approximation technique proposed in Section 3.5, we present the results regarding the improvement in execution time, over the original SIMD Demodulation method. We examined the performance of the Receiver Block (RX) of our chain in two scenarios:

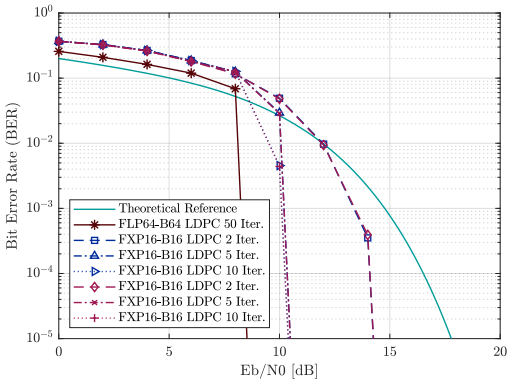
1. *Without FEC Code*, where no Forward Error Correction was used, and the classification of the output bit $b_i$ was done by only checking the sign of the calculated $LLR$ value, as stated in Eq. 5.1.

$$b_i = \begin{cases} 0, & \text{if } LLR < 0 \\ 1, & \text{if } LLR \geq 0 \end{cases} \tag{5.1}$$

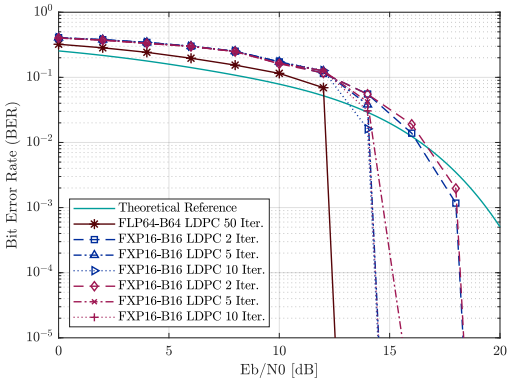
2. *With LDPC Code*, where a Low-Density Parity Check Code (LDPC) was used for generating parity bits, with a constant coding rate through all simulation runs. As we will analyze below, we investigated various LDPC iteration settings, to assess their impact on the Receiver's execution time, as well as the alterations in the system's BER.



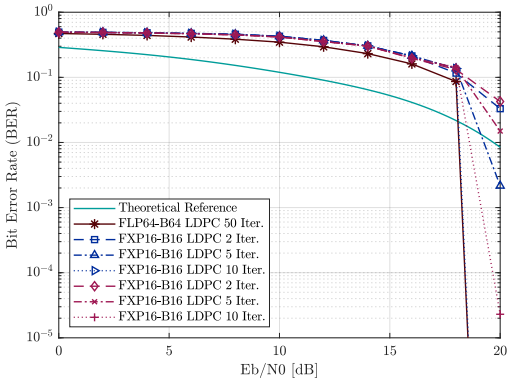

**(i)** QAM16, MUL16



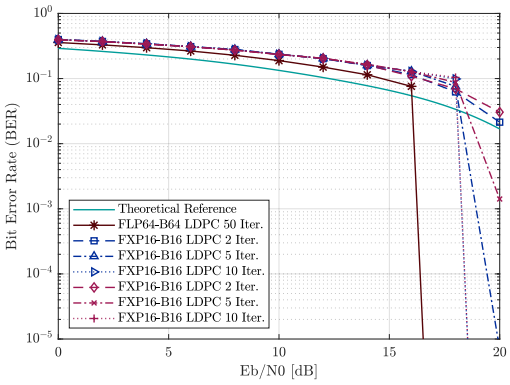

**(ii)** QAM16, MUL32




**(iii)** QAM64, MUL16

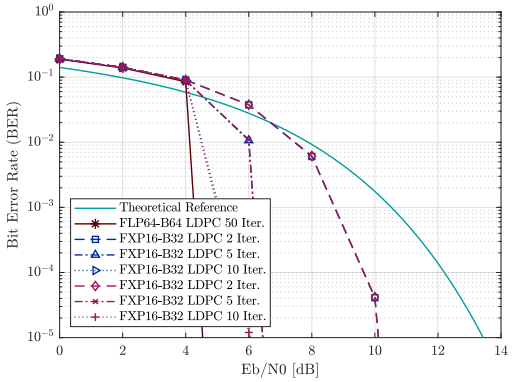



**(iv)** QAM64, MUL32

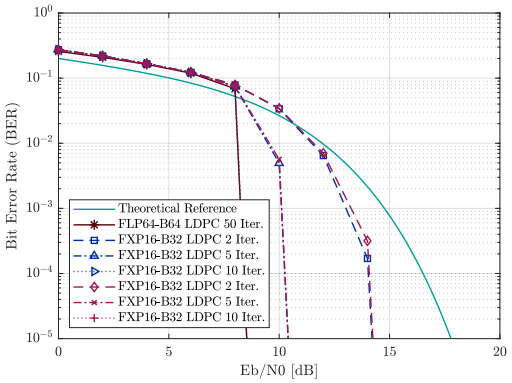



**(v)** QAM1024

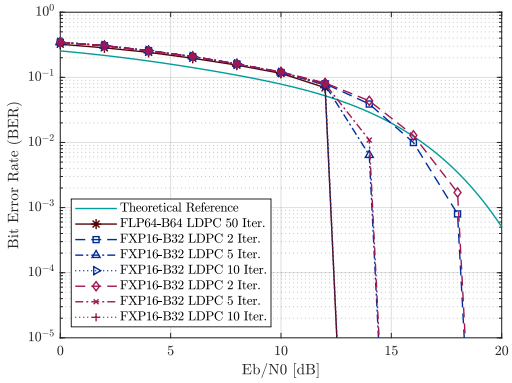
**Figure 5.3**: BER Performance with LDPC Coding for SIMD Demodulation implementations with 16-bit multiplication



