Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Design and Evaluation of High-Order QAM Circuits using Hybrid Approximate Techniques and Arithmetic on FPGAs

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ ΑΡΜΕΝΙΑΚΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εργαστηριο Μικροϋπολογιστων και Ψηφιακων Συστηματων

Αθήνα, Ιούλιος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Design and Evaluation of High-Order QAM Circuits using Hybrid Approximate Techniques and Arithmetic on FPGAs

Διπλωματικη Εργασια

του

ΓΕΩΡΓΙΟΥ ΑΡΜΕΝΙΑΚΟΥ

Επιβλέπων: Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21 Ιουλίου 2020.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Σούντρης	Παναγιώτης Τσανάκας	Αθανάσιος Δ. Παναγόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2020

(Υπογραφή)

.....

ΑΡΜΕΝΙΑΚΟΣ ΓΕΩΡΓΙΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Περίληψη

Στον σημερινό ψηφιακό κόσμο, οι τηλεποικινωνίες αποτελούν τα θεμέλια για τη σύνδεση και την κοινή χρήση πληροφοριών. Η υλοποίηση κρίσιμων λειτουργιών και καθηκόντων σε μια ψηφιακή τηλεπικοινωνιακή αλυσίδα επιβάλλει τις αυστηρές έννοιες της «υψηλής απόδοσης» και της «χαμηλής ισχύος». Προς αυτήν την κατεύθυνση, τα Field Programmable Gate Arrays (FPGAs) θεωρούνται ελκυστικές λύσεις, καθώς προσφέρουν εξαιρετική αναλογία απόδοσης/ισχύος μεταξύ των ενσωματωμένων συσκευών.

Σε αυτή τη διπλωματική, στοχεύουμε στη λειτουργία της αποδιαμόρφωσης, μια βασική διαδικασία στο σύστημα ψηφιακών επικοινωνιών. Πιο συγκεκριμένα, πραγματοποιούμε μια διεξοδική εξερεύνηση χώρου, λαμβάνοντας υπόψη την αριθμητική και τις προσεγγίσεις στους υπολογισμούς, για να σχεδιάσουμε κυκλώματα FPGA με χρήση γλώσσας περιγραφής υλικού (VHDL). Όσον αφορά την αριθμητική, θεωρούμε fixed-point και floating-point αριθμούς. Αναφορικά με τις προσεγγίσεις, εφαρμόζουμε περικοπή των bits, αντικαθιστούμε τον ακριβή πολλαπλασιασμό με πολλαπλασιαστές Radix και μοντελοποιούμε τον πολλαπλασιασμό κινητής υποδιαστολής με λιγότερες υπολογιστικές λειτουργίες. Για τους αλγόριθμους αποδιαμόρφωσης, εξετάστηκαν 3 Soft Decision και 1 Hard Decision. Η αξιολόγηση των αλγορίθμων πραγματοποιήθηκε στο MATLAB εξετάζοντας το Bit Error Rate (BER) και το LLR. Η υλοποίηση των αλγορίθμων στο FPGA είναι παραμετροποιημένη ως προς τη μεταβλητή M για M-ary QAM και στοχεύει σε πλήρεις παράλληλες αρχιτεκτονικές για την παροχή υψηλής απόδοσης. Σε σύγκριση με τις άλλες τεχνικές διαμόρφωσης, ο αλγόριθμος 64-QAM Approximate LLR προσφέρει τις καλύτερες αντισταθμίσεις όσον αφορά πόρους-BER. Για αυτόν τον αλγόριθμο, τα αποτελέσματα εφαρμογής του στο FPGA δείχνουν ότι ανάλογα την αριθμητική και την προσεγγιστική τεχνική, οι λογικοί πόροι μειώνονται έως και 15% με μια αμελητέα διαφορά στο BER. Για μεγαλύτερες διαμορφώσεις QAM , δηλαδή 256, οι λογικοί πόροι του σχεδιασμού μειώνονται έως και 59%, εξοικονομώντας σημαντικούς πόρους του FPGA για άλλα στοιχεία του τηλεπικοινωνιακού συστήματος. Η συχνότητα ρολογιού κυμαίνεται από 357 MHz έως 555 MHz. Τέλος, πραγματοποιείται μια ολοκληρωμένη σύγκριση μεταξύ των προσεγγιστικών τεχνικών και τις περιπτώσεις που ελέγχθηκαν.

Λέξεις Κλειδιά

Ψηφιακή Αποδιαμόρφωση, Διαμόρφωση, Απόδοση, Σχεδιαστική Πολυπλοκότητα, Πόροι

Abstract

In today's digital world, telecommunication has become the foundation for communities to seamlessly connect and share information through digital processes. The implementation of critical functions and tasks of the digital telecommunication chain imposes the strict constraints of "high-performance" and "low-power". In this direction, the Field Programmable Gate Arrays (FPGAs) are considered attractive solutions, as they offer excellent performance/Watt ratio among the embedded devices.

In this thesis, we target the demodulation operation, i.e., a key process in the digital communication system. More specifically, we perform an in-depth design space exploration, considering arithmetic and approximations in the computations, to design FPGA circuits with hardware description language (VHDL). Regarding the arithmetic, we consider both fixed- and floating-point. In terms of approximations, we apply bit truncation, replace the costly accurate fixed-point multiplication with inexact radix multipliers, and model the floating-point multiplication with less computational-intensive operations. For the demodulation algorithms, we examine 3 Soft Decision and 1 Hard Decision. The evaluation of the algorithms is performed in MATLAB by examining their Bit Error Rate (BER) and LLR. The implementation of the algorithms on the FPGA is generic in variable M-ary QAM and targets full-parallel architectures to provide high-throughput. Compared to the other modulation techniques, the 64-QAM Approximate LLR algorithm delivers the best trade-offs in terms of BER-resources. For this algorithm, the FPGA implementation results show that depending on the arithmetic and the approximation scheme, we deliver logic resources reduction up to 15% with a negligible difference in BER i.e., almost the same results with the full-precision algorithm. For higher order QAM, i.e., 256, the design's logic resources reduce up to 59%, saving significant FPGA resources for other components of the telecommunication system. The clock frequency varies from 357 MHz to 555 MHz. Finally, a comprehensive comparison among all the approximation schemes and algorithms is performed.

Keywords

FPGA, VHDL, Error Tolerance, Log Likelihood Ratio, Decoding, Hardware Complexity, BER, Performance, Resources, SNR, Parallel Architecture, QAM Modulation

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κύριο Δημήτριο Σούντρη που μου έδωσε τη δυνατότητα και την ευκαιρία να εκπονήσω ένα θέμα διπλωματικής άκρως ενδιαφέρον για εμένα.

Επίσης θέλω να ευχαριστήσω θερμά τους υποψήφιους διδάκτορες Βασίλη Λέων που όποτε αντιμετώπιζα κάποιο πρόβλημα θα ήταν εκεί και θα με βοηθούσε άμεσα και αποτελεσματικά και τον Ιωάννη Στρατάκο που εκτός από την επίλυση απορριών μου προσέφερε πάντα και περαιτέρω πληροφορίες. Ακόμα ένα ευχαριστώ και στον Δόκτορα Γιώργο Λεντάρη για τις παρεμβάσεις του και τη συμβολή του σε σημαντικά σημεία του έργου.

Φυσικά, να ευχαριστήσω τον αδερφό μου και τους γονείς μου, καθώς και τους φίλους μου που χωρίς τη στήριξη όλων αυτών όλα αυτά τα χρόνια δεν θα βρισκόμουν σε αυτή τη θέση.

Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη

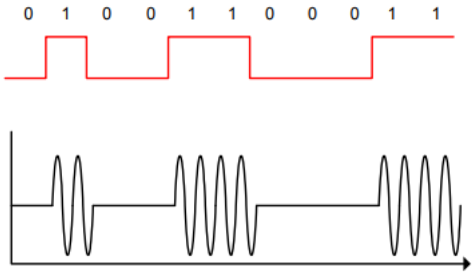
Εισαγωγή

Τα τελευταία χρόνια έχουν γίνει πολλές καινοτομίες και πρόοδοι στην ψηφιακή επικοινωνία και στα πολυμέσα. Σαν αποτέλεσμα, η ανάγκη για τηλεπικοινωνιακές εφαρμογές έχει αυξηθεί ραγδαία. Η ψηφιακή επικοινωνία έχει γίνει έτσι απαραίτητη για τη σημερινή κοινωνία, συνδέοντας τον κόσμο με ένα υψηλής ταχύτητας και αξιόπιστο δίκτυο.

Ένα οποιοδήποτε ψηφιακό σύστημα ξεκινάει με την περιγραφή του καναλιού, το οποίο περιλαμβάνει την λαμβανόμενη ισχύ (received power), το διαθέσιμο εύρος ζώνης (bandwith), στατιστικά θορύβου και άλλα προβλήματα όπως το ξεθώριασμα καναλιού (fading channel). Ο ρυθμός δεδομένων και η ανοχή στα σφάλματα είναι συγκεκριμένες και απαραίτητες προϋποθέσεις για ένα ψηφιακό σύστημα. Σε αυτό, το σήμα που πρόκειται να μεταδοθεί πρώτα κωδικοποιείται. Η διαδικασία κωδικοποίησης της πληροφορίας σε μορφή κατάλληλη για μετάδοση ονομάζεται διαμόρφωση. Αυτή όσο και το είδος της καθορίζουν αρκετά πράγματα όπως την αντοχή στο θόρυβο και την παραμόρφωση του καναλιού, το εύρος ζώνης που απαιτείται για τη μετάδοση της πληροφορίας, την πολυπλοκότητα των συστημάτων εκπομπής και λήψης, και άλλα.

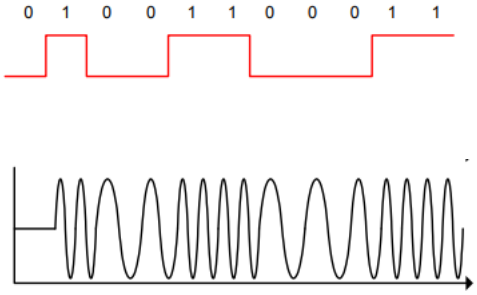
Είδη Διαμόρφωσης

Για την ψηφιακή διαμόρφωση υπάρχουν τα παρακάτω βασικά είδη:



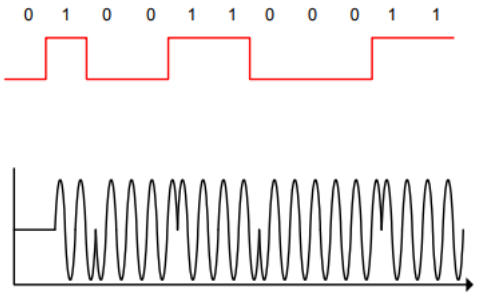
Σχήμα 1: Amplitude shift-keying (ASK)

- Στη μεταλλαγή μετατόπισης πλάτους (Amplitude Shift Keying) το πλάτος του φέροντος σήματος αλλάζει σε σχέση με τη πληροφορία και το υπόλοιπο παραμένει σταθερό.



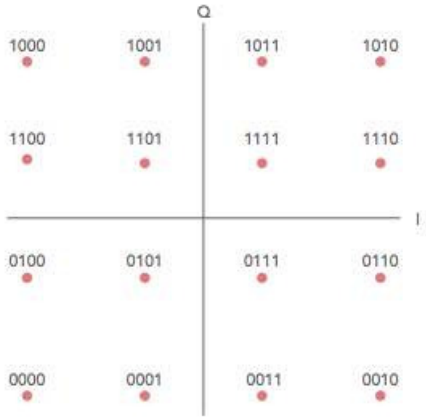
Σχήμα 2: Frequency shift-keying (FSK)

- Στη μεταλλαγή μετατόπισης πλάτους (Frequency Shift Keying) η συχνότητα αλλάζει σε σχέση με τη πληροφορία και το υπόλοιπο παραμένει σταθερό.



Σχήμα 3: Phase shift-keying (PSK)

- Στη μεταλλαγή Μετατόπισης Φάσης (Phase Shift Keying) η φάση αλλάζει σε σχέση με τη πληροφορία και το υπόλοιπο παραμένει σταθερό.



Σχήμα 4: Παράδειγμα QAM διαμόρφωσης 16 καταστάσεων

- Η διαμορφωση QAM στην οποία θα εστιάσουμε περισσότερο στην διπλωματική αυτή, χρησιμοποιεί τόσο το πλάτος όσο και τη φάση του φέροντος. Τα bits του ψηφιακού σήματος ομαδοποιούνται σε n σύμβολα δημιουργώντας 2^n συνδυασμούς για κάθε έναν από τους οποίους προβλέπεται ένα ζεύγος τιμών πλάτος-φάση. Στο σχήμα 4 φαίνονται οι 16 καταστάσεις σε μια διαμόρφωση των 16-QAM.

Προσθετικός Λευκός Γκαουσιανός Θόρυβος (AWGN)

Ως θόρυβος ορίζεται κάποιο ανεπιθύμητο είδος ενέργειας ηλεκτρική ή ηλεκτρομαγνητική που τείνει να αναμειχθεί κατά τη λήψη και αναπαραγωγή του. Αυτό έχει ως αποτέλεσμα την αλλοίωση του σήματος. Υπάρχουν αρκετά είδη θορύβου, αλλά στην παρούσα φάση θα επικεντρωθούμε στον Προσθετικό Λευκό Γκαουσιανό Θόρυβο (AWGN). Ο προσθετικός λευκός Γκαουσιανός θόρυβος χρησιμοποιείται για την ανάλυση των διαμορφώσεων σε παγκόσμιο μοντέλο καναλιού. Ο συγκεκριμένος θόρυβος υπάρχει πάντα ανεξάρτητα αν υπάρχουν ή όχι εμπόδια από άλλα κανάλια όπως περιορισμός του εύρους ή εξασθένιση και για αυτό χρησιμοποιείται για αναλύσεις των αποδόσεων ενός συστήματος.

Κωδικοποίηση Gray

Κατά την αποκωδικοποίηση ενός σήματος είναι πιθανό λόγω του θορύβου και της παραμόρφωσης που υφίσταται να ληφθεί λανθασμένο bit. Προκειμένου να ελαχιστοποιηθεί αυτός ο αριθμός λαθών χρησιμοποιείται η κωδικοποίηση Gray. Σύμφωνα με αυτήν και όπως φαίνεται και στο Σχήμα 4 όλα τα γειτονικά σύμβολα διαφέρουν μεταξύ τους κατά ένα bit. Έτσι, σε περίπτωση που ο δέκτης αποδιαμορφώσει λάθος το σύμβολο, θα δίνει μόνο ένα λάθος bit.

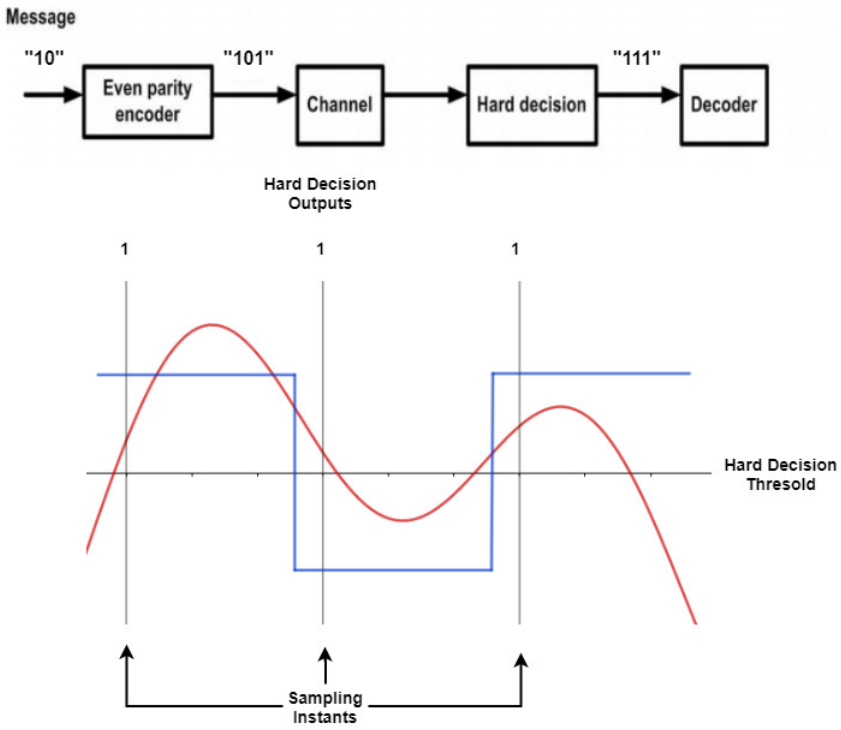
Τεχνικές Αποκωδικοποίησης

Μία πρόκληση στην διόρθωση λαθών που αναφέρθηκε είναι η αποκωδικοποίηση ενός συμβόλου που έχει παραμορφωθεί λόγω θορύβου. Τα δεδομένα πριν μεταδοθούν δέχονται μια κωδικοποίηση στην οποία προστίθενται bits στις κωδικολέξεις (codewords) που σχηματίζουν το μήνυμα. Στη συνέχεια το μήνυμα αυτό στέλνεται και αφού το λάβει ο δέκτης προσπαθεί να το αποκωδικοποιήσει για να πάρει το αρχικό μήνυμα που εστάλη. Για την κωδικοποίηση αυτή υπάρχουν δύο τεχνικές που ονομάζονται Hard Decision Decoding και Soft Decision Decoding.

Η πρώτη μέθοδος παίρνει ένα μπλοκ από bits από το κατώφλι του δέκτη και αποκωδικοποιεί κάθε bit θεωρώντας το ως σίγουρα 1 ή 0. Παίρνει τους ληφθέντες παλμούς και συγκρίνει τις τάσεις τους με τις τιμές κατωφλίου. Εάν μια τάση είναι μεγαλύτερη από την τιμή κατωφλίου, αποκωδικοποιείται ως 1 και αποκωδικοποιείται διαφορετικά ως 0.

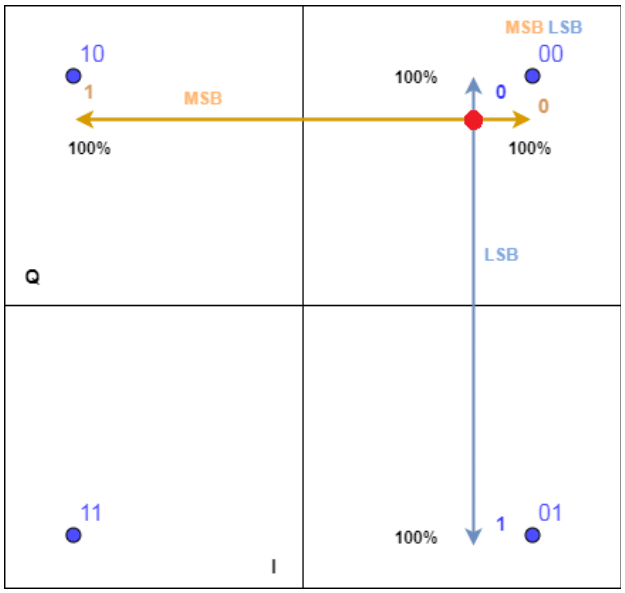
Η δεύτερη μέθοδος (Soft Decision Decoding) είναι μια κατηγορία αλγορίθμων που λαμβάνει μια ροή δυαδικών ψηφίων και τα αποκωδικοποιεί λαμβάνοντας υπόψη μια σειρά πιθανών τιμών

που μπορεί να πάρει. Δρα σύμφωνα με την αξιοπιστία του κάθε παλμού που λαμβάνει για να σχηματίσει καλύτερες εκτιμήσεις των δεδομένων εισόδου.



Σχήμα 5: Παράδειγμα ενός Hard Decision Decoder

Η συγκεκριμένη τεχνική χρησιμοποιεί μια μέθοδο που λέγεται Log Likelihood Ratio (LLR). Η μέθοδος αυτή λαμβάνει υπόψιν ένα συνόλο παραμέτρων και δυνατών αποτελεσμάτων και βγάζει την αντίστοιχη πιθανότητα για καθένα από αυτά τα αποτελέσματα.



Σχήμα 6: Παράδειγμα 4 σημείων αστερισμού

Κάθε μία από τις μπλε κουκκίδες είναι τα τέσσερα σημεία αστερισμού. Όταν μεταδίδεται ένα σύμβολο, υπάρχει ένας θόρυβος στο κανάλι που μεταβάλλει το αρχικό σήμα. Η κόκκινη κουκκίδα, επομένως, είναι το μετατοπισμένο σύμβολο. Όπως φαίνεται στο Σχήμα 6, το λιγότερο σημαντικό ψηφίο είναι 0 πάνω από τον κάθετο άξονα και 1 κάτω από αυτόν. Αυτό σημαίνει ότι το λαμβανόμενο σύμβολο έχει περισσότερες πιθανότητες να είναι 0. Ομοίως το περισσότερο σημαντικό ψηφίο είναι 0 δεξιά από τον οριζόντιο άξονα και 1 αριστερά του. Αυτό σημαίνει ότι το λαμβανόμενο σύμβολο έχει περισσότερες πιθανότητες να είναι 0. Άρα το σύμβολο είναι πιο πιθανό να είναι το ¨00¨.

Παρακάτω θα δούμε τον τρόπο υπολογισμού του LLR, καθώς και κάποιους αλγόριθμους που χρησιμοποιούν αυτή τη μέθοδο.

Αλγόριθμοι Αποκωδικοποίησης

Οι αλγόριθμοι αποκωδικοποίησης που υλοποιήθηκαν και θα αναλυθούν παρακάτω είναι τρεις Soft Decision και ένας Hard Decision.

Exact LLR

Ο Exact LLR είναι η πιο κοντινή προσέγγιση για τον ακριβή υπολογισμό του LLR και αποτελεί τη βάση για όλους τους άλλους LLR αλγορίθμους. Λόγω των σύνθετων συναρτήσεων όμως που χρειάζεται για τον υπολογισμό του, καθιστούν πολύπλοκη την υλοποίηση με μεγάλη κατανάλωση ενέργειας. Ο τύπος για τον υπολογισμό του LLR για το bit b μιας κωδικολέξης είναι:

$$LLR(b) = \ln\left(\frac{\sum_{s\in S_0} \exp -\frac{1}{\sigma^2}((x-s_x)^2+(y-s_y)^2))}{\sum_{s\in S_1} \exp -\frac{1}{\sigma^2}((x-s_x)^2+(y-s_y)^2))}\right) \tag{0.1}$$

όπου σ^2 η διακύμανση ή μεταβλητότητα του θορύβου, S_0 και S_1 είναι τα σημεία αστερισμού που έχουν bit 0 και 1 αντίστοιχα και s_x και s_y είναι οι I και Q συντεταγμένες του λαμβανόμενου συμβόλου αντίστοιχα.