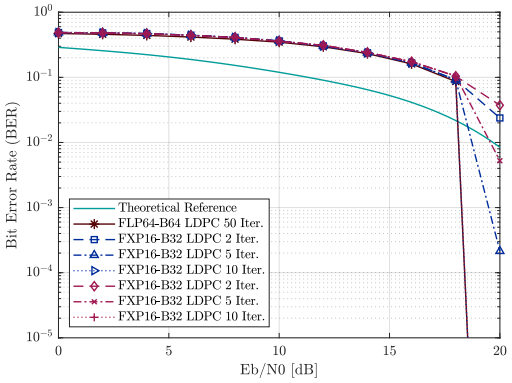

**(i)** QAM16, MUL16



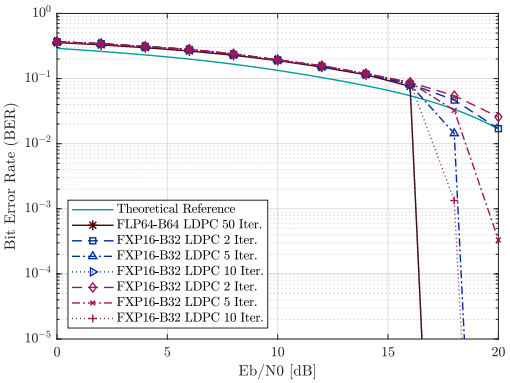

**(ii)** QAM16, MUL32




**(iii)** QAM64, MUL16

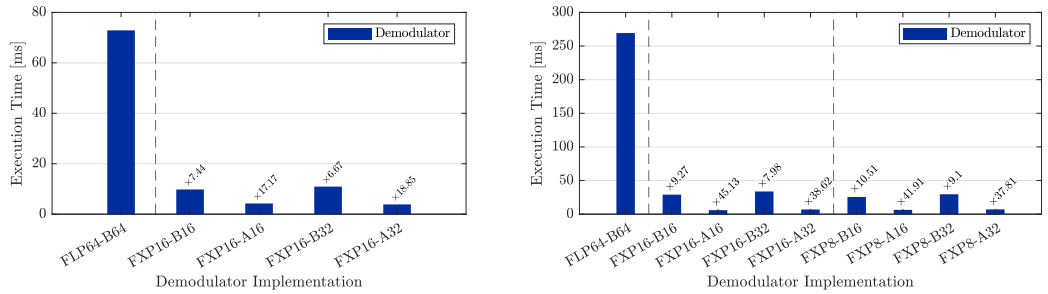



**(iv)** QAM64, MUL32



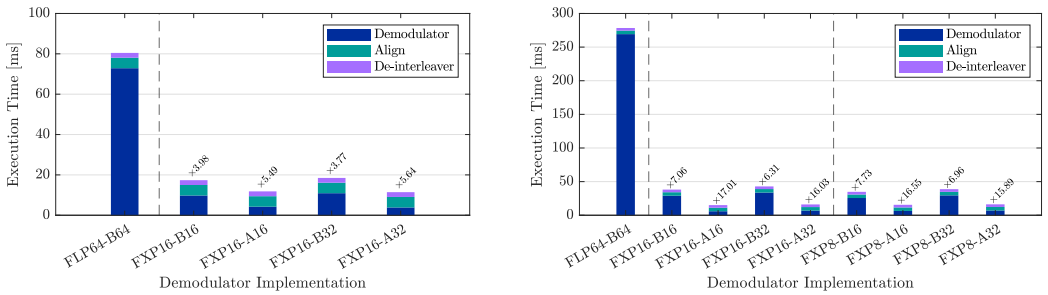

**(v)** QAM1024

**Figure 5.4:** BER Performance with LDPC Coding for SIMD Demodulation implementations with 32-bit multiplication




**(i)** QAM16 **(ii)** QAM64

**Figure 5.5**: SIMD QAM Demodulation performance




**(i)** QAM16 **(ii)** QAM64

**Figure 5.6**: Receiver (RX) performance without FEC

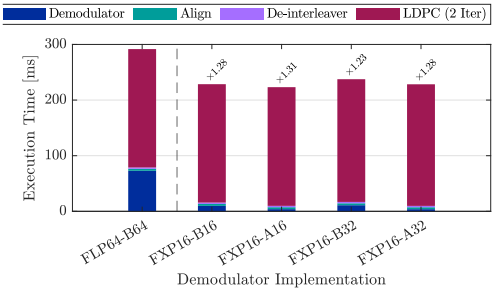
Referring to Fig 5.5, where only the execution time of the Demodulator Block for QAM16 and QAM64, we observe that we have a speedup up to $45\times$, compared to the initial implementation. Additionally by applying the proposed Approximation Technique, there is an additional performance enhancement up to $4\times$, over the baseline NEON implementation; a foreseeable outcome, since as we demonstrated in Table 3.1, about $1/4$ of the arithmetic operations are required.

With the whole Receiver block, we can examine the impact of the Demodulation Optimization over the total time needed for the processing of a received symbol. As described previously, Fig. 5.6 illustrates the execution time of the receiver without the use of FEC for each modulation.

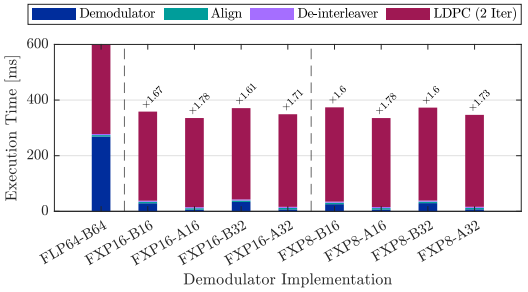
By now applying LDPC Code for varying iterations (2, 5, 10), as depicted in Fig. 5.7, it becomes evident that as the iteration count increases, the LDPC Decoder Block bottlenecks the Receiver's performance. The impact of this block is significant, to the extend that for 10 iterations, the acceleration achieved through the Demodulation is negligible.

### 5.2.4 Demodulation with Approximation Techniques BER

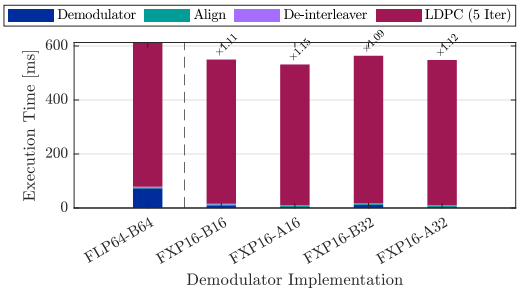
Fig. 5.8 shows the BER performance of our Approximation Technique for QAM16 and QAM64 without the use of FEC. In both constellation schemes we can observe that the BER doesn't get significantly affected from the original algorithm, while all