Approx LLR

Η τεχνική αυτή υπολογίζει το LLR βρίσκοντας τα δύο πιο κοντινά σημεία από τον χάρτη αστερισμού που έχουν το bit τους 0 και 1. Στη συνέχεια γίνεται μια αφαίρεση μεταξύ αυτών των δύο σημείων με το αποτέλεσμα να καθορίζει αν το bit είναι πιο πιθανό να είναι 1 ή 0. Ο συγκεκριμένος αλγόριθμος αποτελεί μια προσέγγιση του πρώτου, καθώς από τον αρχικό τύπο μέσω της προσέγγισης του λογάριθμου αθροίσματος εκθετικών [20], καταλήγει στην παρακάτω μαθηματική έκφραση:

$$LLR(b) = -\frac{1}{\sigma^2}(\min_{s\in S_0}((x-s_x)^2+(y-s_y)^2) - \min_{s\in S_1}((x-s_x)^2+(y-s_y)^2)) \tag{0.2}$$

Piecewise LLR

Ο αλγόριθμος Piecewise LLR είναι μια περαιτέρω προσέγγιση του προηγούμενου αλγορίθμου, αφού για κάθε bit προκύπτει μια ξεχωριστή γραμμική συνάρτηση σύμφωνα με το [19]. Οι γραμμικές αυτές συναρτήσεις μπορούν να εκφραστούν όπως παρακάτω:

$$D_{I,k} = \begin{cases} y_I[i] & k = 1 \\ -|D_{I,k-1}| + d_{I,k} & k > 1 \end{cases} \tag{0.3}$$

$$D_{Q,k} = \begin{cases} y_Q[i] & k = 1 \\ -|D_{Q,k-1}| + d_{Q,k} & k > 1 \end{cases} \tag{0.4}$$

όπου τα $d_{I,k}$ και $d_{Q,k}$ υποδεικνύουν τη μισή απόσταση μεταξύ των επιμέρους ορίων των $b_{I,k}$ και $b_{Q,k}$, και y το λαμβανόμενο σύμβολο.

Maximum Likelihood Detection

Ο τελευταίος αλγόριθμος αποτελεί Hard Decision, που σημαίνει ότι σε αντίθεση με τους προηγούμενους δεν υπολογίζει LLR. Αντιθέτως, χρησιμοποιώντας μια μη γραμμική μέθοδο ανίχνευσης συμβόλων εκτιμά τα bit κάθε κωδικολέξης. Σύμφωνα με το [22], το εκτιμώμενο σύμβολο της M-QAM διαμόρφωσης μπορεί να γραφτεί ως:

$$x = \sum_{n=1}^{log_2\sqrt{M}} c_n \tag{0.5}$$

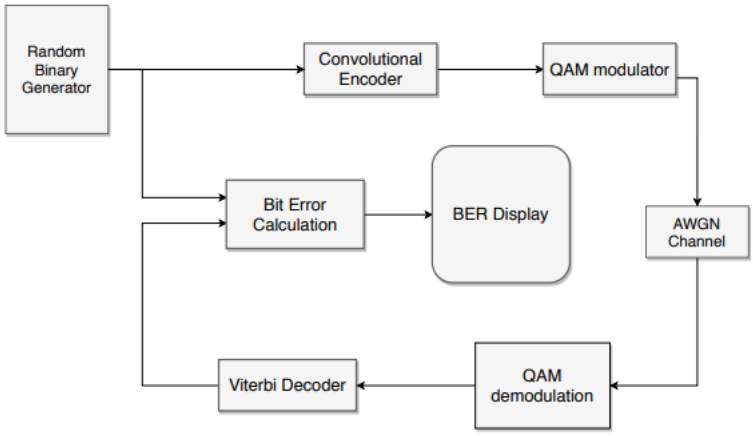
όπου

$$c_n = \sqrt{\frac{3M}{M-1}} 2^{-n} e^{jg(y - \sum_{m=1}^{n-1} c_m)} \tag{0.6}$$

Η μέθοδος αυτή πετυχαίνει ακριβώς την ίδια απόδοση με το συμβατικό αλγόριθμο του Max Likelihood αλλά συγχρόνως και μια υλοποίηση με αρκετά μειωμένους πόρους.

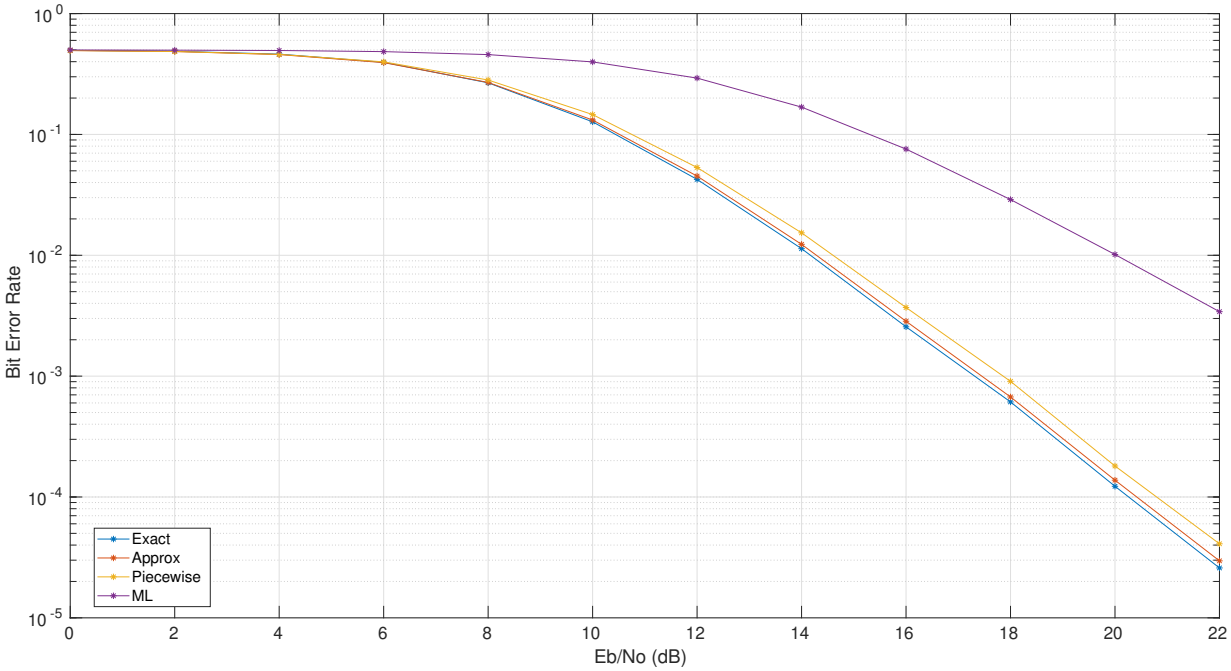
Ανάλυση σε επίπεδο Matlab

Προκειμένου να μελετηθεί η συνολική απόδοση των προηγούμενων αλγορίθμων χρειαζόταν να γίνει και μια μελέτη γύρω από το Bit Error Rate (BER). Για το σκοπό αυτό σχεδιάστηκε ένα σύστημα όπως φαίνεται στο Σχήμα 7, ώστε να παρατηρήσουμε τα λάθη που βγάζει κάθε αλγόριθμος δεδομένων συνθηκών και για ένα εύρος θορύβου.

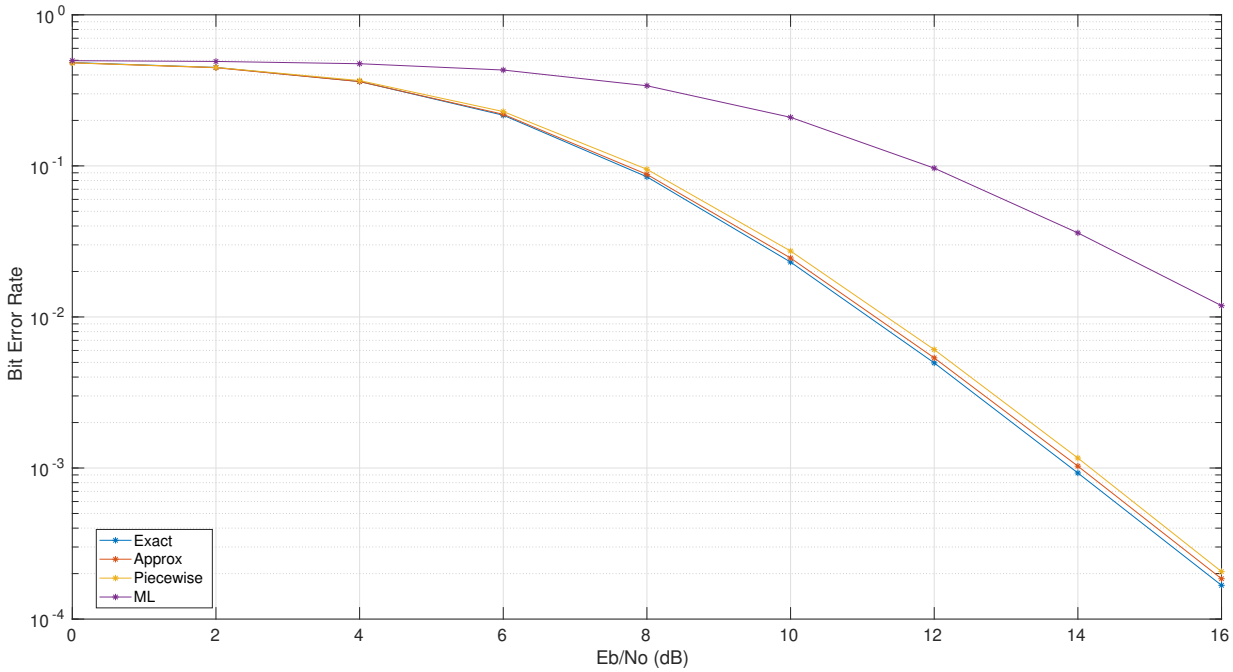


Σχήμα 7: Σχηματικό διάγραμμα για QAM αποδιαμόρφωση χρησιμοποιώντας Viterbi Decoder

Στις προσομοιώσεις που έγιναν ο θόρυβος ήταν Προσθετικός Λευκός Γκαουσιανός Θόρυβος (AWGN) με διακύμανση $1/\sqrt{10^{(E_b/N_0)/10} \cdot log_2 M}$. Επίσης η κωδικοποίηση έγινε με generator polynomial (133,171) και constraint length 7. Τα αποτελέσματα των τεσσάρων προαναφερθέντων αλγορίθμων φαίνονται παρακάτω.



Σχήμα 8: Σύγκριση των 4 αλγορίθμων για 256-QAM



Σχήμα 9: Σύγκριση των 4 αλγορίθμων για 64-QAM

Όπως φαίνεται και από τα δύο σχήματα παραπάνω ο ακριβής υπολογισμός του LLR δίνει το μικρότερο αριθμό λαθών στα bits, όπως περιμέναμε. Ακολουθεί ο προσεγγιστικός τύπος και μετά η μέθοδος με τις γραμμικές συναρτήσεις. Τέλος, παρατηρούμε ότι ο αλγόριθμος Hard Decision δίνει τα περισσότερα λάθη και από τους τέσσερις. Αυτός είναι και ο λόγος που στη συνέχεια θα επικεντρωθούμε στους υπόλοιπους τρεις αλγορίθμους, σε αυτούς δηλαδή που υπολογίζουν το LLR προκειμένου να ανιχνεύσουν το αρχικό σύμβολο.

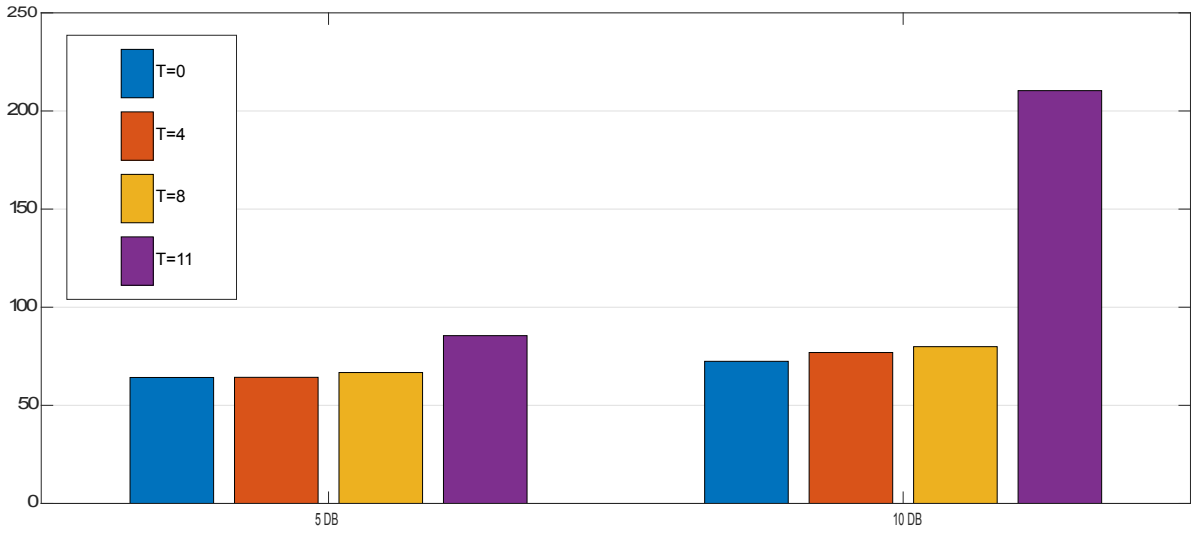
Προσεγγιστικές τεχνικές για τον υπολογισμό του LLR

Η ανάγκη για μείωση των πόρων που καταναλώνει ένα κύκλωμα είναι μεγάλη και σημαντική. Για τον λόγο αυτό, σε αυτό το σημείο εισάγουμε κάποιες προσεγγιστικές τεχνικές που θα βοηθήσουν στην απλοποίηση των κυκλωμάτων για τον υπολογισμό του LLR. Παρακάτων παρουσιάζονται οι προσεγγιστικές τεχνικές που εφαρμόστηκαν και υλοποιήθηκαν αργότερα και στο FPGA. Οι μετρήσεις αυτές πάρθηκαν από προσομοιώσεις στο Matlab.

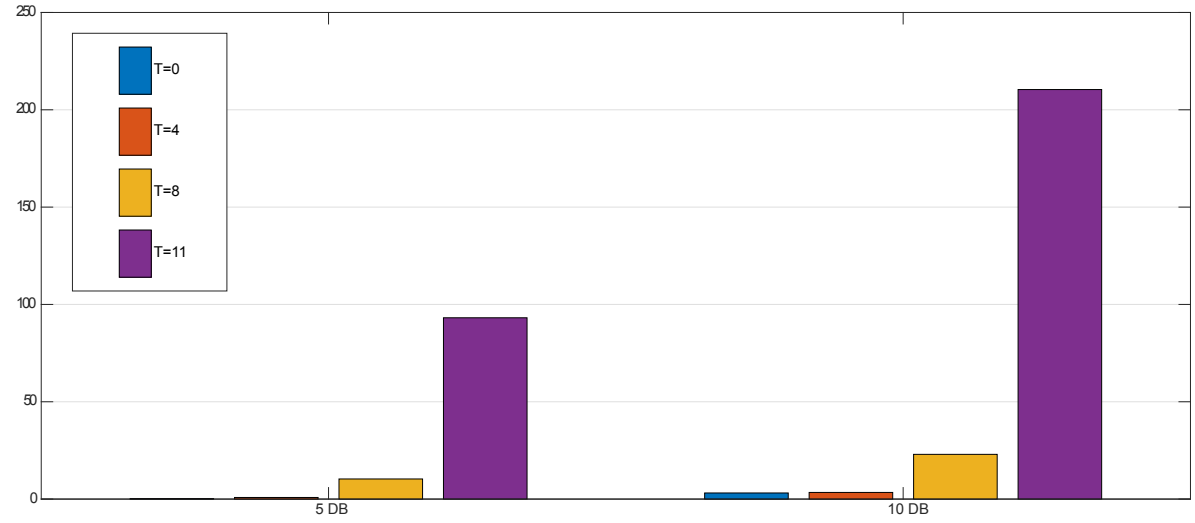
Περικοπή των bits σε fixed-point αριθμητική

Η αριθμητική που χρησιμοποιούμε για τα αποτελέσματα αυτά είναι fixed-point όπου οι αρχικές είσοδοι αποτελούνται από 16 bits με 14 bits στο δεκαδικό μέρος. Η τεχνική αυτή λοιπόν αναφέρεται στο κόψιμο T ψηφίων, ώστε να μικρύνει η πολυπλοκότητα των ενδιάμεσων πράξεων. Παράλληλα, για να μην χαλάσει η ακρίβεια των αποτελεσμάτων, η περικοπή αυτή γίνεται

από τα λιγότερο σημαντικά bits. Παρακάτων, παρουσιάζεται η απόλυτη σχετική απόκλιση των τελικών LLR σε σχέση με το T συγκριτικά με τους ακριβείς αλγόριθμους (Full Precision) για διαφορετικές περιπτώσεις διαμόρφωσης, αλγόριθμου και θορύβου.

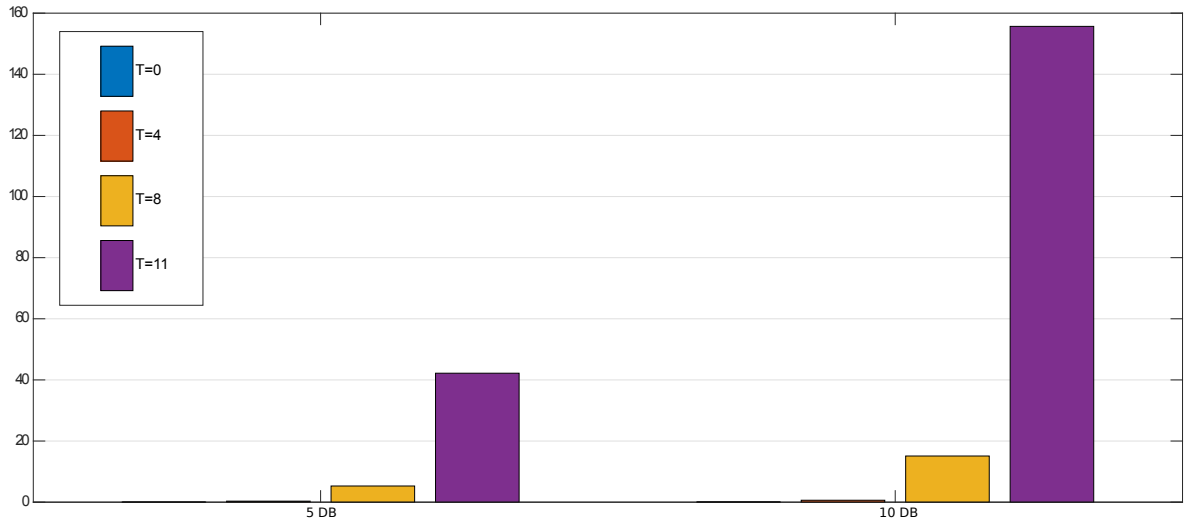


Σχήμα 10: Απόλυτη σχετική απόκλιση του LLR για 64-QAM Exact LLR



Σχήμα 11: Απόλυτη σχετική απόκλιση του LLR για 64-QAM Approx LLR

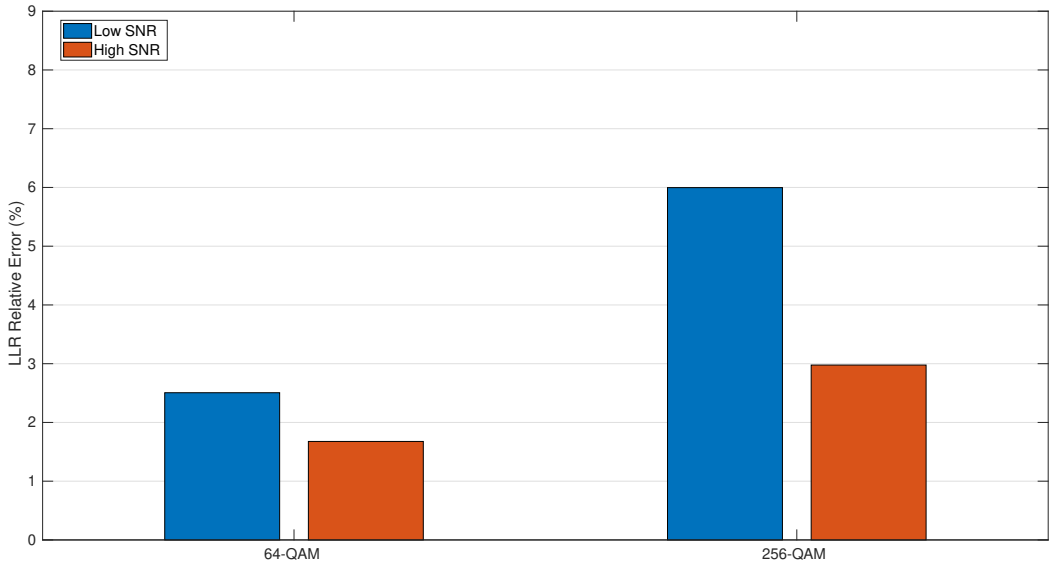
Όπως φαίνεται από τα Σχήματα 10,11,12 ο Exact LLR είναι αυτός που παρουσιάζει το μεγαλύτερο σχετικό σφάλμα. Αυτό οφείλεται λόγω των πολύπλοκων πράξεων που έχει (λογάριθμοι και εκθετικά) και την δυσκολία αναπαράστασής τους σε αριθμητική fixed point. Οι υπόλοιποι δύο αλγόριθμοι παρουσιάζουν παρόμοιες διαφορές με τον Piecewise να είναι ελαφρώς καλύτερος.



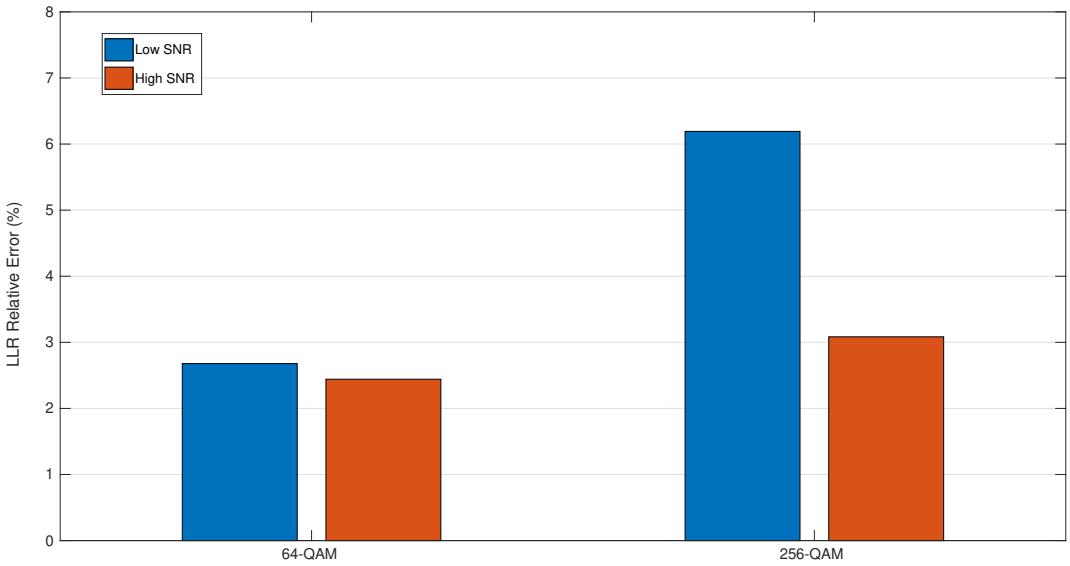
Σχήμα 12: Απόλυτη σχετική απόκλιση του LLR για 64-QAM Piecewise LLR

Πολλαπλασιαστής Radix

Η τεχνική αυτή αφορά την αντικατάσταση των πολλαπλασιαστών του εκάστοτε κυκλώματος με πολλαπλαστιαστές Radix [9]. Η Radix κωδικοποίηση προσφέρει συχνά μερική μείωση των πόρων που χρειάζεται ένα κύκλωμα οδηγώντας σε εξοικονόμηση ενέργειας και σε μείωση της καθυστέρησης. Στη συνέχεια φαίνεται η σύγκριση των full-precision αλγορίθμων με αυτούς που χρησιμοποιούν τον συγκεκριμένο πολλαπλασιαστή. Τα αποτελέσματα αφορούν μόνο τον Exact και Approx LLR, καθώς ο τρίτος δεν χρησιμοποιεί πολλαπλασιαστές.



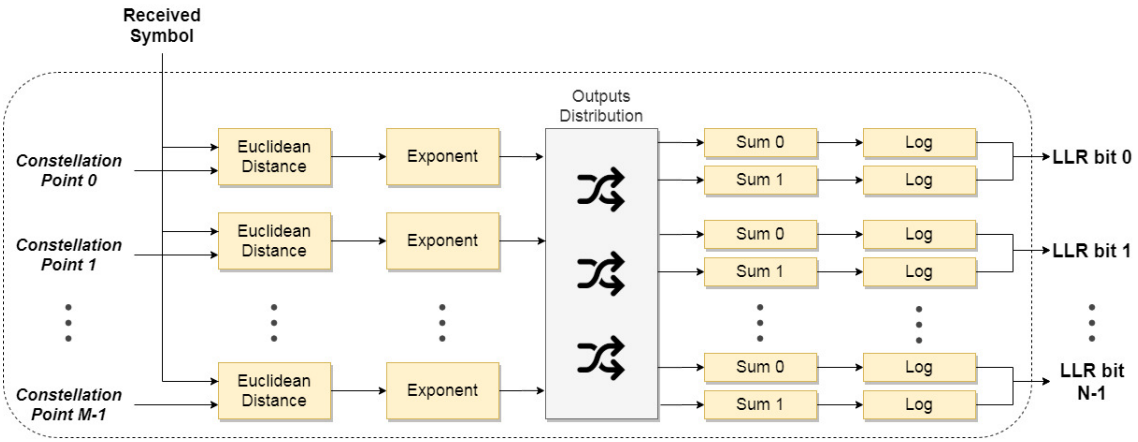
Σχήμα 13: Απόλυτη σχετική απόκλιση του LLR για 64-QAM και 256-QAM Exact Algorithm



Σχήμα 14: Απόλυτη σχετική απόκλιση του LLR για 64-QAM και 256-QAM Approx Algorithm

Αρχιτεκτονική Αλγορίθμων

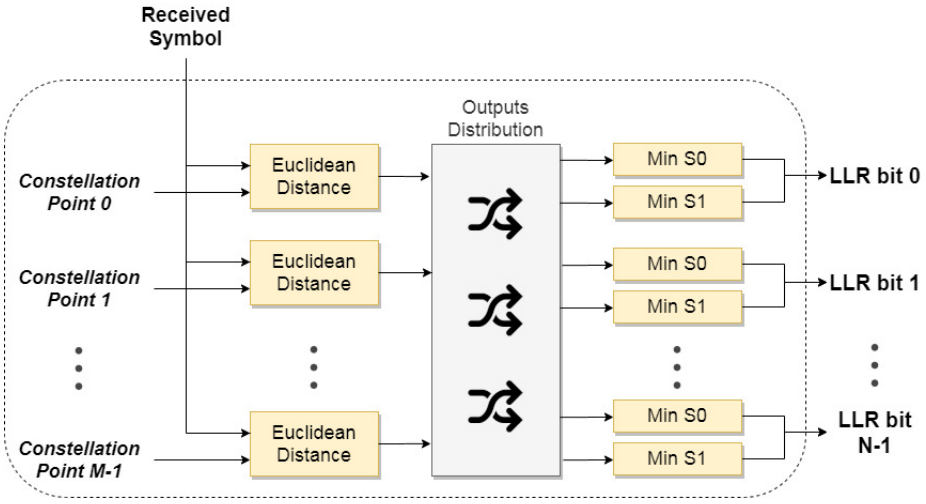
Στο σημείο αυτό, αφού έγινε η ανάλυση και σύγκριση των αλγορίθμων σε επίπεδο Matlab, θα επικεντρωθούμε στην υλοποίηση τους στη σχεδιαστική γλώσσα VHDL. Για το λόγο αυτό, οι αλγόριθμοι που επιλέχτηκαν να υλοποιηθούν ήταν οι τρεις soft decision(Exact LLR, Approx LLR, Piecewise LLR), καθώς έχουν καλύτερη απόδοση από τους Hard Decision. Παρακάτω περιγράφεται η αρχιτεκτονική του κάθε αλγορίθμου που υλοποιήθηκε.



Σχήμα 15: Σχηματικό διάγραμμα της υλοποίησης του Exact LLR

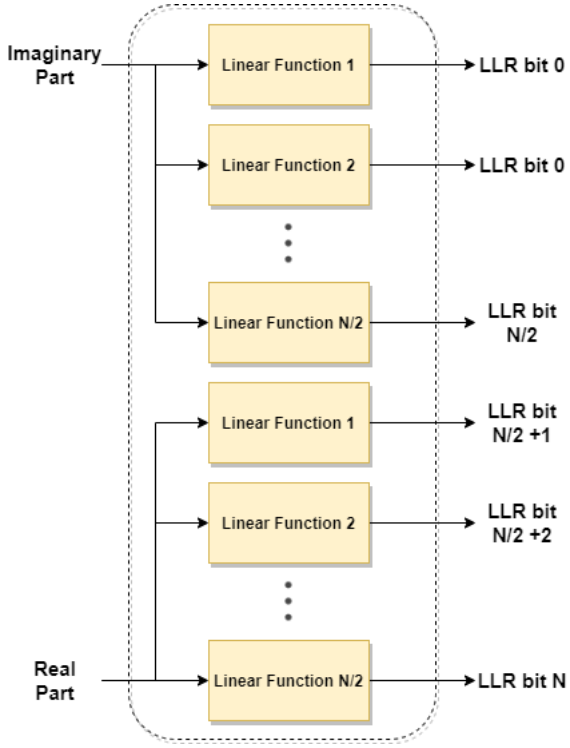
Η συγκεκριμένη μέθοδος υπολογίζει αρχικά τις αποστάσεις από το λαμβανόμενο σύμβολο και τα M σημεία αστερισμού και τα περνάει στο κομμάτι του εκθετικού το οποίο υπολογίζει

την εκθετική τιμή τους. Αφού κάνει ανακατανομή των M αποτελεσμάτων αυτών, υπολογίζει τους λογάριθμους των $log_2 M$ αθροισμάτων όπως ορίζει ο αρχικός τύπος.



Σχήμα 16: Σχηματικό διάγραμμα της υλοποίησης του Approx LLR

Η αρχιτεκτονική αυτού του αλγόριθμου μοιάζει αρκετά με την προηγούμενη αφού είναι και μια κοντινή προσέγγιση του ακριβή τύπου. Η διαφορά είναι ότι εδώ παραλείπονται οι σύνθετες πράξεις των λογαριθμικών και εκθετικών και αντικαθίστανται από μια διαδικασία που βρίσκει τις ελάχιστες αποστάσεις όπως ορίζει ο τύπος του Approx LLR.



Σχήμα 17: Σχηματικό διάγραμμα της υλοποίησης του Piecewise LLR

Η τρίτη μέθοδος του Piecewise LLR αποτελείται από $N = log_2 M$ προσεγγιστικές συναρτήσεις. Οι μισές από αυτές χρησιμοποιούν το φανταστικό μέρος του λαμβανόμενου συμβόλου και οι υπόλοιπες το πραγματικό. Κάθε γραμμική συνάρτηση προκύπτει όπως περιγράφτηκε στις προηγούμενες εξισώσεις.

Υλοποίηση λογαριθμικού και εκθετικού

Όπως αναφέρθηκε προηγουμένως ο Exact LLR για τον υπολογισμό του LLR για κάποιο bit χρειάζεται να κάνει εκθετικούς και λογαριθμικούς υπολογισμούς. Υπάρχουν αρκετοί τρόποι να υλοποιηθούν οι μαθηματικές πράξεις αυτές σε γλώσσα VHDL. Στην διπλωματική αυτή επιλέχτηκαν να γίνουν μέσω της μεθόδου CORDIC και του αλγορίθμου Remez. Η πρώτη τεχνική αφορά την περιστροφή διανύσματος βήμα προς βήμα με δεδομένη γωνία [13]. Βασίζεται σε υπολογισμούς που χρησιμοποιούν μόνο καταχωρητές ολίσθησης και προσθέσεις και όχι πολλαπλασιασμούς που πιάνουν αρκετούς πόρους σε ένα κύκλωμα. Η δεύτερη μέθοδος αφορά την εύρεση μιας κοντινής ορθολογικής προσέγγισης της εκάστοτε συνάρτησης (εκθετικό ή λογαριθμικό) [18]. Η προσέγγιση αυτή αποτελεί πολυώνυμο ενός επιθυμητού βαθμού και προσφέρει μια απλούστερη και γρηγορότερη υλοποίηση με αντάλλαγμα κάποια απώλεια στην ακρίβεια των αποτελεσμάτων.

Από τα προηγούμενα προκύπτει μια επιτακτική ανάγκη να δούμε κατά πόσο όλες οι προσεγγιστικές τεχνικές που αναλύθηκαν επηρεάζουν τα τελικά αποτελέσματα του κάθε αλγόριθμου. Και με τη σειρά τους αυτά κατά πόσο μεταβάλλουν τον αριθμό λαθών που ανιχνεύονται σε μια τηλεπικοινωνιακή αλυσίδα.

Αποτελέσματα από την σχεδίαση και υλοποίηση των αλγορίθμων

Για την ολοκλήρωση της εξερεύνησης των αλγορίθμων συνδιαστικά με τις προσεγγιστικές τεχνικές, είναι αναγκαίο να γίνει μελέτη και στα αποτελέσματα που προκύπτουν από την υλοποίηση τους. Αξίζει να αναφερθεί ότι η σχεδίαση έγινε με τη χρήση του Vivado Design Suite 2019.2 και η πλατφόρμα που χρησιμοποιήθηκε ήταν η Zynq UltraScale + MPSoC ZCU106. Εξίσου σημαντικό είναι ακόμα το ότι τα παρακάτω αποτελέσματα δεν αφορούν μόνο τη χρήση των FPGA αλλά και κυκλωμάτων ASIC. Για αυτό το λόγο γίνεται και μια μελέτη χωρίς τη χρήση DSPs, ώστε οι διαφορές των πόρων να είναι πιο ξεκάθαρες.

Στο σημείο αυτό ορίζονται κάποιες μετρικές που θα χρησιμοποιηθούν. Είναι το απόλυτο σχετικό σφάλμα στις LLR τιμές, όπως ορίστηκε και παραπάνω, και η αντιστροφή της πολικότητάς τους. Η τελευταία παίζει σημαντικό ρόλο, καθώς το πρόσημο των τιμών είναι αυτό που καθορίζει αν ένα bit είναι πιο πιθανό να είναι 1 ή 0, αφού περαστεί από τον αποκωδικοποιητή ανίχνευσης λαθών.

Exact LLR

Πίνακας 1: Αποτελέσματα για 64-QAM Exact LLR αποδιαμόρφωση με υλοποίηση Cordic και Polynomial συγκριτικά με τους ακριβείς αλγορίθμους του Matlab

	64-QAM			
	CORDIC		**POLYON**	
	10DB	**15DB**	**10DB**	**15DB**
LLR Reversal Polarity	0.15%	0.17%	0.029%	0.037%
LLR Relative Error	47.25%	79.42%	46.25%	77.94%

DSP	0%	0%
LUTS	53.12%	63.51%
FF	17.73%	37.78%

Εδώ βλέπουμε ότι ο Exact LLR παρουσιάζει τεράστικα διαφορά από την ακριβή full-precision μορφή του. Αυτό οφείλεται στις πολύπλοκες πράξεις των εκθετικών και λογαριθμικών και τη δυσκολία τους να αναπαρασταθούν σε fixed-point αριθμητική.

Approx LLR

Πίνακας 2: Αποτελέσματα για 64 και 256 QAM Approximate LLR αποδιαμόρφωση συγκριτικά με τους ακριβείς αλγορίθμους του Matlab

	64-QAM		**256-QAM**	
	10DB	**15DB**	**15DB**	**20DB**
LLR Reversal Polarity	0.00075%	0.000083%	0.0019%	0.00013%
LLR Relative Error	0.14%	0.04%	0.09%	0.023%
BER Variation	1.67×10^{-6}	0	3.75×10^{-6}	1.00×10^{-6}

DSP	0%	0%
LUTS	24.09%	97.28%
FF	4.69%	22%

Piecewise LLR

Πίνακας 3: Αποτελέσματα για 64 και 256 QAM Approximate LLR αποδιαμόρφωση συγκριτικά με τους ακριβείς αλγορίθμους του Matlab

	64-QAM		**256-QAM**	
	10DB	**15DB**	**15DB**	**20DB**
LLR Reversal Polarity	0.0013%	0.00025%	0.0036%	0,00025
LLR Relative Error	0.04%	0.015%	0.073%	0.024%
BER Variation	3.33×10^{-6}	1.00×10^{-6}	2.00×10^{-5}	0

DSP	0%	0%
LUTS	0.07%	0.14%
FF	0.05%	0.09%

Οι παραπάνω πίνακες παρουσιάζουν κάποιες διαφορές σχετικά με τους αλγόριθμους που υλοποιήθηκαν σε VHDL και στους ακριβείς του Matlab. Από αυτά φαίνεται εύκολα ότι ο Exact LLR απέχει αρκετά από την ακριβή αναπαράστασή του. Φαίνεται επίσης ότι οι άλλοι δύο αλγόριθμοι είναι αρκετά κοντά με τις full-precision μορφές τους, το οποίο οδηγεί σε μικρή διαφορά σε επίπεδο bit error rate (τάξης 10^{-6}).

Αποτελέσματα αλγορίθμων με την προσεγγιστική τεχνική Truncation (περικοπή bits)

Exact LLR

Πίνακας 4: Αποτελέσματα για 64-QAM Exact LLR αποδιαμόρφωση με υλοποίηση Cordic συγκριτικά με τους ακριβείς αλγορίθμους του Matlab

	64-QAM CORDIC					
	10DB			**15DB**		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.17%	0.50%	4.69%	0.15%	0.21%	5.60%
LLR Relative Error	47.25%	60.09%	197.10%	79.42%	80.20%	86.9%

DSP	0%	0%	0%
LUTS	53.12%	40.21%	35.55%
FF	17.73%	16.85%	16.24%

Πίνακας 5: Αποτελέσματα για 64-QAM Exact LLR αποδιαμόρφωση με υλοποίηση Polynomial συγκριτικά με τους ακριβείς αλγορίθμους του Matlab

	64-QAM POLYON					
	10DB			**15DB**		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.029%	0.50%	4.73%	0.037%	0.052%	2.02%
LLR Relative Error	46.25%	60.00%	200.00%	77.94%	78.74%	99.7%

DSP	0%	0%	0%
LUTS	63.51%	42.2%	27.3%
FF	37.78%	3.68%	3.21%

Στους πίνακες αυτούς παρατηρούμε και πάλι μεγάλα σφάλματα τόσο στην απόκλιση όσο και στην διαφορά προσήμου. Σαφώς, υπάρχει μεγάλη μείωση των λογικών μπλοκ όσο αυξάνεται το T, αλλά τα σφάλματα που προαναφέρθηκαν παραμένουν μεγάλα και κάνουν τον Exact LLR έναν αλγόριθμο ακατάλληλο για υλοποίηση σε ενα τηλεποικινωνιακό σύστημα.

Approx LLR

Πίνακας 6: Αποτελέσματα για 256-QAM Approx LLR αποδιαμόρφωση με χρήση Truncation συγκριτικά με τον ακριβή αλγόριθμο του Matlab

	256-QAM					
	15DB			**20DB**		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.0019%	0.76%	8.9%	0.00013%	0.075%	7.35%
LLR Relative Error	0.09%	0.45%	219%	0.023%	8.6%	74.99%
BER Variation	3.75×10^{-6}	3.63×10^{-5}	0.0045	1.00×10^{-6}	5.00×10^{-6}	3.21×10^{-4}

DSP	0%	0%	0%
LUTS	97.28%	38.64%	21.2%
FF	22%	11.5%	7.85%

Πίνακας 7: Αποτελέσματα για 64-QAM Approx LLR αποδιαμόρφωση με χρήση Truncation συγκριτικά με τον ακριβή αλγόριθμο του Matlab

	64-QAM					
	10DB			**15DB**		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.00075%	0.48%	4.64%	0.000083%	0.044%	1.94%
LLR Relative Error	0.14%	15.28%	142.94%	0.04%	5.31%	44.63%
BER Variation	1.67×10^{-6}	1.30×10^{-5}	0.0013	0	3.33×10^{-7}	1.52×10^{-5}

DSP	0%	0%	0%
LUTS	24.09%	8.79%	4.48%
FF	4.69%	2.32%	2.02%

Στους πίνακες αυτούς βλέπουμε πολύ μικρότερα νούμερα από τον Exact LLR. Μέχρι και για $T = 8$ τα σφάλματα είναι ανεκτά και όσον αφορά τους σχεδιαστικούς πόρους παρατηρούμε ότι έχουμε μια τεράστια μείωση από 97.28% σε 38.64% στο 256-QAM, ενώ από 24.09% σε 8.79% στο 64-QAM.

Piecewise LLR

Πίνακας 8: Αποτελέσματα για 256-QAM Piecewise LLR αποδιαμόρφωση με χρήση Truncation συγκριτικά με τον ακριβή αλγόριθμο του Matlab

	256-QAM					
	15DB			20DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0,0036%	1,65%	12,99%	0,00025%	0,35%	12,19%
LLR Relative Error	0,073%	38,75%	185,22%	0,024%	11,92%	71,36%
Bit Error Variation	2,00E-05	4,38E-05	0,071	0	1,25E-06	0,039

DSP	0%	0%	0%
LUTS	0,14%	0,05%	0,03%
FF	0,09%	0,05%	0,03%

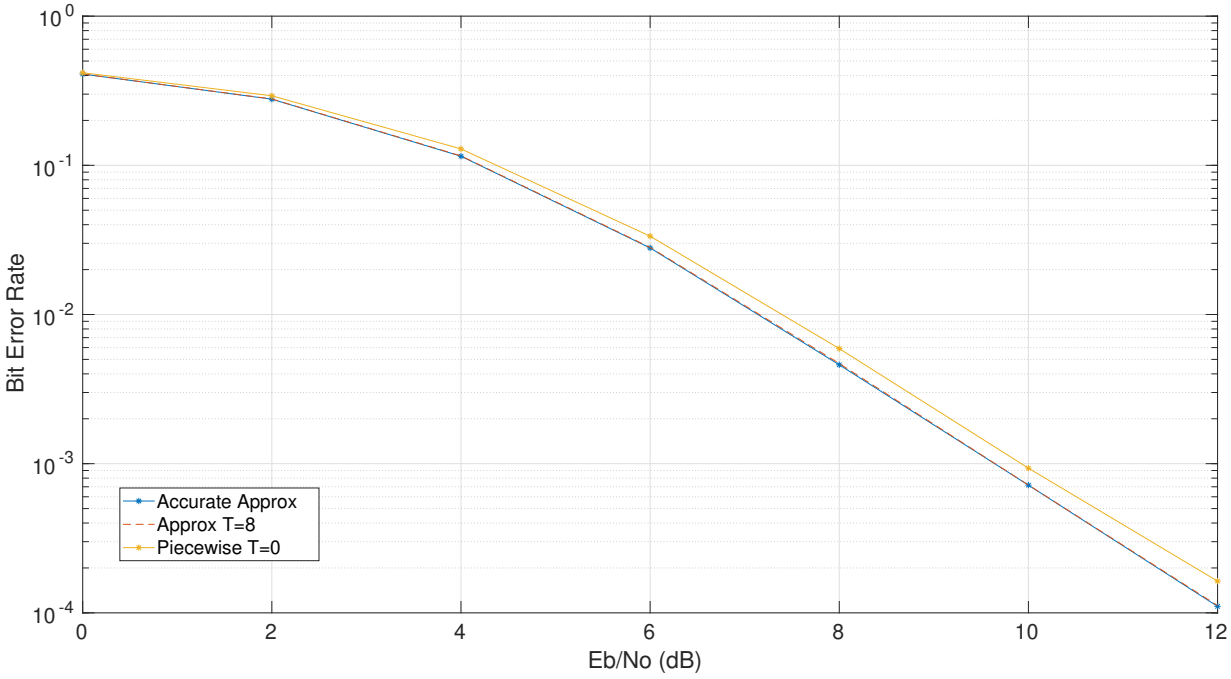
Πίνακας 9: Αποτελέσματα για 64-QAM Piecewise LLR αποδιαμόρφωση με χρήση Truncation συγκριτικά με τον ακριβή αλγόριθμο του Matlab

	64-QAM					
	10DB			15DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0,00125%	0,71%	6,39%	0,00025%	0,072%	6,21%
LLR Relative Error	0,04%	13,14%	93,54%	0,015%	5,64%	39,37%
Bit Error Variation	3,33E-06	1,00E-04	0,0045	1,00E-06	1,83E-06	1,07E-04

DSP	0%	0%	0%
LUTS	0,07%	0,03%	0,02%
FF	0,05%	0,03%	0,02%

Από αυτούς τους πίνακες μπορούμε να συμπεράνουμε ότι η μέθοδος Truncation αποδίδει χειρότερα από ότι στον Approx LLR, καθώς τα σχετικά σφάλματα είναι μεγαλύτερα. Επιπλέον, παρατηρούμε μικρότερη μείωση στα λογικά μπλοκ, αλλά οι σχεδιαστικοί πόροι που απαιτούνται παραμένουν ελάχιστοι. Έτσι, ενδιαφέρον παρουσιάζει να δούμε και πώς ερμηνέυονται αυτές οι διαφορές και αυτά τα σφάλματα σε επίπεδο BER.