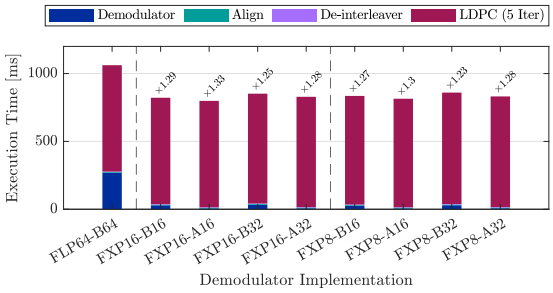

**(i)** QAM16, LDPC 2 Iter.



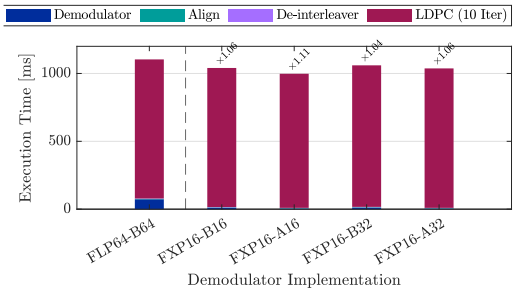

**(ii)** QAM64, LDPC 2 Iter.



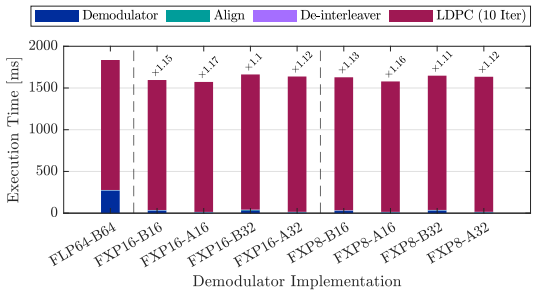

**(iii)** QAM16, LDPC 5 Iter.




**(iv)** QAM64, LDPC 5 Iter.

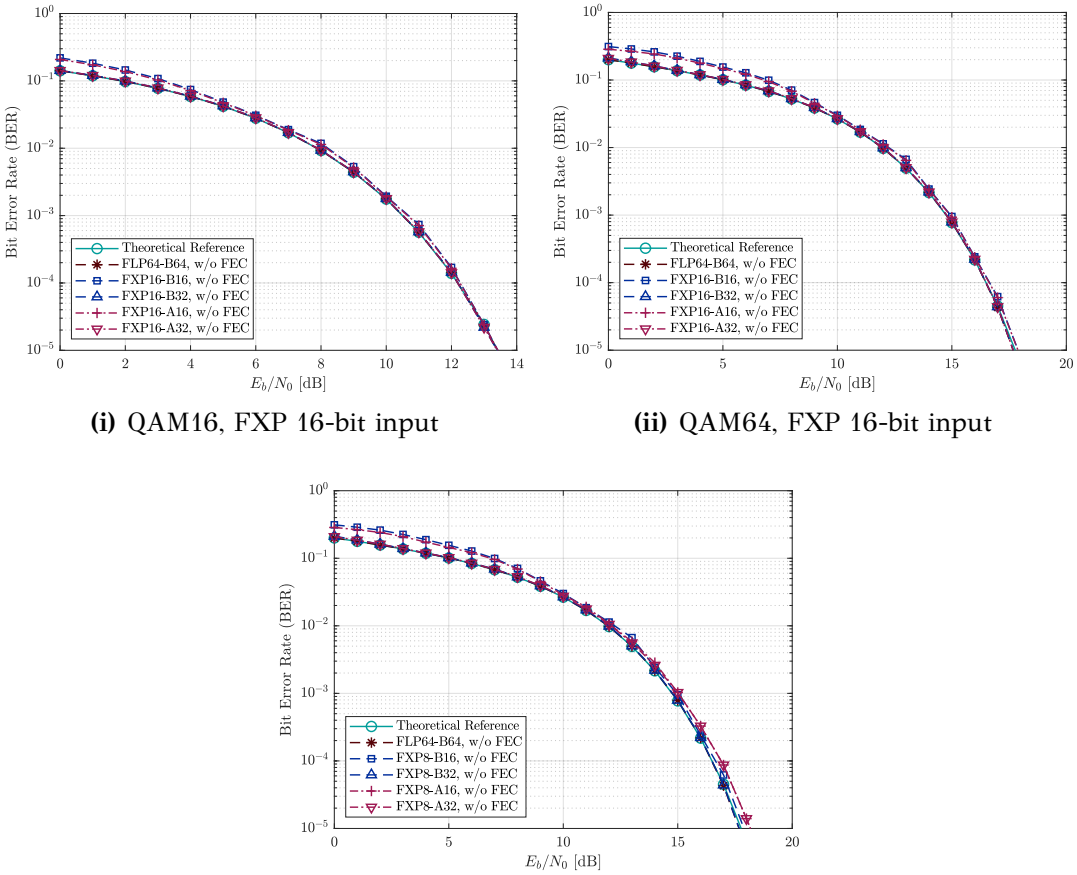



**(v)** QAM16, LDPC 10 Iter.




**(vi)** QAM64, LDPC 10 Iter.

**Figure 5.7**: Receiver (RX) performance with LDPC Decoding




**(i)** QAM16, FXP 16-bit input

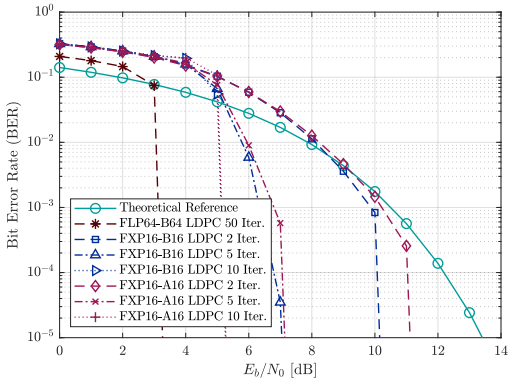
**(ii)** QAM64, FXP 16-bit input

**(iii)** QAM64, FXP 8-bit input

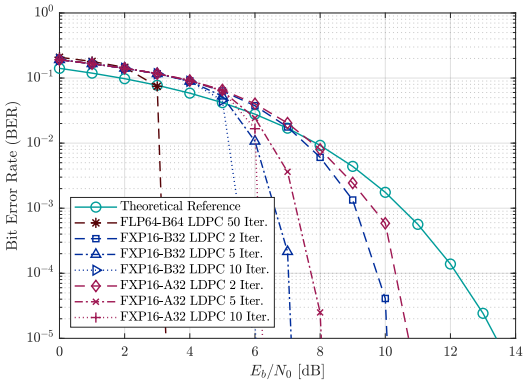
**Figure 5.8**: BER Performance without FEC for our proposed Demodulation implementation with Approximation Techniques

implementations match the theoretical estimation. Furthermore, the BER accuracy is similar regardless of the input format (FXP8/FXP16).

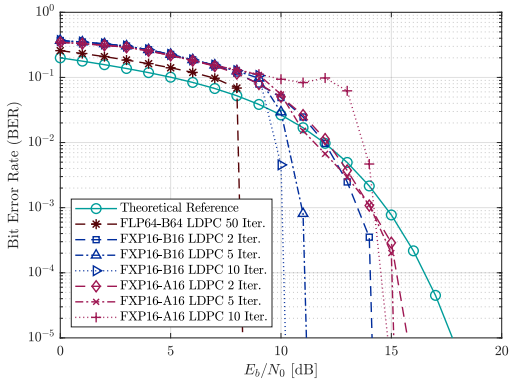
Fig. 5.9 shows the performance of the Demodulator using LDPC for different iterations. In the QAM64 plots for the highest LDPC iteration settings (10 Iterations) we observe a sudden rise in the BER value. This accuracy loss in contrast to the original algorithms, happens possibly due to the LLR values deviation, in the points were a Euclidean Distance from an adjacent Quadrature is necessary.