Για όλους του λόγους που αναφέρθηκαν μέχρι τώρα ο αλγόριθμος που επιλέχτηκε και προτιμήθηκε για το σύστημά μας και περαιτέρω εφαρμογή του πολλαπλασιαστή Radixήταν ο Approx LLR. Η μέθοδος αυτή παρόλο που απαιτεί περισσότερους πόρους μπορεί να δεχτεί και περαιτέρω μειώσεις σε αντίθεση με τον Piecewise LLR και αποτελέι την πιο ακριβή προσσέγιση του υπολογισμού του LLR. Επομένως διαθέτει και καλύτερη απόδοση σε BER όπως φαίνεται και στο Σχήμα 18.



Σχήμα 18: Σύγκριση των BER αποδόσεων μεταξύ των Approx T0, Approx T8, Piecewise T0 για 64-QAM

Αποτελέσματα του Approx LLR με την προσεγγιστική τεχνική Radix

Πίνακας 10: Αποτελέσματα για 64-QAM Approx LLR αποδιαμόρφωση με χρήση Radix συγκριτικά με τον ακριβή αλγόριθμο του Matlab

	64-QAM			
	10DB		**15DB**	
	T=0	**K=10**	**T=0**	**K=10**
LLR Reversal Polarity	0.00075%	0.0058%	0.000083%	0.0013%
LLR Relative Error	0.14%	2.92%	0.04%	2.56%

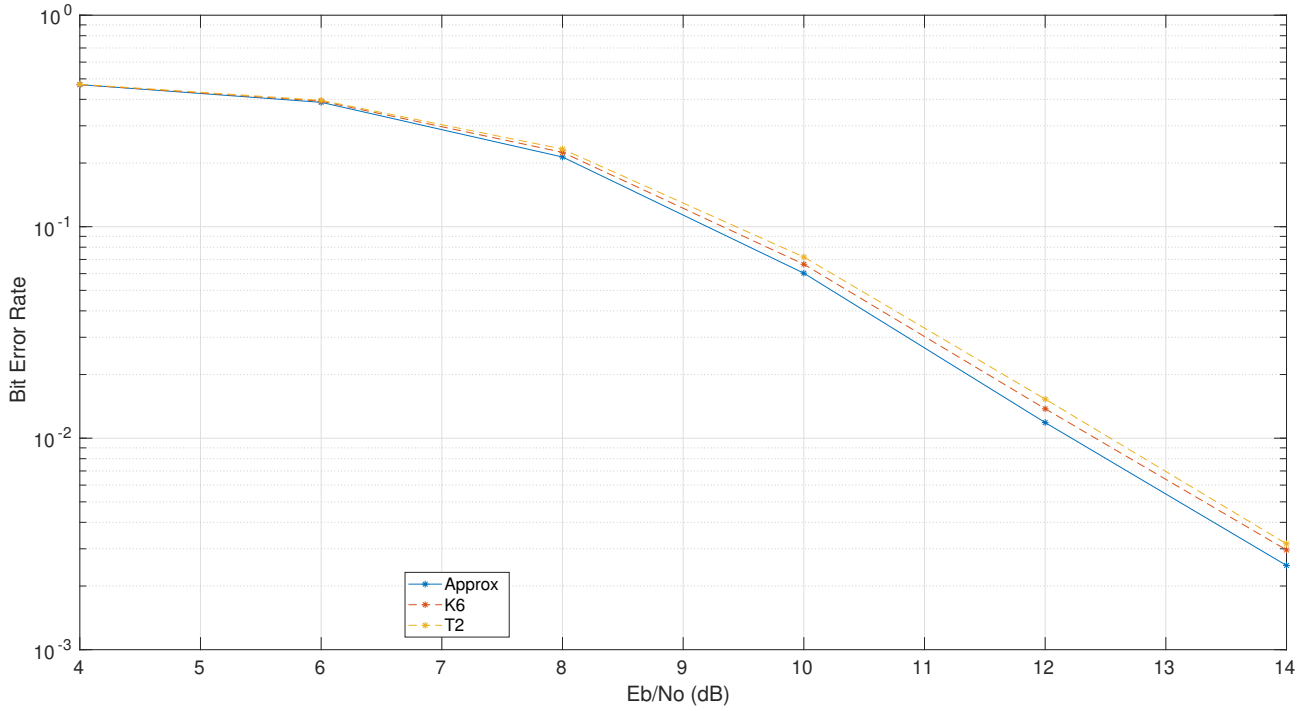
DSP	0%	0%
LUTS	24.09%	16.37%
FF	4.69%	11.66%
LUTRAM	0%	1.13%

Με την τεχνική Radix παρατηρείται και εδώ μια αξιοσημείωτη μείωση στους πόρους που χρειάζεται για να υλοποιηθεί ο Approx LLR με ελάχιστη αύξηση των LLR τιμών. Όπως αναφέρθηκε και στα προηγούμενα είναι εξίσου σημαντικό να μεταφράσουμε την (απόλυτη) σχετική αύξηση αυτή σε επίπεδο bit error rate. Επειδή όμως με την αρχική μας υπόθεση των 14 bits στο δεκαδικό μέρος η ακρίβεια των πράξεων παραμένει μεγάλη, δοκιμάστηκαν είσοδοι

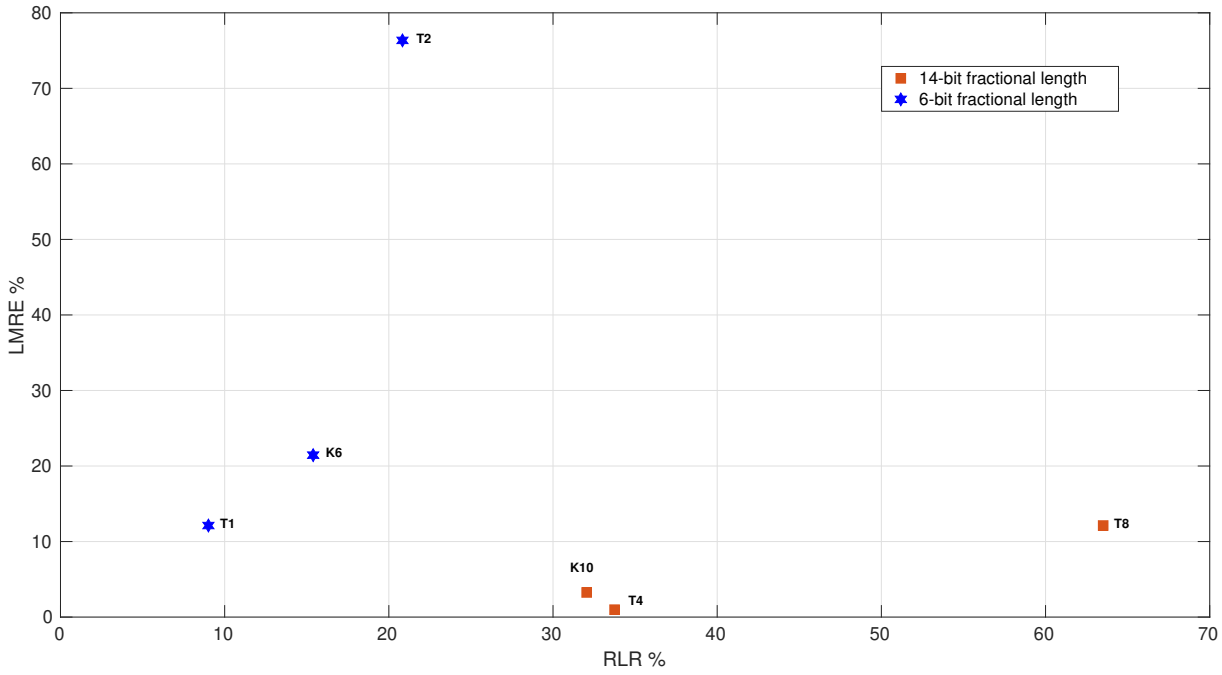
με 6 bits δεκαδικά. Έτσι, θα προκύψει ένα πιο ολοκληρωμένο συμπέρασμα για το ποιά προσεγγιστική τεχνική είναι καλύτερη και υπο ποιές συνθήκες. Τα αποτελέσματα που προέκυψαν τόσο για τις LLR τιμές όσο και για την απόδοση σε BER παρουσιάζονται παρακάτω.

Πίνακας 11: Αποτελέσματα για 64-QAM Approx LLR αποδιαμόρφωση στα 10db με 6 bits στο δεκαδικό μέρος. Τα LLR συγκρίνονται με τους αλγορίθμους του Matlab, ενώ τα LUTs με αυτούς για T=0.

	64-QAM		
	K=6	T=1	T=2
LLR Reversal Polarity	0.58%	1.18%	1.78%
LLR Relative Error	20.91%	11.23%	54.2%
Relative LUTS-gain	15.36%	9.01%	20.83%



Σχήμα 19: Σύγκριση των BER αποδόσεων μεταξύ των Approx T2, Radix K6 και Full-Precision Approx.



Σχήμα 20: RLR-LMRE tradeoff για τους προσεγγιστικούς αλγορίθμους Approx LLR που εξετάστηκαν

Έχοντας ως είσοδο fixed-point αριθμούς με 6 bits στο δεκαδικό μέρος, φαίνεται πράγματι ότι η τεχνική με τους πολλαπλασιαστές Radix αποδίδει καλύτερα σε bit error rate. Επομένως, για να εξεταστούν σε αυτό το σημείο τα πλεονεκτήματα της αντικατάστασης των ακριβών αλγορίθμων από τους προσεγγιστικούς τους, εισάγονται δύο μετρικές. Η LLR Mean Relative Error (LMRL) και η Relative LUTs Reduction (RLR). Η πρώτη αφορά το μέσο σχετικό σφάλμα των τιμών του LLR όπως ορίστηκε και στα προηγούμενα και η δεύτερη τη σχετική μείωση στους καταναλώσιμους πόρους. Το Σχήμα 20 παρουσιάζει τις διαφορετικές τεχνικές που εφαρμόστηκαν υπό διαφορετικό αριθμό bit στην είσοδο. Έτσι, λαμβάνοντας υπόψιν κάποιες παραμέτρους μπορεί κανείς να εξάγει τη βέλτιστη λύση από την άποψη οικονομική σχεδίαση - χαμηλό σφάλμα.

Chapter 1

Introduction

In the past decade there have been numerous innovations and advances in communication and multimedia. As a result, the need for digital communication for numerous applications has grown rapidly. Applications for digital communication include television, telephone, digital cinema, radio, military, and internet access [21]. The transition to a digital information infrastructure provides the opportunity to remove many limitations of analog communication systems caused by the need for tight coupling between the acquisition, transmission, and display components [11].

The block diagram of a typical digital communication system is illustrated in Figure 1.1. The purpose of channel encoding and decoding is to minimize the possibility of erroneous transmission. The error correction code used, as well as the encoding and decoding processes, define to a large extent the system efficiency. A digital communication system should be capable of transmitting the information from the source to the destination with no errors. The channel introduces noise to the transmitted information, thus resulting in reduced system reliability. In order to improve the reliability of the system and to protect it from the channel noise, channel encoder adds some redundant information, i.e., the so-called parity bits, to the transmitted data (information bits). The channel decoder undertakes to remove this redundant information and to convert the received sequence into binary, using a decoding algorithm. This process is called channel decoding.

FPGAs (Field Programmable Gate Arrays) are reconfigurable platforms that provide excellent performance/Watt ratio for the implementation computational intensive algorithms, e.g., from the field of Digital Signal Processing (DSP). During the last decade, the FPGA devices have progressed both in terms of resources and performance. The adoption of ultra-thin chip geometries, down to 14nm, and higher levels of integration, as well as the use of faster communication links and specialized cores, derive FPGAs that are easily customizable for DSP, data processing, and system connectivity applications [1]. Worthy competitors of FPGAs remain the ASIC (Application-Specific Integrated Circuit) and microprocessors.

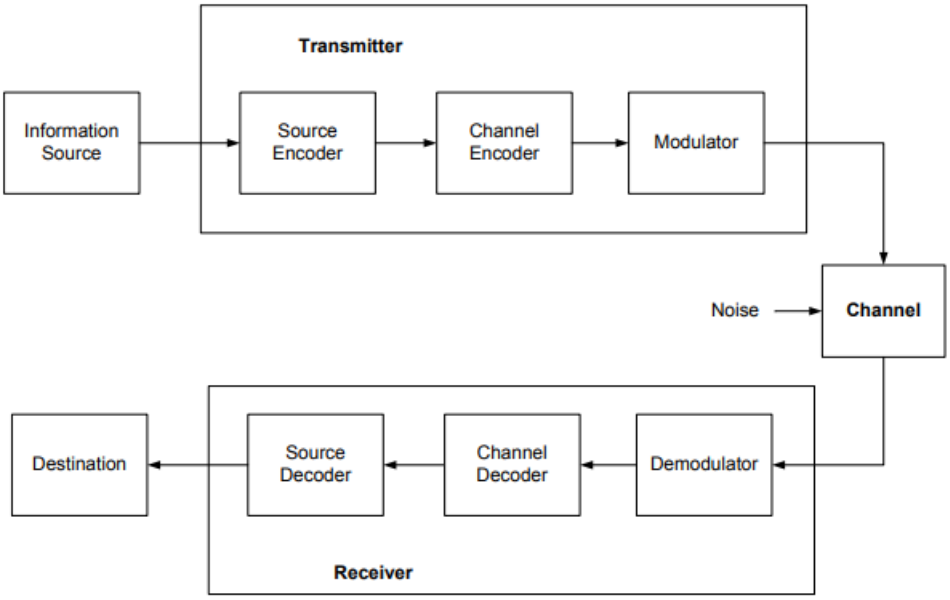


Figure 1.1: Block diagram of a Digital Communication System

Approximate Computing (AC) [14] is an alternative design approach that exploits the inherent error tolerance of algorithms and applications from domains such as machine learning (ML), DSP, numerical analysis, etc, and relaxes the accuracy in the calculations to provide significant gains in power and/or energy consumption, area, latency, etc. It can be applied at different layers of the design abstractions, i.e., starting from the application level and moving to the hardware and VLSI level [3].

1.1 Motivation and Thesis Objectives

This dissertation contributes to the areas of digital telecommunications and integrated systems design, focusing on the development of error correction systems. Specifically, extensive research has been conducted on decoding algorithms with approximation techniques.

More explicitly, the current thesis aims at:

- Studying and understanding various algorithms used in digital demodulation.
- Testing and verifying the algorithms, as well as examining their accuracy, through MATLAB simulations.
- Applying approximation techniques in the computational intensive tasks of the algorithms.
- Comparing the approximate versions of the algorithms with their accurate counterparts, by performing a theoretical study of their circuit complexity.

- Efficient implementation and parallelization of the algorithms on FPGA to exploit its full potential and achieve maximum throughput.
- Analyzing the experimental results and drawing conclusions about the novel implementations.

1.2 Thesis Outline

In chapter 2 some theoretical mandatory background will be given in order to better understand the broader meaning of digital communication. More specifically, some modulation/demodulation schemes will be analyzed and components of a telecommunications chain will be explained.

In chapter 3 approximation techniques will be introduced. After we verify the proper operation of our algorithms, these techniques will be added and we will observe the loss of accuracy each technique causes. The results will be carried out from Matlab simulations and will be compared to the theoretical ones (full-precision algorithms).

Following, in chapter 4 we will present the architecture of every algorithm and their approximate form implemented on an FPGA platform. The whole pipeline will be presented in detail in its parallel form.

Finally, in chapter 5 the experimental results will be displayed and some comparisons between approximate techniques will be made. The VHDL results will be compared to the Matlab ones and the trade-offs between utilization of resources and algorithm precision (BER achieved) will be presented.

Chapter 6 shows some conclusion drawn from the previous results, as well as some suggestions for future work.

Chapter 2

Theoretical Background

2.1 Digital Communication

Digital communication is the backbone for today's society, as the percentages of how many people use digital communication are extremely high, and that's why the digital world is growing bigger more powerful.

The design of a digital communication system starts with describing the channel which includes received power, available bandwidth, channel noise and other impairments such as fading. The data rate and the error performance are basic requirements of a digital communication system. The last introduces a level of error during transmission from the source to a receiver because of the noise in the physical channel. As a result of this introduction of error, many communication systems are coded in order to limit the number of errors that appear when decoding a noisy communication signal.

2.2 Digital Modulation Techniques

There are three basic ways in order to convert an analog waveform into a group of digital bits, by modifying the amplitude, the phase or frequency. Some modern techniques combine two or more variations to improve spectral efficiency.Most known techniques are briefly presented below.

2.2.1 Amplitude Shift-Keying (ASK)

Amplitude shift-keying is a form of amplitude modulation that represents digital data as variations in the amplitude of a carrier wave. In M-ary ASK each group of log_2M bits generates a symbol. The binary signal when ASK modulated, gives a zero value for low input while it gives the carrier output for high input.

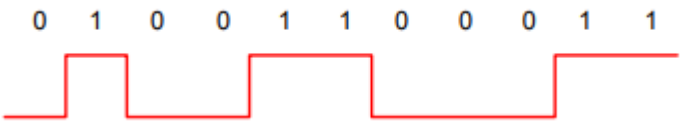


Figure 2.1: Original signal in a pulse sequence

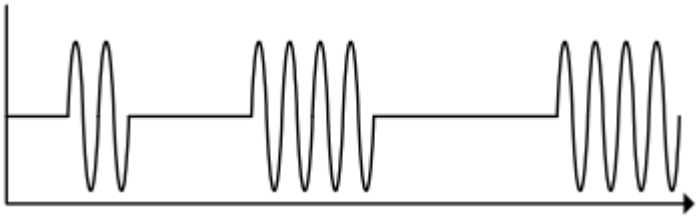


Figure 2.2: Amplitude shift-keying (ASK)

2.2.2 Frequency Shift-Keying (FSK)

Frequency-shift keying is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal. The output of a FSK modulated wave is high in frequency for a binary high input and is low in frequency for a binary low input.

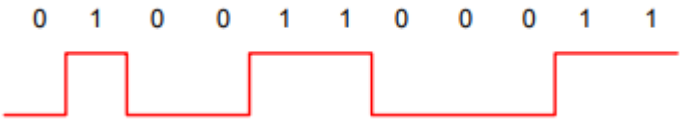


Figure 2.3: Original signal in a pulse sequence

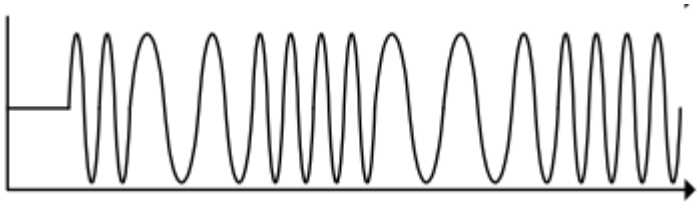


Figure 2.4: Frequency shift-keying (FSK)

2.2.3 Phase Shift-Keying (PSK)

Phase shift-keying is the digital modulation technique in which the phase of the carrier signal is changed by varying the sine and cosine inputs at a particular time. At the receiver, distinguishing between the two segments of sinuids is easier if their phases differ by as much as possible.

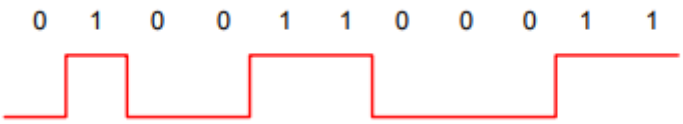


Figure 2.5: Original signal in a pulse sequence

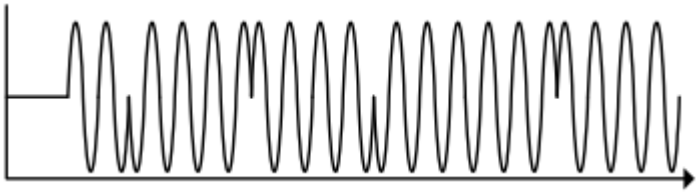


Figure 2.6: Phase shift-keying (PSK)

2.2.4 Quadrature Amplitude Modulation (QAM)

Quadrature Amplitude Modulation is a digital modulation technique in which the creation of symbols are some combination of amplitude and phase. In this way they can transmit more bits per symbol. For example, 16-QAM uses twelve carrier phases plus three amplitude levels to transmit 4 bits per symbol. Other popular variations are 64-QAM and 256-QAM, which they transmit 6 and 8 bits per symbol respectively.

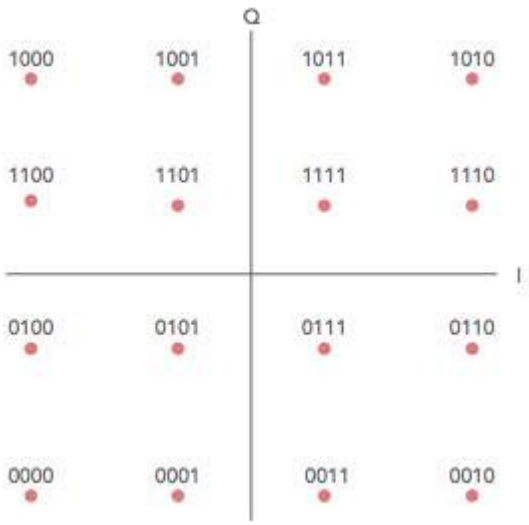


Figure 2.7: An example of 16-QAM constellation

As indicated, two controlling signals, known as the in-phase (I) and quadrature (Q) components, are required to implement Quadrature Amplitude Modulation. The number of QAM states is 2^N, as determined by the number N of binary bits per symbol.

A QAM modulator

A QAM signal can be generated by independently amplitude-modulating two carriers in quadrature ($\cos\omega t$ and $\sin\omega t$), as shown in Figure 2.8 [2].

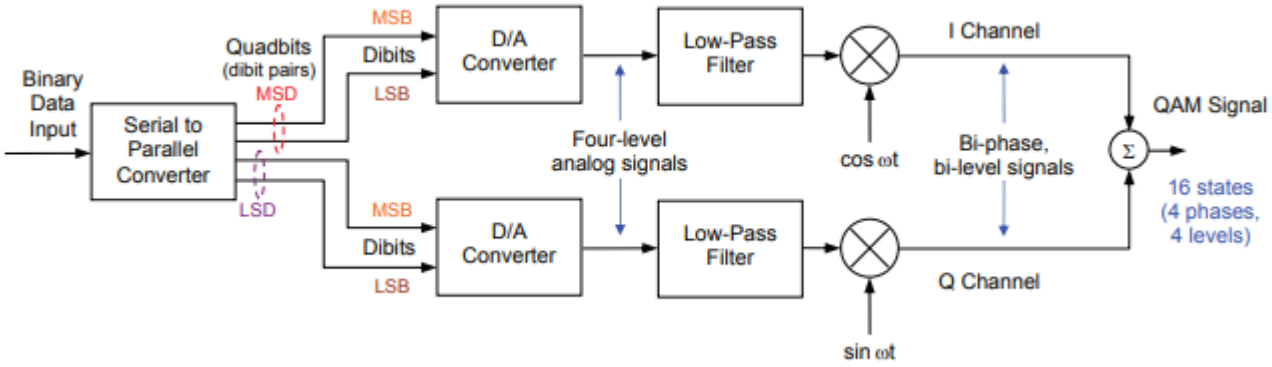


Figure 2.8: Simplified block diagram of a QAM modulator

Each time four bits are clocked serially into its buffer. The Serial to Parallel Converter outputs one quadbit in parallel at its four outputs.

2.3 Gray Code

Gray Code is a way ordering bit symbols such that two successive values differ in only one bit. It is used in order to minimize bit errors while demodulating a symbol, as the neighbor symbol of every value will be definitely off by one bit. For example, the representation of the decimal value "1" in binary would normally be "001" and "2" would be "010". In Gray Code, these values are represented as "001" and "011". That way, incrementing a value from 1 to 2 requires only one bit to change, instead of two. In Figure 2.9, an example of 16-QAM gray encoded vs natural numbering is shown.

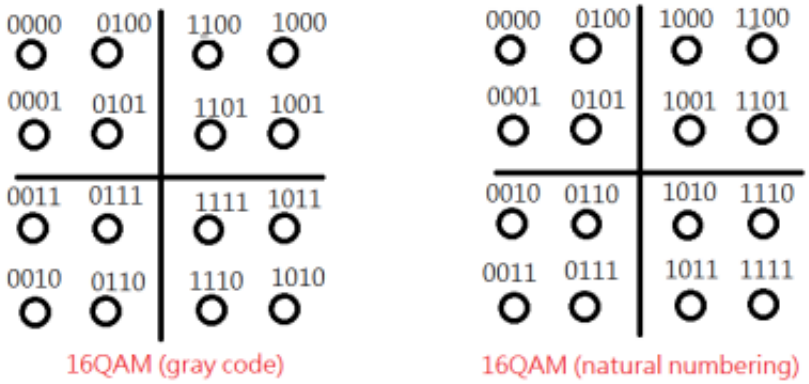


Figure 2.9: Example of 16-QAM gray encoding and natural numbering

2.4 Forward Error Correction (FEC)

Forward error correction works by adding redundant bits to a bitstream to help the decoder detect and correct some transmission errors without the need for retransmission. Like most error correction systems, this looks like we are reducing the data throughput of the system as we are creating redundant bits that do not form part of the specific stream. However, adding error correction to a system allows us to take advantage of a reduction in the signal to noise ratio. In telecommunications this results in higher line speeds as we can send more bits down a cable, in effect increasing the bandwidth and data throughput. In summary, FEC allows some tolerance for data loss and corruption without having to provide a reverse tally connection to signal data validity, thus maintaining high data throughput and integrity.

The transmission data rate of a signal is equivalent to:

$$Transmission\ \ Data\ \ Rate = Information\ \ rate \cdot (1/FEC\ \ rate)$$

FEC rate is typically in the range 1/2 to 7/8 so the transmission data rate is always significantly more than the information rate. The formula for the Symbol Rate is :

$$SymbolRate = DataRate/(m \cdot FEC) \tag{2.1}$$

where m is modulation factor (transmission rate bits per symbol) and FEC is forward error correction code rate (eg. 1/2, 2/3, 3/4, 5/6, 7/8).

Using smaller order FEC rates while keeping the same modulation has shown to have better performance than decreasing the modulation and increasing the FEC coding rate [15].

2.5 Additive White Gaussian Noise (AWGN)

All wireless receivers suffer from thermal noise. This noise is added to the received signal and makes detection of weak signals a difficult challenge. White refers to the idea that it has uniform power across the frequency band for the information system. It is an analogy to the color white which has uniform emissions at all frequencies in the visible spectrum. In simulations this noise is usually modeled as a Gaussian Random Process (thus called Gaussian).

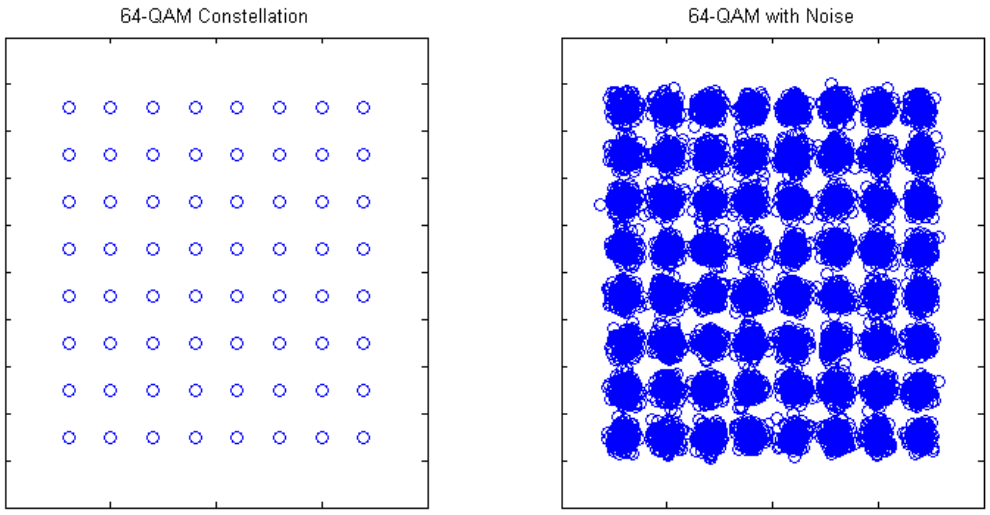


Figure 2.10: 64-QAM Constellation example with AWGN

2.6 Log Likelihood Ratio

The LLR method is a decoding technique that is used in soft decision algorithms. It concerns the probability that a bit of a received symbol be 0 or 1, given a set of parameters and possible outcomes. This prediction is based on symbol's mapping to a constellation based on each modulation.

For a better understanding an example of 4-QAM constellation map is shown in the figure below. Each of the blue dots are the four constellation points. When a symbol is transmitted there is an amount of noise in the channel that alters the initial signal. Therefore, the red dot is the relocated received symbol.

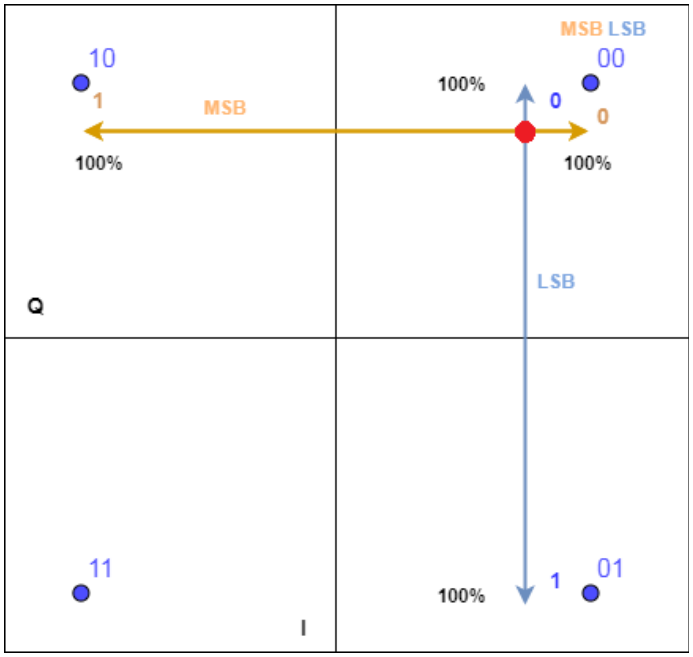


Figure 2.11: Example of 4-QAM Constellation Map

The Least Significant Bit (LSB) has a bit value 0 for the symbols above the Q-axis and a bit value 1 below the Q-axis. That means that the LSB of a transmitted symbol is expected to be 1 above the Q-axis and 0 below it. So, the received symbol has a higher probability to have its second bit with a value of 0.

The Most Significant Bit (MSB) has a bit value 0 for the symbols right of the I-axis and a bit value 1 left of the I-axis. That means that the MSB of a transmitted symbol is expected to be 0 right of the Q-axis and 1 left of it. So, the received symbol has a higher probability to have its first bit with a value of 0. Therefore, there is a high probability that the initial transmitted symbol was "00".

2.7 Soft Decision Algorithms

This section is concerned with the performance of binary codes under maximum likelihood soft decision decoding. In general, soft decision has better performance than hard decision decoding and the fact that it is able to estimate the performance of codes makes it attractive.

2.7.1 Exact LLR

The exact LLR is an algorithm which computes the most accurate values of LLR. However, this high accuracy results in combination of complex hardware and large power consumption due to the complicated mathematical operations. In this algorithm, the LLR for a transmitted bit b is defined as:

$$L(b) = \ln \frac{Pr(b = 0|r = (x, y))}{Pr(b = 1|r = (x, y))} \tag{2.1}$$

where r is the received signal with coordinates (x, y) and $Pr(b = j|r = (x, y))$ is the probability that the bit value of the transmitted bit b is j ($j = 0$ or 1) conditioned that x and y are received.

Assuming equal probability for all symbols, the LLR for an AWGN channel can be expressed as:

$$LLR(b) = \ln \left(\frac{\sum_{s \in S_0} \exp -\frac{1}{\sigma^2}((x - s_x)^2 + (y - s_y)^2))}{\sum_{s \in S_1} \exp -\frac{1}{\sigma^2}((x - s_x)^2 + (y - s_y)^2))} \right) \tag{2.2}$$

where σ^2 is the noise variance of baseband signal, S_0 and S_1 are constellation points with bit 0 and 1 respectively and s_x and s_y are In-Phase and Quadrature coordinate respectively.

2.7.2 Approximate LLR

The Approximate LLR is an algorithm that calculates LLR by using only the two closest constellation points with bit value j at the given bit position. The equation (2.2) is complicated due to the fact that there are terms in both numerator and denominator. Sub-optimal solution with simplified LLR can be obtained by log-sum-exponential approximation [20] : $log \sum_i exp(\phi_i) = max_i(\phi_i)$.

$$LLR(b) = -\frac{1}{\sigma^2}(\min_{s \in S_0}((x - s_x)^2 + (y - s_y)^2) - \min_{s \in S_1}((x - s_x)^2 + (y - s_y)^2)) \tag{2.1}$$

The calculation of the LLR is determined by the LLR of each bit in the symbol. For a specific bit there is a list of points on the constellation where that particular bit has the value of one or zero. In order to calculate the LLR for each bit, the points where that bit has the value of one or zero are separated into two lists – a list of points where the bit value is zero and a list where the bit value is one. The probability of the bit value of the received point is determined by the difference between the minimum of the one list when subtracted from the minimum of the zero list. The result is a value indicating whether that bit is more likely to be a one or zero. The larger from the list of ones or zeros will dominate the other and indicate by the magnitude of the result the relative probability.

2.7.3 Piecewise LLR

The piecewise LLR is a further approximation of Approximate LLR as every function of a single bit can be represented as a linear function [19]. After this simplification the LLRs are expressed as below:

$$D_{I,k} = \begin{cases} y_I[i] & k = 1 \\ -|D_{I,k-1}| + d_{I,k} & k > 1 \end{cases} \tag{2.1}$$

$$D_{Q,k} = \begin{cases} y_Q[i] & k = 1 \\ -|D_{Q,k-1}| + d_{Q,k} & k > 1 \end{cases} \tag{2.2}$$

where $d_{I,k}$ and $d_{Q,k}$ denote half the distance between the partition boundaries relative to bit $b_{I,k}$ and $b_{Q,k}$, with $k > 1$ and $y[i]$ is the received equalized signal.
For example, the approximate expressions for a 64-QAM are given by:

$$\begin{aligned} D_{I,1} &\simeq y_I[i] \\ D_{I,2} &\simeq -|y_I[i]| + 4 \\ D_{I,3} &\simeq -|y_I[i] - 4| + 2 \\ D_{Q,1} &\simeq y_Q[i] \\ D_{Q,2} &\simeq -|y_Q[i]| + 4 \\ D_{Q,3} &\simeq -|y_Q[i] - 4| + 2 \end{aligned}$$

2.8 Hard Decision Algorithms

Hard Decision decoders receive a stream or a block of bits and decide whether each received bit is one or zero by setting the threshold as shown in Figure 2.12. It compares samples' voltages to threshold values and if a voltage is greater than that value it is decoded as one, otherwise as zero. The decoding is done irrespective of how close the voltage is to the threshold.

2.8.1 Hamming Distance

A bounded distance decoder that uses Hamming Distance compares the received codeword with all the possible codewords. Hamming Distance is the number of bits that these codewords differ. A block of bits with the minimum hamming distance is picked.

All possible Codewords	Hard Decision Outputs	Hamming Distance
000	111	3
011	111	1
101	111	1
110	111	1

Assume the message bits are ”10” and applied to parity encoder and we get ”101” as the output codeword. The output codeword “101” is then transmitted through the channel. The channel attenuates the signal that is being transmitted and the receiver sees a distorted waveform (red color waveform). At each sampling instant in the receiver the hard decision decoder determines the state of the bit to be “0” if the voltage level falls below the threshold and “1” if the voltage level is above the threshold. Therefore, the output of the hard decision block is “111”. Perhaps this “111” output is not a valid codeword, which implies that the message bits cannot be recovered properly. The decoder compares the output with all possible codewords and computes the minimum Hamming Distance for each case. In our case, as shown below, the min Hamming Distance is 1 and there are 3 codewords with this distance. So, the decoder picks randomly one of them. The probability of picking the correct codeword is 1/3.

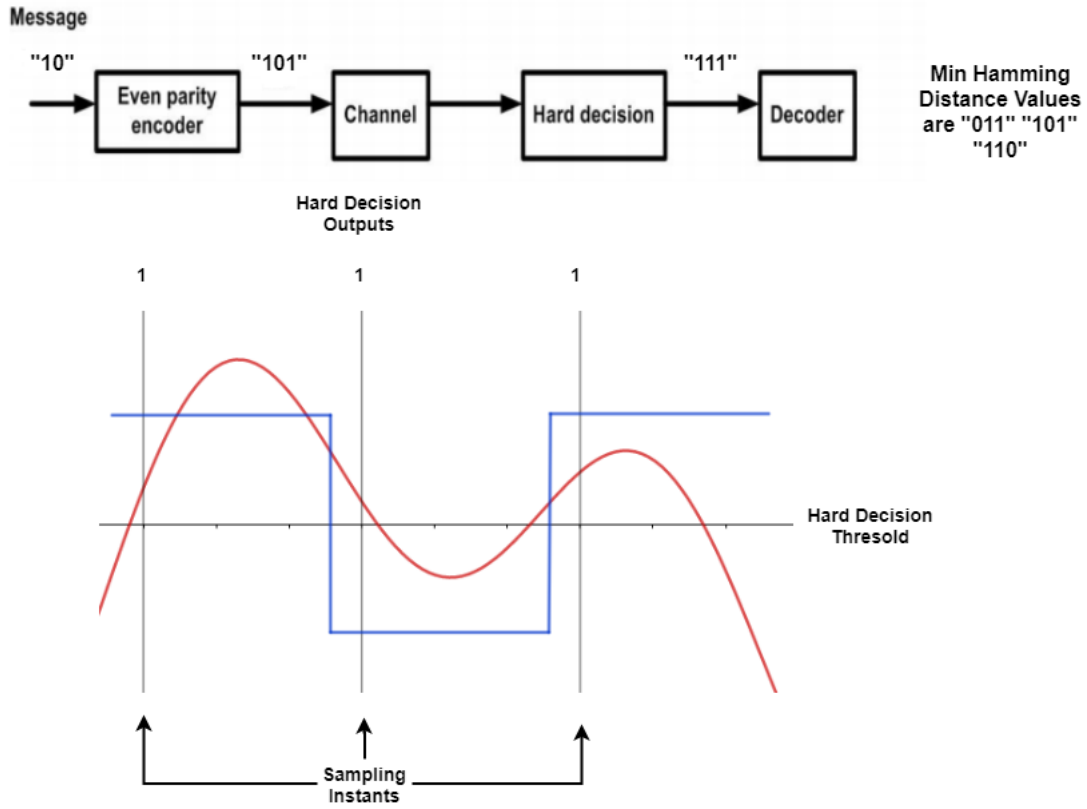


Figure 2.12: Example of Hard Decision Decoding

2.8.2 Maximum Likelihood Detection

The novel Maximum likelihood (ML) is a non-linear symbol detection method that has been used for optimal symbol detection in various engineering fields. This algorithm achieve exactly the same performance as the conventional ML detection with a reduced implementation of ML detection.

Based on [22] the estimated square M-ary QAM symbol x can be written in a closed form as a function of the received signal y as:

$$\mathrm{x} = \sum_{n=1}^{log_2\sqrt{M}} c_n \tag{2.1}$$

where

$$\mathrm{c}_n = \sqrt{\frac{3M}{M-1}} 2^{-n} e^{jg(y-\sum_{m=1}^{n-1} c_m)} \tag{2.2}$$

Thus, the novel Maximum likelihood is adopted to have the estimated square QAM symbol given in a closed form as a function of the received signal and the estimated channel.

Chapter 3

Testing and Verification

3.1 Algorithm Comparison

In most digital systems, Exact LLR is considered the best decoding algorithm. However, it has some drawbacks. Composite computational operations, such as exponential and logarithmic, make its hardware complexity and power consumption high. In order to reduce these disadvantages, Approximate LLR was created. This algorithm, although its large design circuit, is much simpler than Exact LLR, since it does not have the complex operations of the first one. Its BER performance is quite close to Exact LLR, something that makes it a worthy soft decision decoder. The third algorithm, Piecewise LLR, tries to further reduce the complexity of the Approximate LLR, as it introduces piecewise linear functions that approximate the effect of nonlinearity in Approx LLR function. It is undoubtedly the least complicated algorithm. The one disadvantage of this method is that the definition of the piecewise linear functions is not done systematically but heuristically. In the following, the differences mentioned between the algorithms will be developed and presented, as well as some other metrics in which some methods lag behind and some exceed.

3.1.1 Circuit Complexity

The soft decision algorithms are compared considering the number of multipliers required for their implementation on an FPGA.

Exact LLR Algorithm

The Exact LLR algorithm is described by the equation below. For a M-QAM, equation involves three multipliers done for M/2 constellation points and three multipliers done for the rest M/2 constellation points.Therefore, there are a total of $3 \cdot M$ multipliers for one symbol.

$$LLR(b) = log(\frac{\sum_{s \in S_0} \exp -\frac{1}{\sigma^2}((x-s_x)^2+(y-s_y)^2))}{\sum_{s \in S_1} \exp -\frac{1}{\sigma^2}((x-s_x)^2+(y-s_y)^2))}$$

- Total Number of Multipliers $= 3 \cdot \frac{M}{2} + 3 \cdot \frac{M}{2} = 3 \cdot M$

Approximate LLR Algorithm

The Approximate LLR algorithm is described by the equation below. The equation involves two multiplications, as each real and imaginary component requires an individual multiplication calculation. Also, there is an additional multiplication included in the calculation for each bit in the symbol.

$$LLR(b) = -\frac{1}{\sigma^2}(\min_{s \in S_0}((x-s_x)^2 + (y-s_y)^2) - \min_{s \in S_1}((x-s_x)^2 + (y-s_y)^2))$$

- Total Number of Multipliers $= 2 \cdot \frac{M}{2} + 2 \cdot \frac{M}{2} + log_2 M = 2 \cdot M + log_2 M$

Piecewise LLR Algorithm

The piecewise LLR is is described by $log_2(M)$ linear functions. These expressions involves zero multiplications, as shown below in generalized formulae, where $d_{I,k}$ and $d_{Q,k}$ denote half the distance between the partition boundaries relative to bit $b_{I,k}$ and $b_{Q,k}$, with $k > 1$ and $y[i]$ is the received equalized signal.

$$\mathrm{D}_{I,k} = \begin{cases} y_I[i] & k = 1 \\ -|D_{I,k-1}| + d_{I,k} & k > 1 \end{cases}$$

$$\mathrm{D}_{Q,k} = \begin{cases} y_Q[i] & k = 1 \\ -|D_{Q,k-1}| + d_{Q,k} & k > 1 \end{cases}$$

Then. the LLR function can be computed as follows, where $H(i)$ is the channel frequency response (CFR) to the ith subcarrier

$$LLR(b_{I,k}) = |H(i)|^2 \cdot D_{I,k}$$

$$LLR(b_{Q,k}) = |H(i)|^2 \cdot D_{Q,k}$$

- Total Number of Multipliers = 0

Maximum Likelihood Detection

The Maximum Likelihood Detection with the closed form solution described before is an algorithm that receives a code word and tries to decode it in the ideal code word. This estimated M-ary QAM symbol x can be written as:

$$\mathrm{x} = \sum_{n=1}^{log_2\sqrt{M}} c_n \tag{3.1}$$

where

$$\mathrm{c}_n = \sqrt{\frac{3M}{M-1}} 2^{-n} e^{jg(y-\sum_{m=1}^{n-1} c_m)} \tag{3.2}$$

Multiplying $e^{jg(y-\sum_{m=1}^{n-1} c_m)}$ by 2^{-n} for $n = 1, 2, \ldots, log_2\sqrt{M}$ in (3.6) requires $2log_2\sqrt{M}$ total multipliers.

- Total Number of Multipliers = $2log_2\sqrt{M}$

The results of the complexity analysis showed the Piecewise Algorithm and Max Likelihood Detection use significantly less multiplications than the other two decoders. However, it is equally important to analyze their BER in order to investigate all performance metrics.

Total Number of Multipliers

	Exact	**Approx**	**PieceWise**	**ML**
64QAM	192	134	0	6
256QAM	768	520	0	8

3.1.2 BER Performance

One of the main goals of the our design exploration was to find the algorithm with the best overall performance. For this purpose, the BER (Bit Error Rate) metric is employed to evaluate three soft decision and one hard decision algorithms for 64-QAM and 256-QAM.

We assume a transmitter producing random binary digits with a convolutional encoder having a code rate of 1/2. The system employs a QAM modulation considering a normalized constallation diagram to keep the average symbol energy to unit. The modulated signal passes through an Additive White Gaussian Noise channel. Our QAM demodulator computes log-likelihood ratios (LLRs) which are processed by a Viterbi Decoder that is set up in unquantized mode. After the bit error calculation, the BER performance of our receiver is computed and displayed.