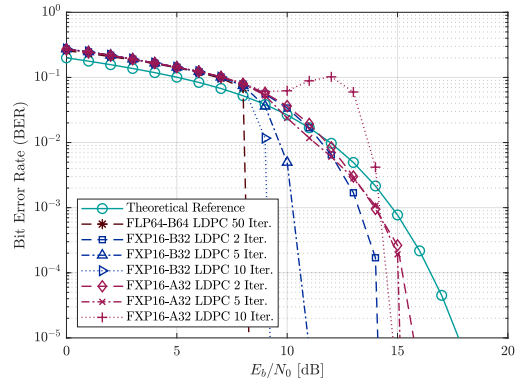

**(i)** QAM16, FXP16-A16



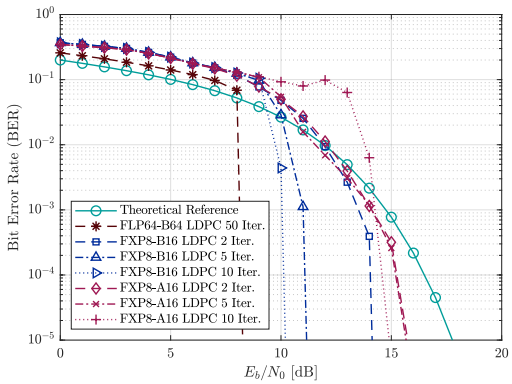

**(ii)** QAM16, FXP16-A32



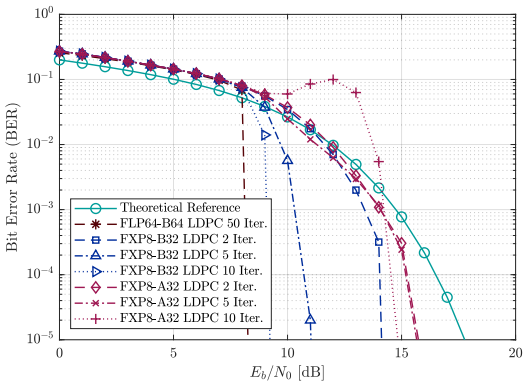

**(iii)** QAM64, FXP16-A16




**(iv)** QAM64, FXP16-A32




**(v)** QAM64, FXP8-A16

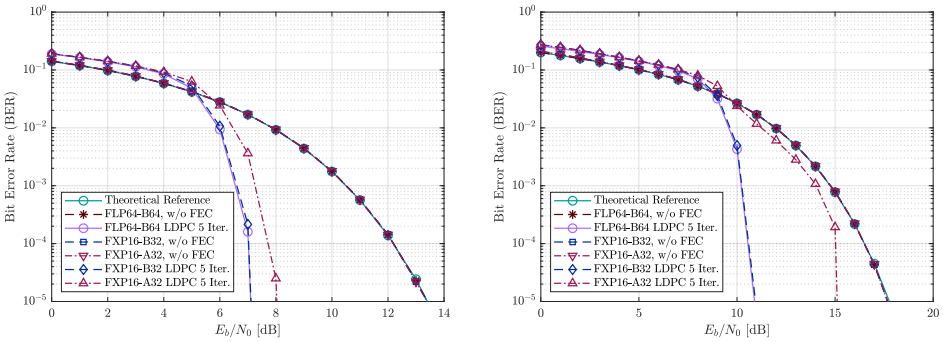



**(vi)** QAM64, FXP8-A32

**Figure 5.9**: BER Performance using LDPC for our proposed Demodulation implementation with Approximation Techniques

### 5.2.5 Chosen Implementation

The aforementioned previous implementations as we've seen present trade-offs, between the execution time and the BER performance. Therefore we concluded that the best option between speed and accuracy is FXP16-A32 model, since it outperformed the other in terms of BER accuracy, while still offering sufficient gains




**(i)** QAM16 **(ii)** QAM64

**Figure 5.10**: BER Performance conclusions for QAM16 and QAM64

regarding execution time. In Fig. 5.10 the results for the selected implementation are presented for both QAM16 and QAM64 constellations.

In Table 5.1 the perfomance gains over the original floating point implementation, for the chosen implementations are presented. The demodulation block depending on the constellation, shows up to 38× speedup over the original algorithm, and up to 4× speedup over the SIMD optimized original Demodulation algorithm.

**Table 5.1**: Chosen Implementation Speedup over Original Algorithm

| | | Modulation | |
|---|---|---|---|
| Implementation | Configuration | QAM16 | QAM64 |
| FXP16-B32 | Only Demodulation | 6.67× | 7.98× |
| | RX w/o FEC | 3.37× | 6.31× |
| | RX w/ LDPC 5 Iter. | 1.09× | 1.25× |
| FXP16-A32 | Only Demodulation | 18.85× | 38.62× |
| | RX w/o FEC | 5.64× | 16.03× |
| | RX w/ LDPC 5 Iter. | 1.12× | 1.28× |

## 5.3 FPGA Fading Channel Emulation

### 5.3.1 Resource Utilization

With the demand of a fading channel emulator that offers high performance, the implementation is heavily pipelined to maximize the operating frequency. For Synthesis and Implementation of the design, Xilinx Vivado 2022.2 was used, with the resources occupied are depicted in Table 5.2. The throughput of the module can be computed directly from the maximum frequency, since one input sample is

processed every clock cycle. Regarding the power usage, running the implementation reports $1.514$ Watt Total on-chip power.

**Table 5.2**: Resource Usage by the Fading Channel Emulation

| Resources | Used | Available | Utilization Ratio |
|---|---|---|---|
| LUT | 2504 | 230400 | 1.09 % |
| FF | 3851 | 460800 | 0.84 % |
| BRAM | 9 | 312 | 2.88 % |
| DSP | 49 | 1728 | 2.84 % |

### 5.3.2 Fading Channel Emulator Evaluation

#### Parameters Selection

Before applying the fading effect to a signal, it's essential to ensure that the channel specifications are appropriately configured for the intended scenario. Furthermore for a realistic channel model parameters, the following suggestions should be taken into consideration:

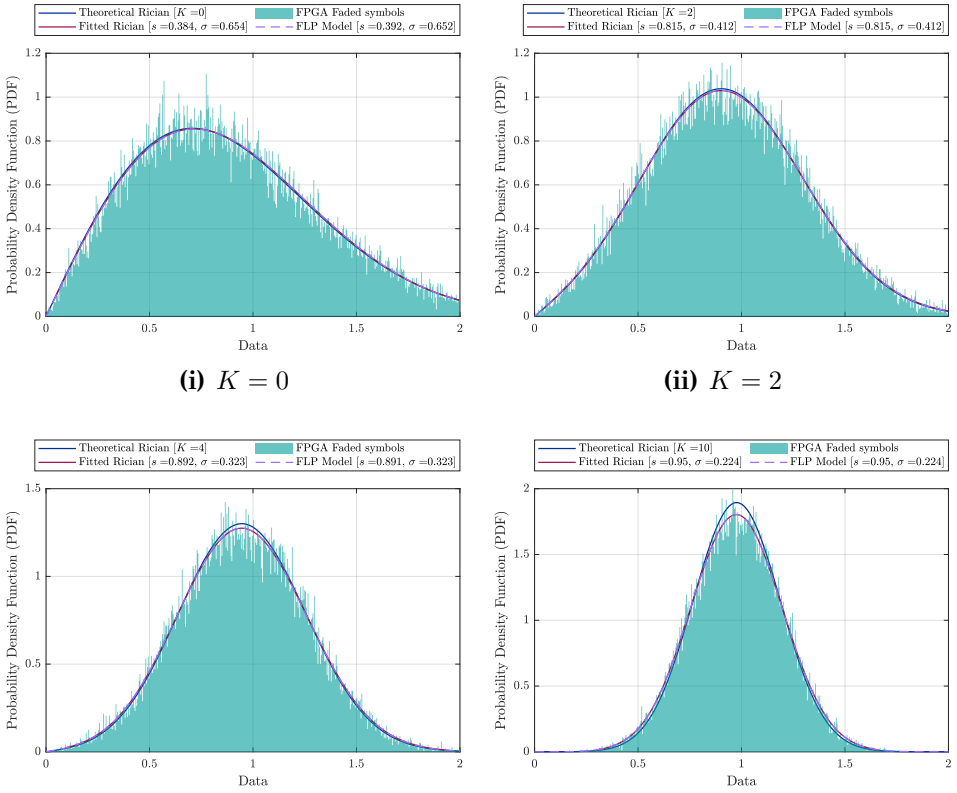
1. *Path Delays* The initial delay that is associated with the first path arrival, is conventionally set to zero. In indoor environments, the subsequent delays typically fall within the interval $[10^{-9}, 10^{-7}]$ seconds, whereas in outdoor setups, the delays typically span the range $[10^{-7}, 10^{-5}]$ seconds.

2. *Average Path Gains* Indicating the average power gain for each fading path, they generally are in the range $[-20, 0]$ dB. Frequently, normalization of these values is done, in order to ensure that the expected total power of the combined path gains is 1.

3. *K-Factor* With this parameter the ratio of a specular-to-diffuse power for a direct LoS path is defined. The factor is expressed linearly in the range $[0, 10]$. A K-factor equal to 0 represents Rayleigh fading, while the rest of the values represent Rician fading.

#### Fading Effect

For the evaluation of the Fading Channel Emulation, we examined the Probability Density Function (PDF) of the Faded Symbols, as shown in Fig. 5.11. In the presented plots, the histogram of the faded symbols is depicted and three calculated PDF curves. The first PDF is a theoretical Rician Distribution based on the according K-factor used. For calculating the $\sigma$ and $s$ parameters form the K-factor the method in [45] was utilized, where the approximations in Eq. 5.2 were proposed.




**(i)** $K = 0$ **(ii)** $K = 2$

**(iii)** $K = 4$ **(iv)** $K = 10$

**Figure 5.11**: PDF of faded symbols for different K-factor, 8 Path Delays and Average Path Gains at -20dB

$$s \to \sqrt{\frac{K}{K+1}}, \quad \sigma \to \sqrt{\frac{1}{2(K+1)}} \tag{5.2}$$

The second and third PDF curves, represents a Rician Distribution fitted to the faded data using Maximum Likelihood Estimation. The fit was performed for the faded symbols generated from both the FPGA and a floating model in Matlab, as a benchmark. The $\sigma$ and $s$ values shown are the calculated parameters for the fitted distribution.

From the presented plots, we conclude that our generated fading symbols for the different K-factors, adheres to the theoretical expected function. The fading symbols generated in the FPGA compared to a reference floating point model in Matlab, achieve minimum accuracy loss. Specifically, for a set of $10^5$ complex input symbols, the Real part demonstrates a MRE of $0.39\%$, the Imaginary part shows an MRE of $0.44\%$, while the Magnitude of the complex symbol registers an MRE of $0.06\%$.