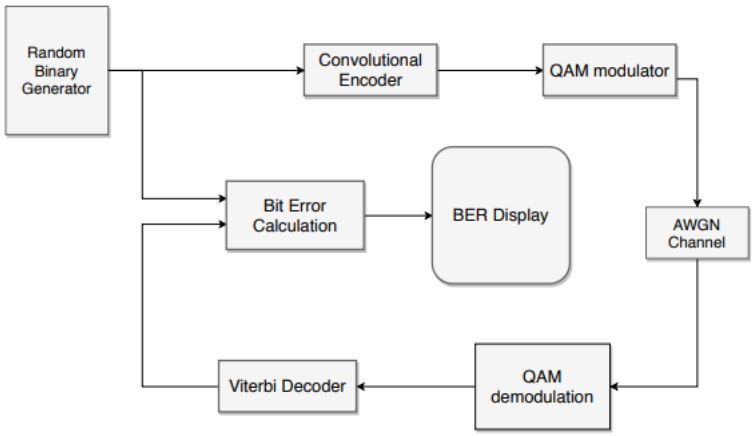


Figure 3.1: Block diagram for QAM Demodulation with Viterbi Decoding

The received symbol corresponding to the kth sample of the transmitted symbol, can be expressed as:

$$\mathrm{Y}(k) = X(k)H(k) + W(k) \tag{3.1}$$

H(k) is the channel frequency response at the kth subcarrier, Y(k) is the received symbol, X(k) is the transmitted symbol and W(k) is the complex additive white Gaussian noise (AWGN) with variance σ_0^2. After performing zero-forcing (ZF) frequency equalisation, one can obtain the following expression:

$$\mathrm{Z}(k) = Y(k)/H(k) = X(k) + W(k)/H(k) = X(k) + V(k) \tag{3.2}$$

where V(k) is the complex AWGN with variance $\sigma^2 = \sigma_0^2/|H(k)|^2$.

In BER simulations shown below, the channel coefficient h is modeled as a zero mean circularly symmetric complex Gaussian random variable with unit variance, as well as the noise w with variance. $1/\sqrt{10^{(E_b/N_0)/10} \cdot log_2 M}$. The employed convolutional encoder has the generator polynomial (133,171) and constraint length of 7.

3.2 LLR Evaluation with Approximation Techniques

Approximate computing techniques are employed in the design of efficient digital systems and circuits for applications that demonstrate inherent error resilience. At circuit-level, extensive research has been conducted in the design of inexact adders [7, 6, 16] and multipliers [9, 5, 4, 10, 8], i.e., the core components of DSP accelerators. Towards this direction, and in order to simplify the complexity of our circuits we applied approximation techniques, i.e., bit truncation in our data, approximate multipliers on fixed-point arithmetic, as well as approximate multipliers on floating-point arithmetic. Each technique was tested based on MATLAB simulations for each one of the 3 soft decision algorithms for specific SNR values and 64-,256-QAM.

Next, we evaluate the approximate versions of the examined algorithms by comparing their LLR outputs with the respective ones of the full-precise algorithms. The proposed error evaluation metric of the bar diagrams below is LLR Mean Relative Error (LMRE) :

$$LMRE = \frac{\sum_{i=1}^{N} \frac{|LLR(i)_{accur} - LLR(i)_{approx}|}{|LLR(i)_{accur}|}}{N} \cdot 100\% \tag{3.1}$$

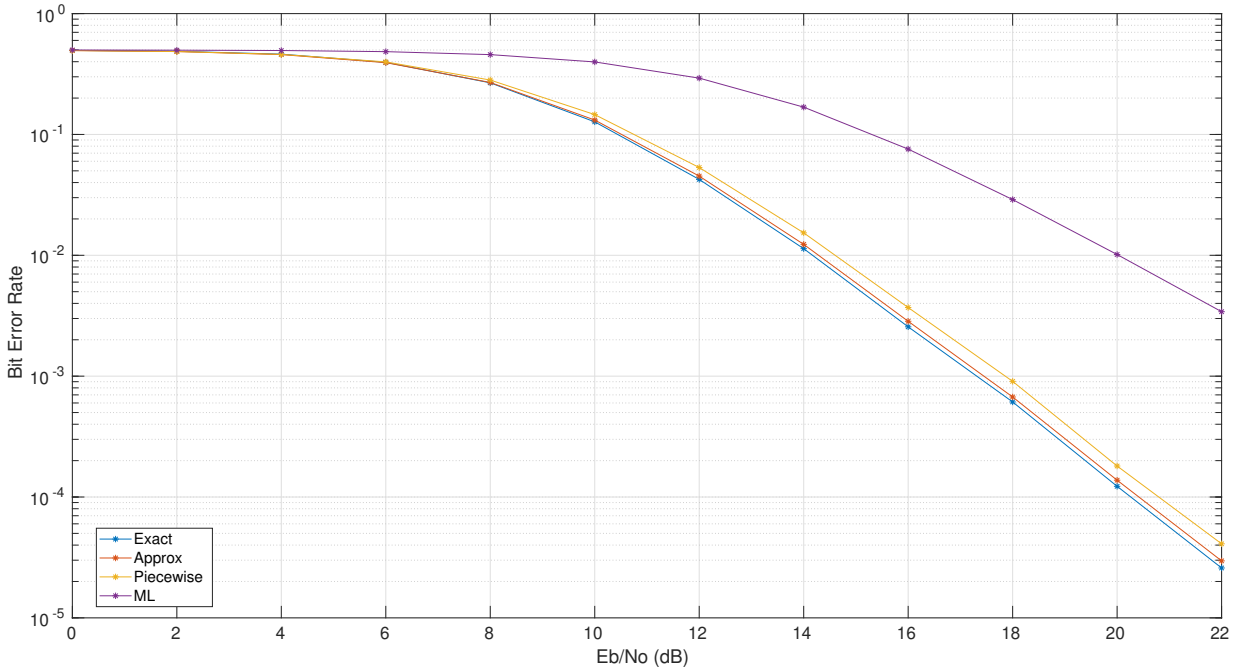


Figure 3.2: BER performance comparison of algorithms in 256-QAM

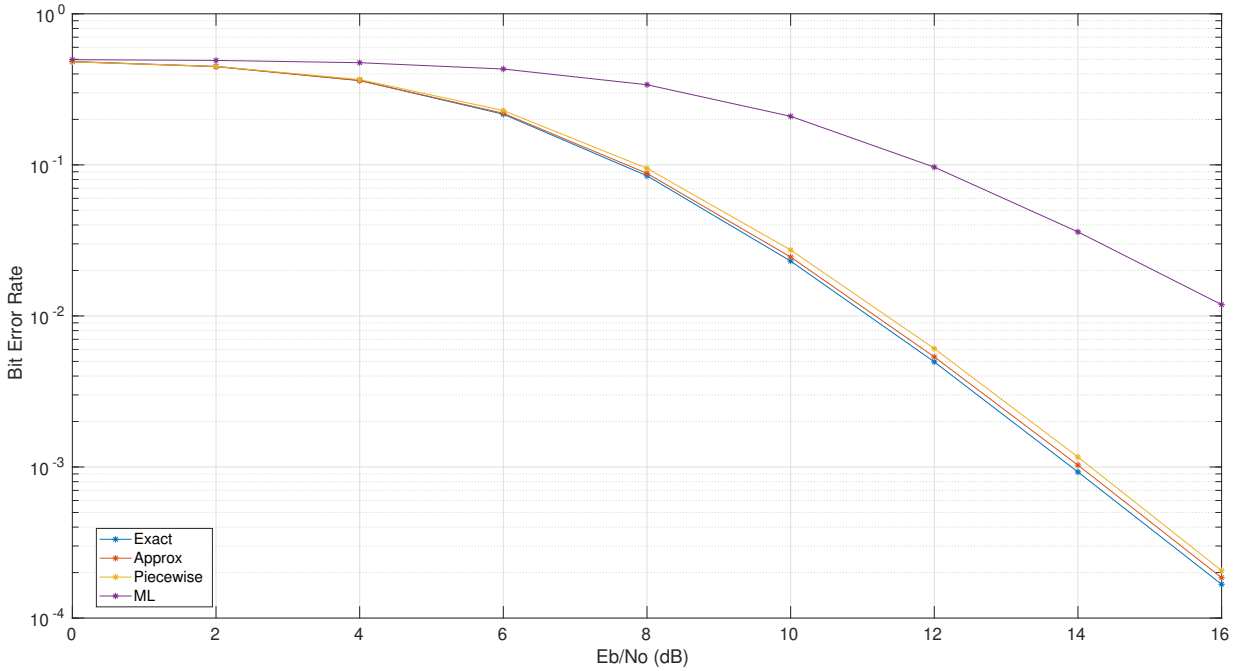


Figure 3.3: BER performance comparison of algorithms in 64-QAM

The presented analysis regards fixed point arithmetic. So we consider our I and Q signals are fixed point signed numbers with a word length of 16 bits and fraction length 14 bits ([16 14]). Our inputs as mentioned are normalized to unit length and that is why 2 bits in integer part are sufficient. The metric for our bar diagrams below is LLR Relative Error and it is expressed as a percent. Its formula is:

3.2.1 Bit Truncation on Fixed-Point Arithmetic

The first approximation technique is the conventional Bit Truncation, which variably truncates the least significant bits (LSB) of the inputs to reduce the complexity of our computations. Our main goal is to achieve an acceptable trade-off in accuracy depending on the decoding algorithm and its accuracy limits. In the figures below, the LLR Relative Error (3.1) is presented for each soft decision algorithm.

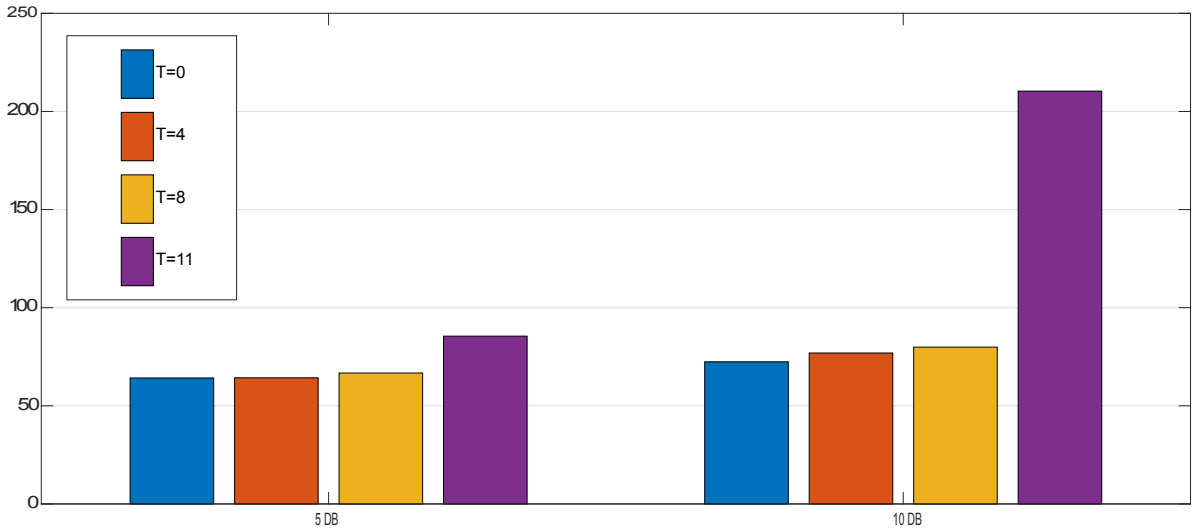


Figure 3.4: LLR Relative Error for 64-QAM Exact Algorithm and two values of EbNo

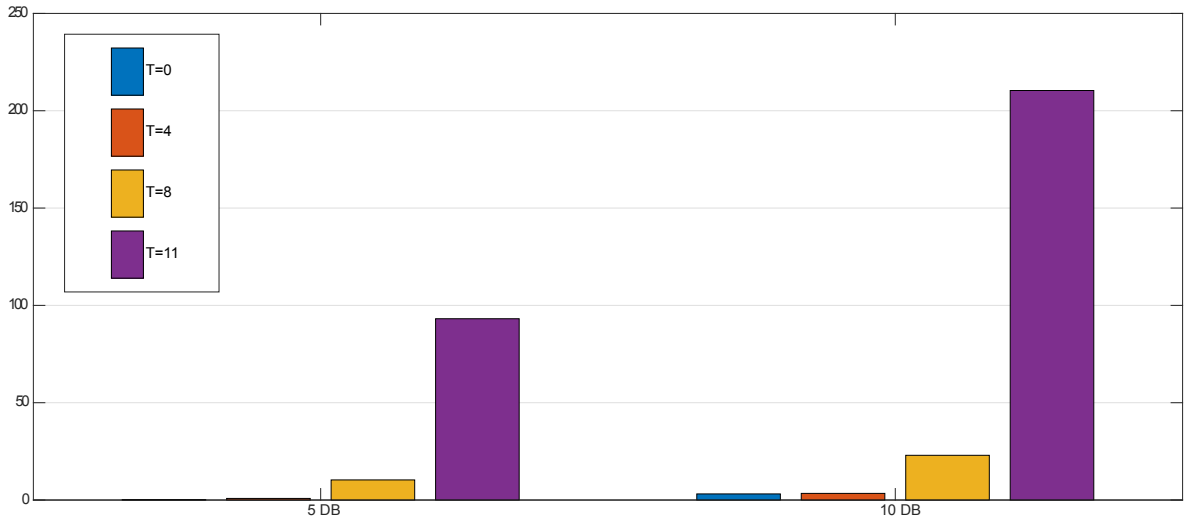


Figure 3.5: LLR Relative Error for 64-QAM Approx Algorithm and two values of EbNo

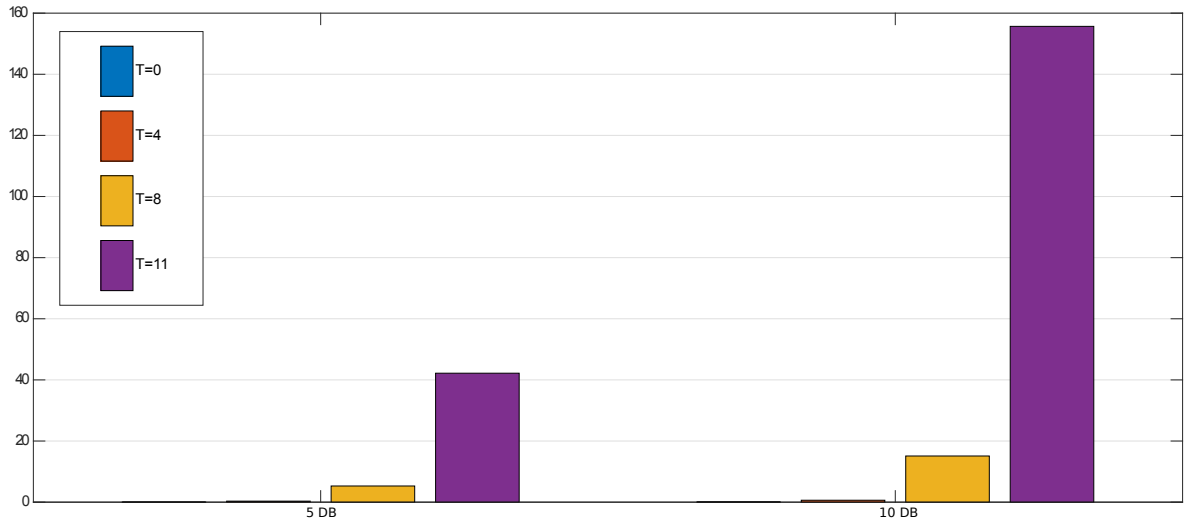


Figure 3.6: LLR Relative Error for 64-QAM Piecewise Algorithm and two values of EbNo

The results show that the Exact Algorithm in contrast with the other two has a significant relative error in its LLR values. This happens because of its complex computational operations like exponential and logarithmic. These have been implemented with a Lookup Table (LUT) and thus, their outputs have a strictly specific range.

3.2.2 Approximate Radix Multiplication on Fixed-Point Arithmetic

This technique is an approximate hybrid high radix encoding for designing energy-error efficient inexact multipliers. High radix encodings offer partial products reduction, and as a result, their accumulation requires smaller trees, leading to energy, area, and/or delay savings [9].

In this technique, the most significant bits (MSBs) of the multiplicand B are encoded using the radix-4 encoding, whereas the k least significant bits (LSBs) are encoded using a radix-2^k (with $k \geq 4$). After this generation of B' the approximate multiplication $A \cdot B'$ is performed.

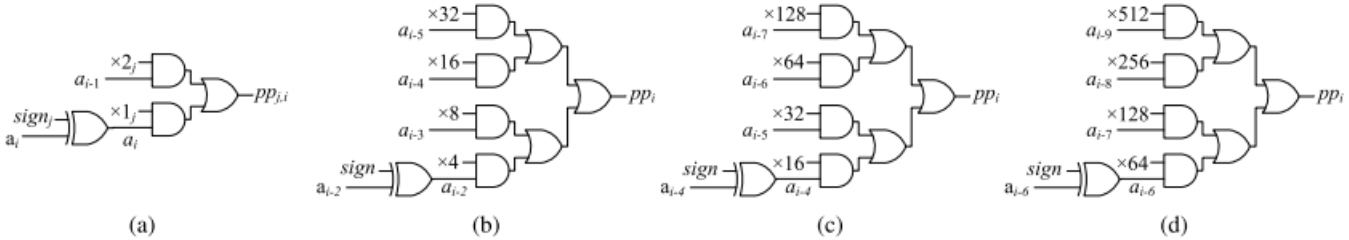


Figure 3.7: i-bit partial product generator based on (a) accurate radix-4 encoding and the approximate (b) radix-64, (c) radix-256, and (d) radix-1024 encoding. a_i : i-bit of operand A, $\alpha_i = a_i \oplus sign$.

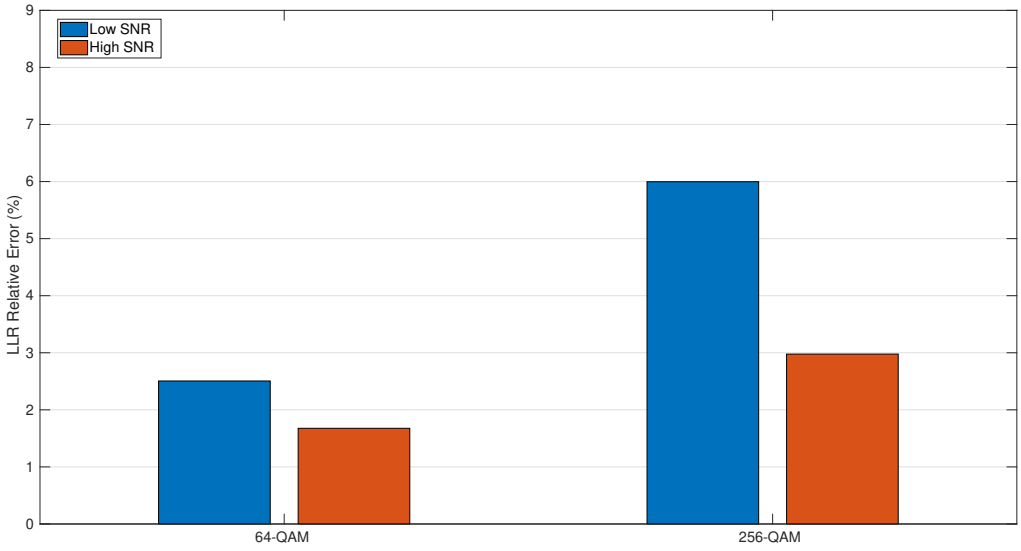


Figure 3.8: LLR Relative Error for 64-QAM via Exact Algorithm

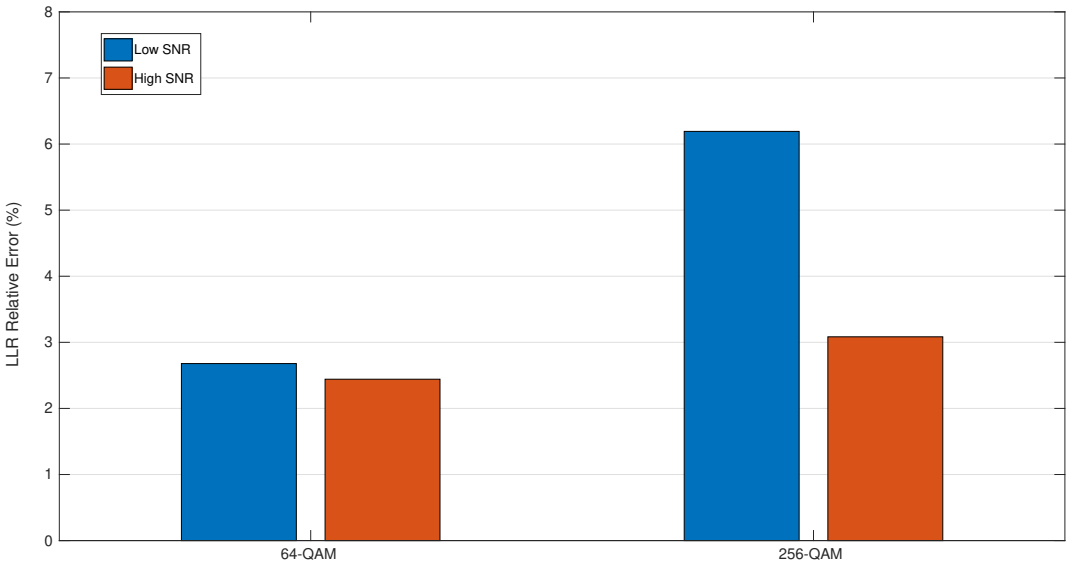


Figure 3.9: LLR Relative Error for 64-QAM via Approx Algorithm

The remarkable conclusions in Figures 3.8 and 3.9 are two. It is shown that the radix method has almost the same behaviour both in Exact and Approx as concerns the LLR accuracy. And last but not least, as the SNR increases, the loss in accuracy gets smaller. This means that in the range of high SNR this method becomes more efficient in a communication system. The 64-QAM was tested under 5db and 10db, while the 256-QAM under 10db and 20db respectively.

3.2.3 Approximate RMAC Multiplication on Floating-Point Arithmetic

RMAC is a Runtime Configurable Floating Point Multiplier for Approximate Computing. This approximate method multiplies two floating numbers and yields a high precision product. RMAC approximates the costly mantissa multiplication to a simple addition between the mantissa of input operands [5]. Despite the fact that this approximate multiplier it is worth mentioning for its energy efficiency and low execution time, our main metric here is again the mean error of LLR. So, some comparisons between this technique and the full precision algorithms are shown below.

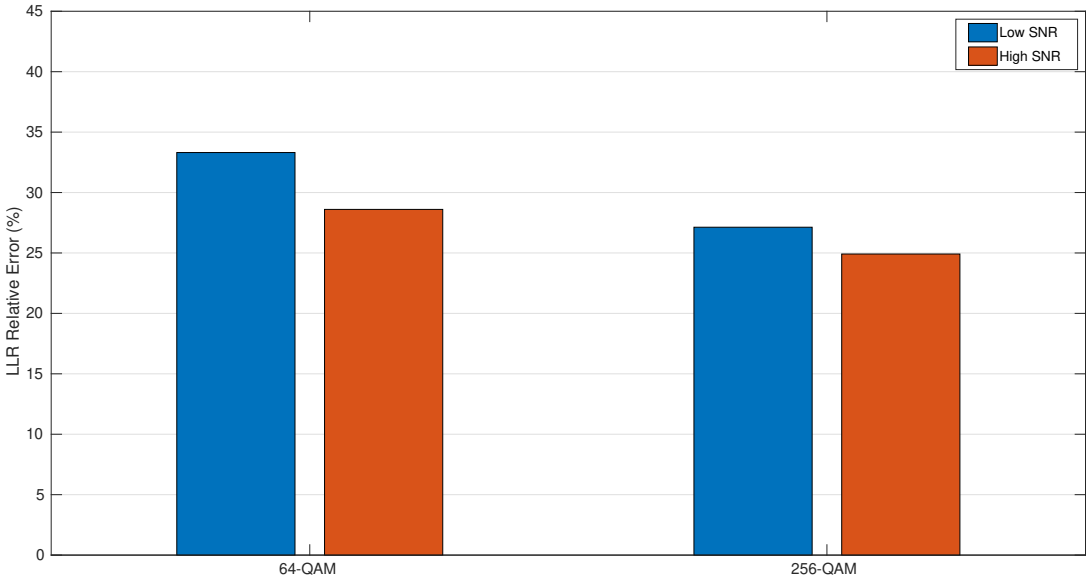


Figure 3.10: LLR Relative Error for Exact LLR using RMAC approximation

In Figures 3.10, 3.11 we can see a higher LLR relative error from the Radix Multiplier regardless from noise variance and SNR. In addition the same conclusion applies here as well, as concerns the SNR increasement. As it goes higher the approximate multiplier has less relative error from the full precision values.

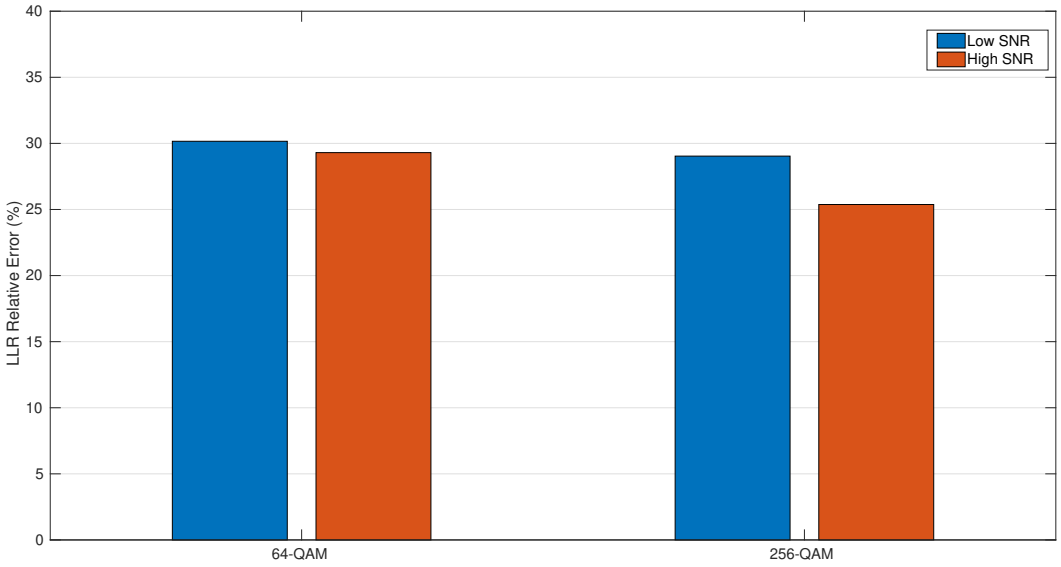


Figure 3.11: LLR Relative Error for Approx LLR using RMAC approximation

3.2.4 Approximate CFPU Multiplication on Floating-Point Arithmetic

CFPU is a Configurable Floating Point Multiplier for Energy-Efficient Computing. This technique works by replacing the most costly step of the operation with a lower energy alternative [4]. By this way, it significantly reduces energy and improves performance of multiplication at the expense of accuracy. The diagrams below analyze this loss of

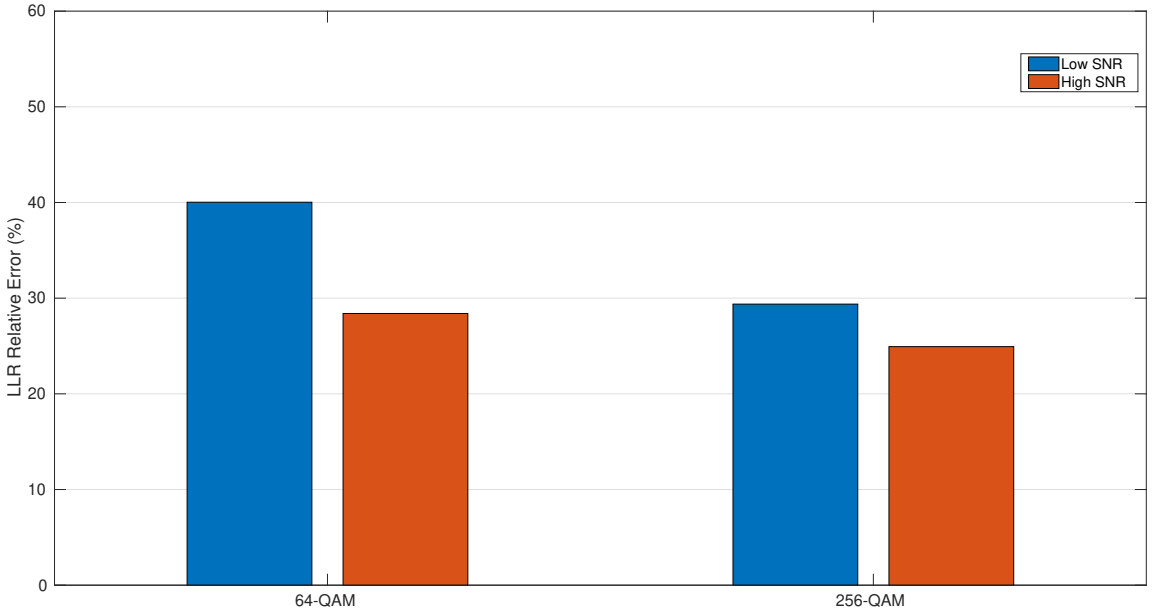


Figure 3.12: LLR Relative Error for Exact LLR using CFPU approximation

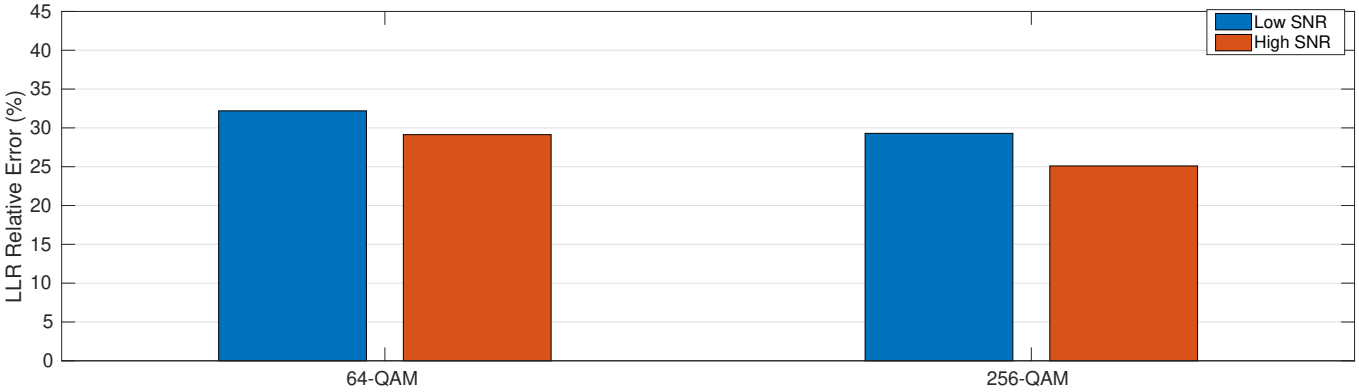


Figure 3.13: LLR Relative Error for Approx LLR using CFPU approximation

From the figures above we can observe similar accuracy as RMAC approximation. However, based on [4], RMAC can achieve significantly higher hit rate and efficiency as compared to CFPU while providing the same level of computation quality.

Chapter 4

FPGA Circuit Design

4.1 Introduction

Until now, we have theoretically studied and analyzed how these decoding algorithms behave from MATLAB codes. In this chapter, we are going to focus on their characteristics and their implementation in VHDL language. The algorithms chosen to be designed and implemented at this point are the Soft Decision (Exact LLR, Approx LLR and Piecewise LLR), as in general they have better BER performance. The arithmetic of these circuits is fixed-point and thus, the approximate techniques that are applied to the algorithms are Bit Truncation and Radix Multiplication. The tool used for this purpose was Vivado Design Suite 2019.2 of Xilinx.

4.2 Block Design

As soft decision decoders have generally better performance than hard decision decoders, we have chosen to implement the Exact LLR, Approximate LLR and Piecewise LLR. These three algorithms base upon the LLR values and offer soft bits that are applied afterwards to the Viterbi decoder. A design abstraction of the way they perform is given below.

4.2.1 Block Diagram of Exact LLR

In the schematic diagram 4.1 for the M-QAM demodulation there are M constellation points of the gray coded map which introduced to the Euclidean Distance component. The last computes the difference d_i of the received point and the expected constellation point. The next component produces the exponent of this result and then the M-distances are redistributed depending of the M-QAM mapping. In continuously, $2 \cdot N(N = log_2 M)$ sums, as described in (2.2), are produced. After their logarithmic calculation N LLR values are parallel arise.

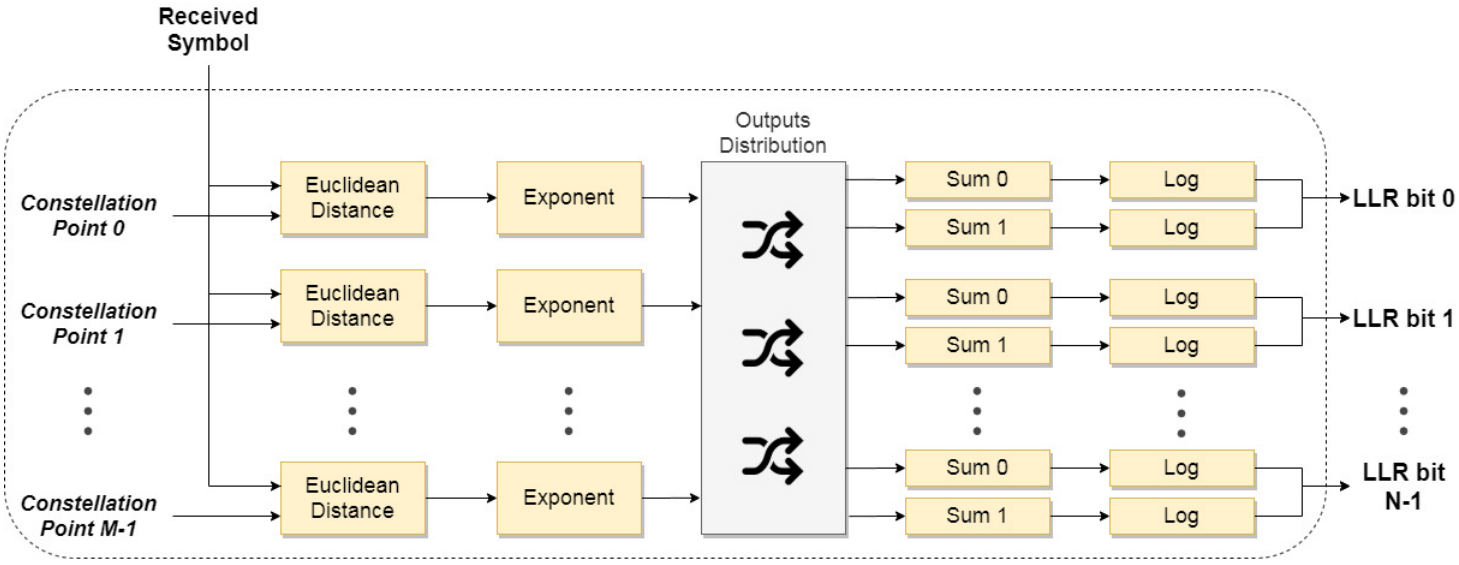


Figure 4.1: Block diagram of M-QAM demodulation via Exact LLR

4.2.2 Block Diagram of Approximate LLR

Block Diagram of Approximate LLR looks quite similar to this of Exact. The main difference is the simplification at the complex operations like exponent and logarithmic. Instead of these, there is a component that finds the minimum value of some calculated distances. The rest functions remain as described at Exact's schematic.

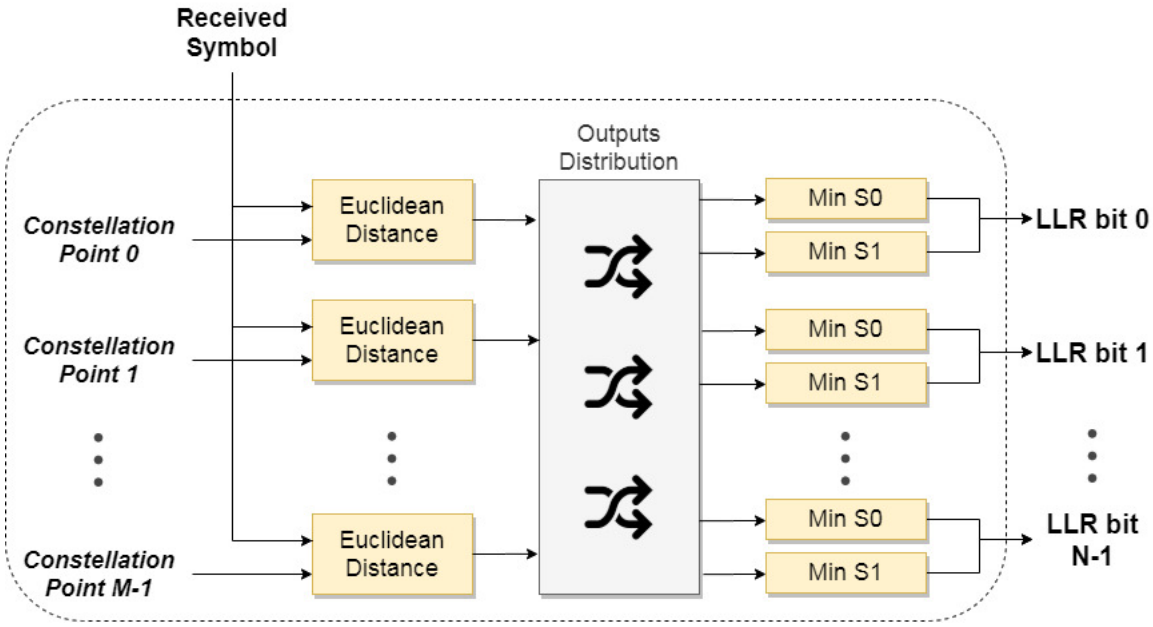


Figure 4.2: Block diagram of M-QAM demodulation via Approx LLR

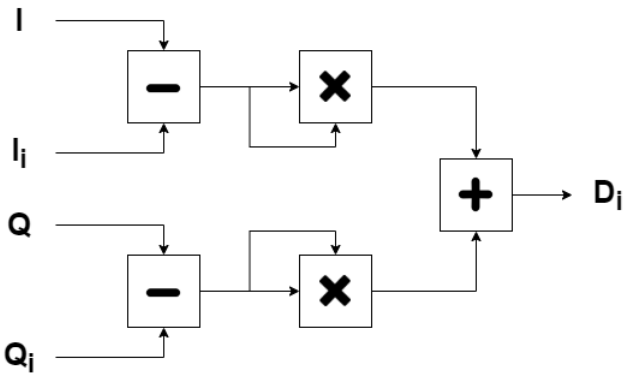


Figure 4.3: Schematic diagram for Euclidean Distance component

In Figure 4.3 the I/Q data are received and subtracted from the expected ith I/Q point. The result is squared and the results are added together. The outcome is the distance from the ith constellation point.

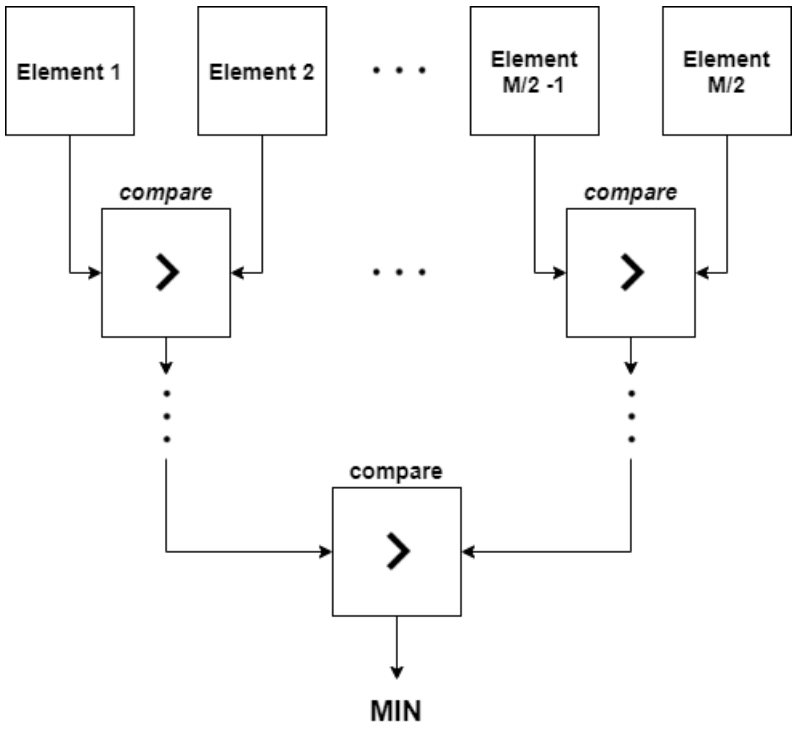


Figure 4.4: Schematic diagram for finding the minimum value of an array

For a M-QAM demodulation, Approx LLR tries to find the minimum value of the $M/2$ computed distances corresponding to bit 0 and the other $M/2$ corresponding to bit 1. Every Min-component consists of a fully pipelined tree. A comparison for every two inputs of the $M/2$-array is done and after $log_2 \frac{M}{2}$ steps the minimum value is arised and stored in a register.

4.2.3 Block Diagram of Piecewise LLR

The third implemented algorithm consists of N approximate functions. Half of them get as input the imaginary part of the received symbol and the rest of them the real part of it. These functions computes N values of LLR as described in equations (2.1-2.2) and produce them in parallel.

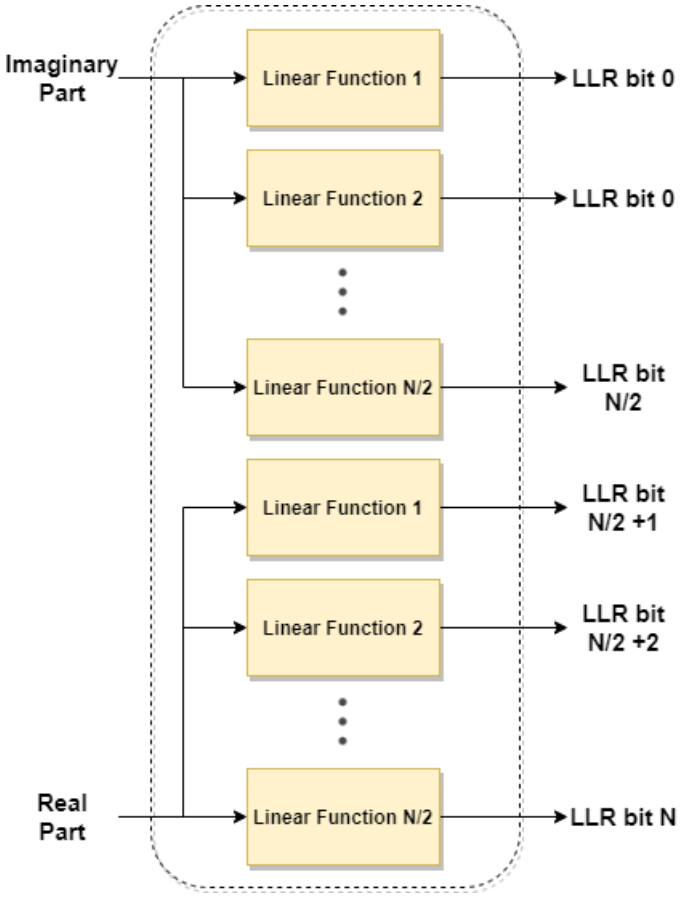


Figure 4.5: Block diagram of 2^N-QAM demodulation via Piecewise LLR

4.3 VHDL Components

This section concerns only the Exact LLR. This algorithm in order to calculate the exact value of LLR as expressed in (2.2), we had to implement exponential and natural logarithmic function. There are several solutions for doing that in VHDL. The first algorithm we chose for this implementation was hyperbolic CORDIC and in fully pipelined version. The second way is Remez algorithm which converts a function to a polyonomial of best approximation [18]. A briefly analysis of their architecture is given below.

4.3.1 Exponential

CORDIC

The hyperbolic CORDIC algorithm as originally proposed by Walther allows the computation of hyperbolic functions in an efficient fashion [13]. The original hyperbolic CORDIC algorithm states the following iterative equations:

$$\begin{aligned} X_{i+1} &= X_i + \delta_i Y_i 2^{-i} \\ Y_{i+1} &= Y_i + \delta_i X_i 2^{-i} \\ Z_{i+1} &= Z_i - \delta_i \theta_i \end{aligned} \tag{4.1}$$

Where $\theta_i = \tanh^{-1}(2^{-i})$ and i is the index of the iteration (i = 1, 2, 3, ... N). The value of δ_i is either +1 or –1 depending on the mode of operation:

$$\begin{aligned} &Rotation : \delta_i = -1 \quad if \quad z_i < 0, \quad +1 \quad otherwise \\ &Vectoring : \delta_i = -1 \quad if \quad x_i y_i \geq 0, \quad +1 \quad otherwise \end{aligned} \tag{4.2}$$

To obtain exp function we have to set some parameters according to [13], depending on the input/output bit width we want. In our circuit, we decided to use an input format of [24 20] (24 word length - 20 fractional length). The reason we chose this is to achieve a respectable LLR accuracy without making a huge-cost design.

The specific format means that from all the M distances multiplied by noise variance, only the ones greater or equal to -7 will be computed and added to the specific sum. This depends on the fractional length our output has (11 bits) and is predetermined from the format we choose for the implementation and the tradeoff between accuracy and utilization we want. Thus, for the rest of these distances the exp function is set to produce a zero output. By this way, keeping the higher values (≥ -7) results in a small loss of LLR accuracy, but in a low cost exponential design too.