Fig. 5.12 shows combined the PDF for each K-factor combined, where the differences between them can easily be distinguished. With the reduction of the K-factor; thus the increase of the fading effect, the distribution center moves to the left, while

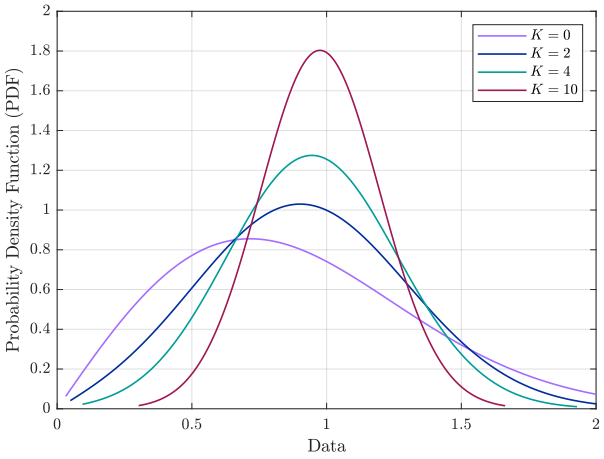
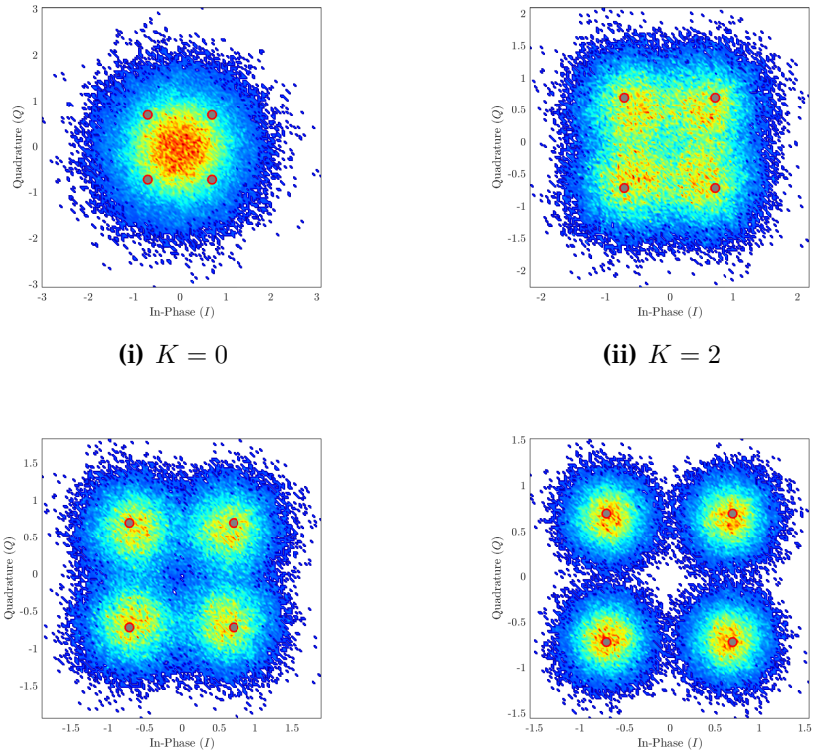



**Figure 5.12**: Combined PDFs for different K-factors

the peak value decreases. significantly. For a large K-factor (ex. $K = 10$) where we have a strong LOS component, we observe that the PDF tends to follow a normal distribution.

An additional method for evaluating the fading effect impact, is by observing the scatter plot of the output symbols. Fig. 5.13 illustrates the effects of different K-factors on a QAM4/QPSK constellation. In the chosen color scheme, orange-red indicates the maximum symbol count at a particular position, whereas blue signifies the minimum symbol count. From the presented plots we can derive that for the largest K-factor the fading effect is negligible, while as the K-factor decreases, the reference symbols shift to the center of the axes. Furthermore, the symbol positions in the original constellations are market with a red/grey circle.




**(i)** $K = 0$ **(ii)** $K = 2$

**(iii)** $K = 4$ **(iv)** $K = 10$

**Figure 5.13**: $I - Q$ Constellation after different fading effects

# Chapter 6

# Conclusions and Future Works

In this thesis in the first place, the soft Demodulation algorithm for QAM constellations is presented and implemented using SIMD instructions, using NEON Intrinsics. Different implementations were evaluated, regarding the input size and the precision of the arithmetic operations. Furthermore, for two constellation types (QAM16/QAM64) an approximation technique was proposed in order to reduce the number of arithmetic operations required, offering improvement in the execution time of the Demodulation block and thus in the Receiver, while exploration was also conducted regarding the accuracy performance. We evaluated the BER of a telecommunication chain for different scenarios; Initially without the use of FEC, while afterwards we added an LDPC Encoder/Decoder with various iterations.

Furthermore, an FPGA-based Fading Channel emulator was implemented, where the desired effect was obtained, by filtering the input signals with uniform Gaussian variables. These constants were generated using various elementary functions which were employed by different approximation methods. The accuracy of the emulator was evaluated by comparing the PDF of the channel's output with a theoretical model, as well as a reference floating point baseline.

Although the results for both the software optimizations and the hardware acceleration were encouraging, there is still room for extensions. Since as we aforementioned the LDPC block for a large number of iterations, although offering better BER performance has a noticeable bottleneck in the Receiver's execution time, as a future project, an optimization of the Decoder with NEON Intrinsics could be explored. For the FPGA-based fading channel emulator, as a future extension, it could be integrating it in a telecommunication chain that is running in the PS section of the SoC-FPGA, in order to provide channel fading effects in real time.

# Bibliography


[1] F. Xiong, *Digital modulation techniques*. London, 2006.

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

[3] S. Haykin, *Communication systems*. John Wiley & Sons, 2008.

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

[5] Y. Gosain and A. Gupta, "Xilinx advanced multimedia solutions with video codec/graphics engines," *Zynq UltraScale+ MPSoC. Xilinx, October*, vol. 23, 2017.

[6] V. Xilinx, "Design suite user guide: Model-based dsp design using system generator," *UG897, v2016. 1 ed., Xilinx*, 2018.

[7] ARM, "ARM Cortex-A53 Processor." https://developer.arm.com/Processors/Cortex-A53.

[8] A. Holdings, "Arm architecture reference manual for a-profile architecture," *ARM, Cambridge, UK, White Paper*, 2022.

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

[10] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 27, no. 10, pp. 1701–1713, 2008.

[11] A. Grami, *Introduction to digital communications*. Academic Press, 2015.

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

[13] J. Hamkins, "Performance of low-density parity-check coded modulation," in *2010 IEEE Aerospace Conference*, pp. 1–14, IEEE, 2010.

[14] I. Stratakos, V. Leon, G. Armeniakos, G. Lentaris, and D. Soudris, "Design space exploration on high-order qam demodulation circuits: Algorithms, arithmetic and approximation techniques," *Electronics*, vol. 11, no. 1, p. 39, 2021.

[15] V. Leon, I. Stratakos, G. Armeniakos, G. Lentaris, and D. Soudris, "Approx-qam: High-order qam demodulation circuits with approximate arithmetic," in *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–5, 2021.

[16] MathWorks, "HDL Filter Architectures." https://www.mathworks.com/help/dsphdl/ug/hdl-filter-architectures.html.

[17] A. Cassagne, M. Leonardon, R. Tajan, C. Leroux, C. Jégo, O. Aumage, and D. Barthou, "A flexible and portable real-time dvb-s2 transceiver using multi-core and simd cpus," in *2021 11th International Symposium on Topics in Coding (ISTC)*, pp. 1–5, IEEE, 2021.

[18] E. Grayver and A. Utter, "Extreme software defined radio–ghz in real time," in *2020 IEEE Aerospace Conference*, pp. 1–10, IEEE, 2020.

[19] M. Xhonneux, J. Louveaux, and D. Bol, "A sub-mw cortex-m4 microcontroller design for iot software-defined radios," *IEEE Open Journal of Circuits and Systems*, 2023.

[20] Z. Zhao, Q. Zhu, K. Mao, W. Liu, N. Li, S. Yan, and W. Huang, "An efficient hardware generator for massive non-stationary fading channels," in *2020 IEEE Globecom Workshops (GC Wkshps*, pp. 1–6, IEEE, 2020.

[21] P. Huang, Y. Du, and Y. Li, "Stability analysis and hardware resource optimization in channel emulator design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 7, pp. 1089–1100, 2016.

[22] C. Fang, K. Mao, S. Fang, Z. Zhao, B. Hua, T. Liu, and Q. Zhu, "Cordic-based general multiple fading generator for wireless channel digital twin," *Sensors*, vol. 23, no. 5, p. 2712, 2023.

[23] T. S. Rappaport, *Wireless communications: Principles and practice*, *2/E*. Pearson Education India, 2010.

[24] B. Sklar, *Digital communications: fundamentals and applications*. Pearson, 2021.

[25] J. G. Proakis, *Digital communications*. McGraw-Hill, Higher Education, 2008.

[26] M. SIMONS and M. Alouini, "Digital communication over fading channels: a unified approach to performance analysis," 2000.

[27] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of communication systems: modeling, methodology and techniques*. Springer Science & Business Media, 2006.

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

[29] A. S. U. Guide, "Ultrascale architecture dsp slice,"

[30] C.-A. ARM, "Series programmer's guide," 2012.

[31] A. Holdings, "Arm cortex-a53 mpcore processor, technical reference manual, 2014."

[32] M. Jang, K. Kim, and K. Kim, "The performance analysis of arm neon technology for mobile platforms," in *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, pp. 104–106, 2011.

[33] R. Yates, "Fixed-point arithmetic: An introduction," *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[34] V. Leon, M. A. Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, "Approximate computing survey, part i: Terminology and software & hardware approximation techniques," *arXiv preprint arXiv:2307.11124*, 2023.

[35] V. B. Olivatto, R. R. Lopes, and E. R. de Lima, "Simplified llr calculation for dvb-s2 ldpc decoder," in *2015 IEEE International Conference on Communication, Networks and Satellite (COMNESTAT)*, pp. 26–31, IEEE, 2015.

[36] D.-U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *IEEE transactions on computers*, vol. 55, no. 6, pp. 659–671, 2006.

[37] P. L'ecuyer, "Maximally equidistributed combined tausworthe generators," *Mathematics of computation*, vol. 65, no. 213, pp. 203–213, 1996.

[38] J. Volder, "The cordic computing technique," in *Papers presented at the the March 3-5, 1959, western joint computer conference*, pp. 257–261, 1959.

[39] J.-M. Muller and J.-M. Muller, *Elementary functions*. Springer, 2006.

[40] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 1, pp. 124–128, 1994.

[41] 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.

[42] J. Giron-Sierra, "Digital signal processing with matlab examples. vol. 1: Signals and data, filtering, non-stationary signals, modulation," 2016.

[43] G. Hawkes, "Dsp: Designing for optimal results. high-performance dsp using virtex-4 fpgas," *Advanced Design Guide. Xilinx Inc*, vol. 1, 2005.

[44] B. Stroustrup, "The c++ programming language: reference manual," tech. rep., Bell Lab., 1984.

[45] M. A. Richards, "Rice distribution for rcs," *Georgia Institute of Technology*, 2006.