Remez Algorithm

This kind of exponent implementation in VHDL concerns the Remez algorithm. The last can compute the best minimax rational approximation of the wanted degree for a real function on the interval [a, b]. As mentioned before, we kept again the inputs greater or equal to -7. Given this fact, the input interval is [-7, 0]. Using the trial and error method we split this interval into [-7, -5), [-5, -2), [-2, 0] for a smaller loss of accuracy. For each situation, a 2nd degree polynomial was produced via Mapple and is expressed below:

$$\exp(x) \approx \begin{cases} 0.068700 + (0.019282 + 0.0013733x)x & x \in [-7, 5) \\ 0.39881 + (0.17214 + 0.018946x)x & x \in [-5, 2) \\ 0.98347 + (0.82344 + 0.20382x)x & x \in [-2, 0] \end{cases} \tag{4.3}$$

where the maxerror for each expression respectively is:

1. maxerror = 0.00011679
2. maxerror = 0.0050554
3. maxerror = 0.016580

As the above equation is already an exponential approach, we had to represent these three coefficients as [30 29], in order to have a remarkable accuracy. The bit width of output remained the same as in CORDIC algorithm.

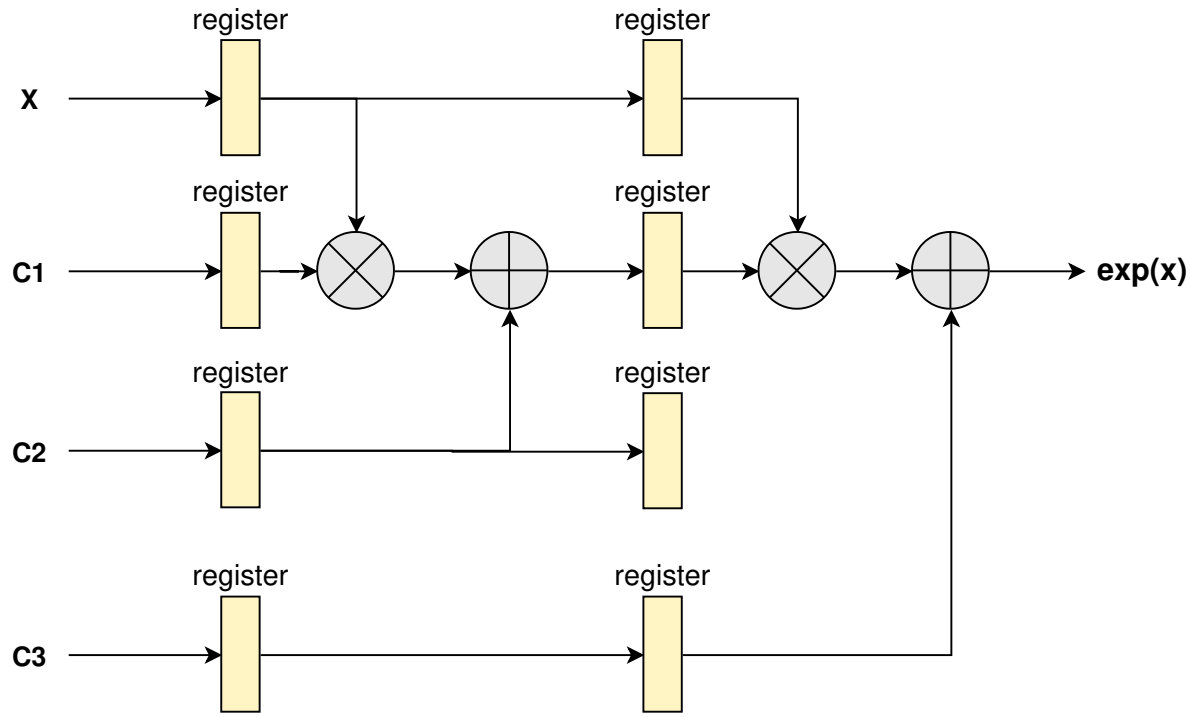


Figure 4.6: Block diagram of the 2nd degree polynomial in parallel and pipelined version

As shown in Figure 4.6 the specific polynomial was implemented in parallel so that the decoding algorithm demonstrates the maximum throughput. At every step a controller checks the value of x and determines the three coefficients.

4.3.2 Natural Algorithm

CORDIC

A fixed-point iterative architecture of the logarithm function based in the expanded hyperbolic CORDIC algorithm is analyzed at this section. Based on [12],

$$\ln(a) = 2\tanh^{-1}(\frac{a-1}{a+1}) \tag{4.1}$$

The function $ln(a)$ is obtained by multiplying by 2 the final result, provided that $Z_0 = 0$, $X_0 = a + 1$ and $Y_0 = a - 1$ from the equation 4.1.

The fixed-point fractional representation means that there is a strict range in which values are fluctuate. This implies that the natural logarithmic function has a minimum value can compute. As the input of ln component is strictly positive and has 10 fractional bits, this minimum value is 2^{-10}. Thus, the interval referred before is [2^{-10}, 1]. The output interval, now, is meant to be [$ln(2^{-10})$, 0]. As $ln(2^{-10})$ needs 4 integer bits, we picked the output format of [16 12].

Remez Algorithm

As before, the polynomial function of $ln(x)$ has domain [0.0009765625, 1]. Again, for a higher accuracy we split this interval into these three ones: [0.0009765625, 0.1), [0.1, 0.2), [0.2, 1]. The 2nd degree polynomials for each of these situations are expressed below:

$$\ln(x) \approx \begin{cases} -6.40627 + (124.773 - 901.976x)x & x \in [0.0009765625, 0.1) \\ -3.03882 + (8.55215 - 7.88225x)x & x \in [0.1, 0.2) \\ -1.5303 + (1.9361 - 0.41872x)x & x \in [0.2, 1] \end{cases} \tag{4.2}$$

where the maxerror for each expression respectively is:

1. maxerror = 0.646413
2. maxerror = 0.0407974
3. maxerror = 0.026485

In order to represent each coefficient as a fixed point number with less possible loss of accuracy we set a format of [30 19]. The schematic diagram is the same as in Figure 4.6, but with different coefficients and controller which determines their values.

4.4 Pipeline Parallelization

In digital telecommunication systems, decoding algorithms require very high operating frequencies. In order to compensate the effect of oversampling in feedforward architectures, parallelization and pipelining methods are applied for implementation of these three algorithms. By utilizing these optimizations, clock frequency of each subsystem is increased in a remarkable number of MHz.

After the successful connection between the modules, the whole system is synchronous and fully pipelined, as stated. This means that after the data has passed through the pipeline we get 1 output at every clock cycle. The latencies that every module has, are described by the following table.

Hardware Module	**Latency Cycles**	
	64-QAM	**256-QAM**
Exact - CORDIC	43	45
Exact - Polynomial	14	16
Approx LLR	4	6
Piecewise LLR	2	3

Throughput

The throughput of our circuits computed based on max clock frequency and are shown below:

- Approx LLR: 357 MSa/s
- Piecewise LLR: 555 MSa/s

4.5 Design Verification

Design verification is an essential step in the development of any circuit. It is a method of confirmation by examining and providing evidence that the design output meets the design input specifications.

In order to verify that theoretical simulations on Matlab and behavioral simulation on the hardware are what we expected, a verification workflow was created (Figure 4.7). It also helped us to obtain the BER results of the VHDL code. Matlab generates a noisy signal and converts it to binary data. This data is read by a VHDL testbench through an input file and then simulation output is written to an output txt file. Finally, Matlab reads that output and producer BER results through scripts.

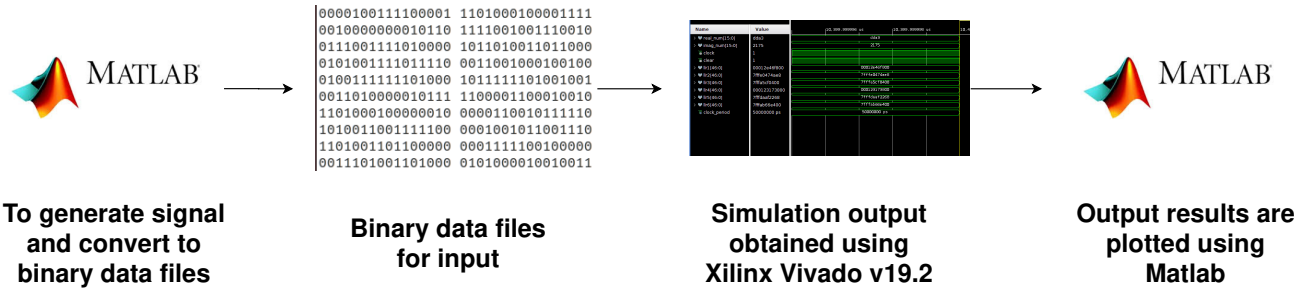


Figure 4.7: Typical workflow of feeding VHDL with Matlab inputs

For further verifying the proper operation of each algorithm, some LLR values of the test data of the Matlab results and the FPGA results are compared and the comparison result is demonstrated as follows in below figures.

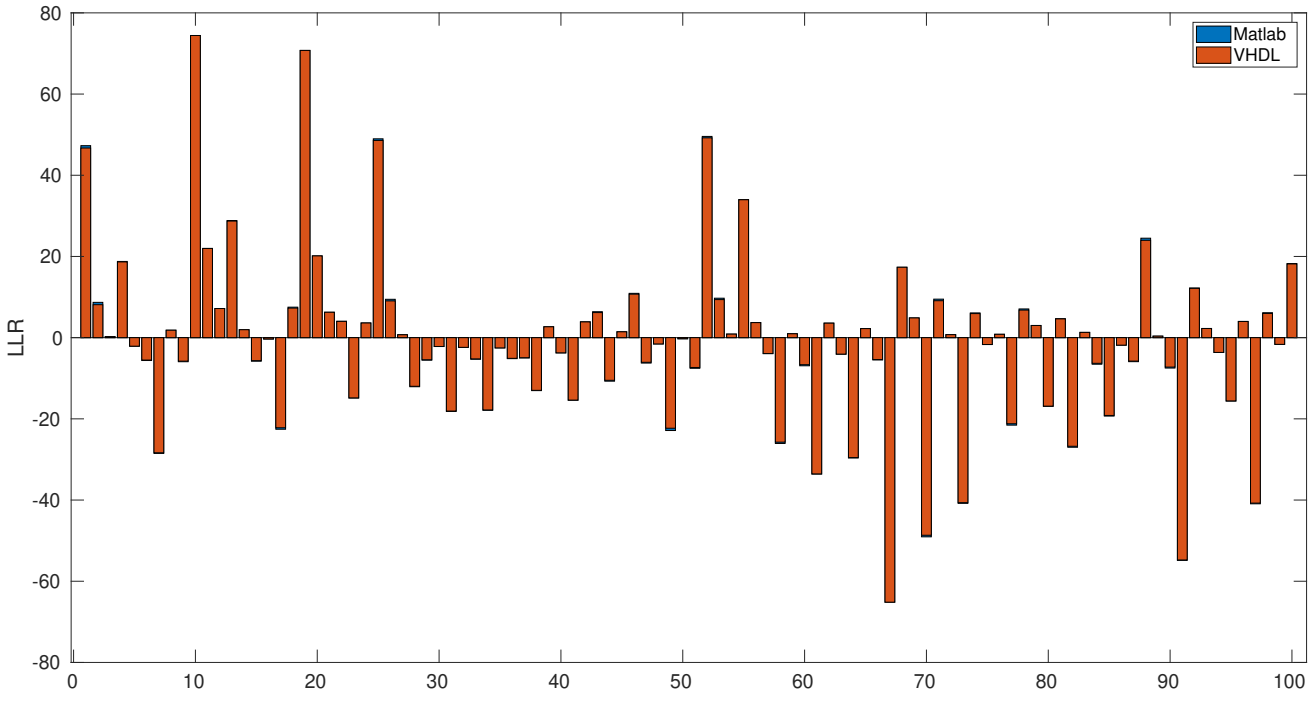


Figure 4.8: LLR comparison between VHDL code and Matlab code for 64-QAM Approx LLR algorithm for 100 samples

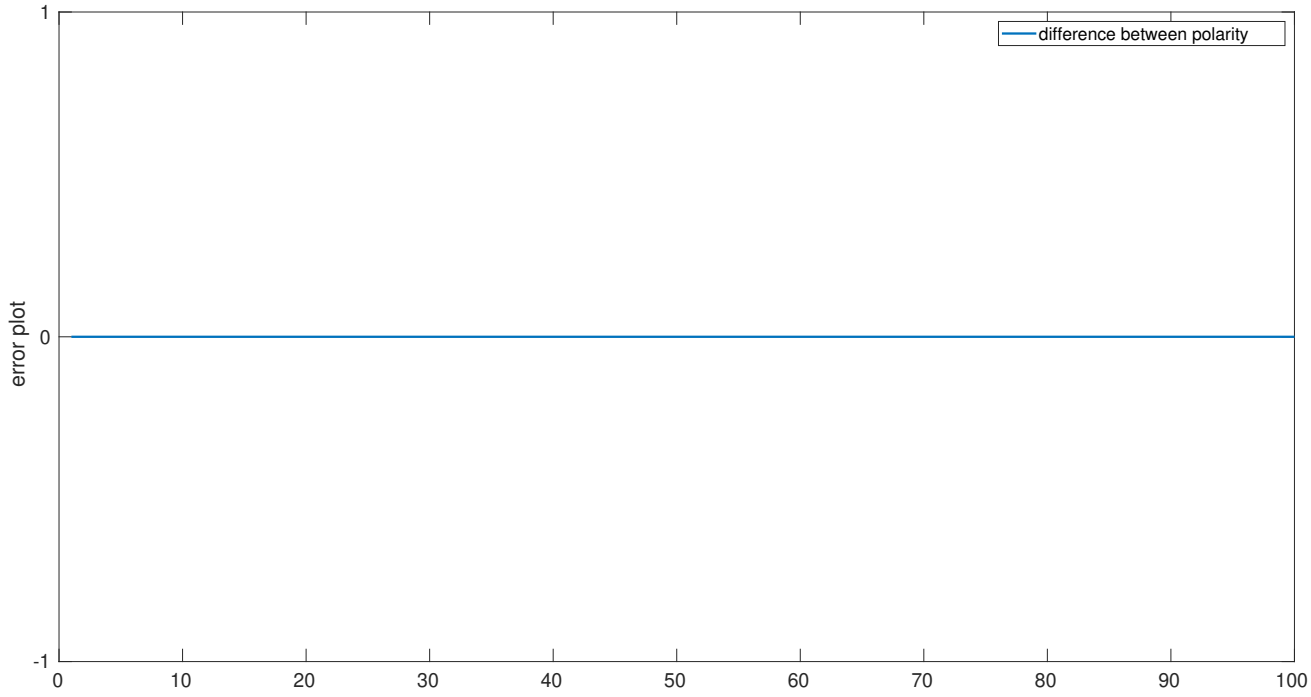


Figure 4.9: Comparison of the polarities of LLR between the simulation and the implementation for 64-QAM Approx LLR

Chapter 5

Experimental Results

5.1 Hardware Comparison of all Algorithms

For the purposes of exploring power consumption and verifying proper operation of the decoding algorithms, Vivado Design Suite 2019.2 and Zynq UltraScale + MPSoC ZCU106 Evaluation Platform were used. The soft decision decoding techniques chosen to be implemented are Exact LLR, Approximate LLR and Piecewise LLR. These algorithms tested for 64-QAM and 256-QAM. The arithmetic of these circuits is fixed-point and the approximate techniques that are applied are Bit Truncation and Radix Multiplication.

In many modulation or demodulation applications high performance, low power consumption and low cost ASIC design is required. For this reason, despite the fact that our circuits designed to fit on FPGA, we present some results without using DSPs. By this way, some comparisons and resources deduction can be observed for both of FPGA and ASIC circuits.

Our metrics for our hardware results are LLR Relative Error as described before (3.1) and LLR polarity reversal. The change of sign in LLR values makes it an equally important metric concerning the lowest values. These values are crucial and worth mentioning because in these there is a greater probability that the decoder picks the wrong codeword and thus a wrong decision about the received bit to be made. Greater values of LLR means higher confidence for the decoder. In addition, we compare these metrics with the resources that every algorithm needs to be implemented. The tables with the results of each implementations are shown below.

Exact LLR

Table 5.1: Evaluation of 64-QAM for Exact LLR with Cordic and Polyonomial implementation for exponent and natural logarithm in comparison with full-precision algorithm.

	64-QAM			
	CORDIC		POLYON	
	10DB	15DB	10DB	15DB
LLR Reversal Polarity	0.15%	0.17%	0.029%	0.037%
LLR Relative Error	47.25%	79.42%	46.25%	77.94%

DSP	0%	0%
LUTS	53.12%	63.51%
FF	17.73%	37.78%

For the Exact Algorithm two implementations were done. The first one concerns computing exponent and natural logarithm by using CORDIC architecture and the second one by polyonomial expressions. Although the second method would seem to be a more economical solution, in order to achieve the desired accuracy, more bits were needed to be used. However, a smaller deviation achieved.

As mentioned the Exact LLR requires complex computational operations. The fact that more than half of FPGA is required to implement a 64-QAM demodulation, makes this algorithm more ineffective than the other two.

Approximate LLR

Table 5.2: Accuracy results and Resources Utilization of 64 and 256 QAM for Approximate LLR in comparison with full-precision algorithm.

	64-QAM		256-QAM	
	10DB	15DB	15DB	20DB
LLR Reversal Polarity	0.00075%	0.000083%	0.0019%	0.00013%
LLR Relative Error	0.14%	0.04%	0.09%	0.023%
BER Variation	1.67×10^{-6}	0	3.75×10^{-6}	1.00×10^{-6}

DSP	0%	0%
LUTS	24.09%	97.28%
FF	4.69%	22%

The results from Table 5.2 show that Approximate LLR implementation is really close to its Full-Precision implementation (matlab). Here, a new metric has been added, called BER Variation. It concerns the difference between the number of error that these two

methods detected after the viterbi decoder. As we expected, this algorithm is much more economical than the Exact LLR.

Piecewise LLR

Table 5.3: Accuracy results and Resources Utilization of 64 and 256 QAM for Piecewise LLR in comparison with full-precision algorithm.

	64-QAM		**256-QAM**	
	10DB	**15DB**	**15DB**	**20DB**
LLR Reversal Polarity	0.0013%	0.00025%	0.0036%	0,00025
LLR Relative Error	0.04%	0.015%	0.073%	0.024%
BER Variation	3.33×10^{-6}	1.00×10^{-6}	2.00×10^{-5}	0

DSP	0%	0%
LUTS	0.07%	0.14%
FF	0.05%	0.09%

It is visible from the Table 5.3 that the Piecewise Algorithm is a significantly lower cost design than all the previous were tested. However, this achievement involves some BER performance loss.

5.2 Hardware Results

Introducing some approximation techniques, which already referred, to our circuits, we can analyze the impact and the cost they have. At this section, the following results show the tradeoff between the accuracy of LLR values and the resources deduction.

5.2.1 Truncation Approximation

In our design implementation and results we consider 16-bits inputs and 14-bits fractional length. In this method, we truncate the LSBs of the fractional part and by test and try we focus on achieving the desired tradeoff. Below, performance and utilization are presented for each soft decision algorithm, where T indicates the number of bits truncated of the internal operations.

Exact LLR

Table 5.4: Accuracy results and Resources Utilization of 64 QAM for Exact LLR using CORDIC, in comparison with full-precision algorithm.

	64-QAM CORDIC					
	10DB			15DB		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.17%	0.50%	4.69%	0.15%	0.21%	5.60%
LLR Relative Error	47.25%	60.09%	197.10%	79.42%	80.20%	86.9%

DSP	0%	0%	0%
LUTS	53.12%	40.21%	35.55%
FF	17.73%	16.85%	16.24%

Table 5.5: Accuracy results and Resources Utilization of 64 QAM Performance and Utilization for Exact LLR using POLYON, in comparison with full-precision algorithm.

	64-QAM POLYON					
	10DB			15DB		
	T=0	**T=8**	**T=11**	**T=0**	**T=8**	**T=11**
LLR Reversal Polarity	0.029%	0.50%	4.73%	0.037%	0.052%	2.02%
LLR Relative Error	46.25%	60.00%	200.00%	77.94%	78.74%	99.7%

DSP	0%	0%	0%
LUTS	63.51%	42.2%	27.3%
FF	37.78%	3.68%	3.21%

Here, we observe a significant reduction to the resources needed for the exact to be implemented. However, the growth of our metric is quite undesirable, as it will definitely lead to a crucial increase of bit error rate. That is the reason we focus on the next two soft decision algorithms later on.

Approximate LLR

Table 5.6: Accuracy results and Resources Utilization of 256 QAM for Approximate LLR in comparison with full-precision algorithm.

	256-QAM					
	15DB			20DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0.0019%	0.76%	8.9%	0.00013%	0.075%	7.35%
LLR Relative Error	0.09%	0.45%	219%	0.023%	8.6%	74.99%
BER Variation	3.75×10^{-6}	3.63×10^{-5}	0.0045	1.00×10^{-6}	5.00×10^{-6}	3.21×10^{-4}
DSP	0%	0%	0%			
LUTS	97.28%	38.64%	21.2%			
FF	22%	11.5%	7.85%			

Table 5.7: Accuracy results and Resources Utilization of 64 QAM for Approximate LLR in comparison with full-precision algorithm.

	64-QAM					
	10DB			15DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0.00075%	0.48%	4.64%	0.000083%	0.044%	1.94%
LLR Relative Error	0.14%	15.28%	142.94%	0.04%	5.31%	44.63%
BER Variation	1.67×10^{-6}	1.30×10^{-5}	0.0013	0	3.33×10^{-7}	1.52×10^{-5}
DSP	0%	0%	0%			
LUTS	24.09%	8.79%	4.48%			
FF	4.69%	2.32%	2.02%			

A remarkable conclusion from the tables above is that the resources reduction is even higher than exact algorithm and with less possible loss of accuracy. Given the fact that its design cost is lower and has better overall performance, Approximate LLR constitutes a better solution.

Piecewise LLR

Table 5.8: Accuracy results and Resources Utilization of 256 QAM for Piecewise LLR in comparison with full-precision algorithm.

	256-QAM					
	15DB			20DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0,0036%	1,65%	12,99%	0,00025%	0,35%	12,19%
LLR Relative Error	0,073%	38,75%	185,22%	0,024%	11,92%	71,36%
Bit Error Variation	2,00E-05	4,38E-05	0,071	0	1,25E-06	0,039
DSP	0%	0%	0%			
LUTS	0,14%	0,05%	0,03%			
FF	0,09%	0,05%	0,03%			

Table 5.9: Accuracy results and Resources Utilization of 64 QAM for Piecewise LLR in comparison with full-precision algorithm.

	64-QAM					
	10DB			15DB		
	T=0	T=8	T=11	T=0	T=8	T=11
LLR Reversal Polarity	0,00125%	0,71%	6,39%	0,00025%	0,072%	6,21%
LLR Relative Error	0,04%	13,14%	93,54%	0,015%	5,64%	39,37%
Bit Error Variation	3,33E-06	1,00E-04	0,0045	1,00E-06	1,83E-06	1,07E-04
DSP	0%	0%	0%			
LUTS	0,07%	0,03%	0,02%			
FF	0,05%	0,03%	0,02%			

The results here show that the truncation method does not fit well on Piecewise Algorithm, as the increasement of accuracy values are high for our tradeoff in resources. Nevertheless, this algorithm has no need for urgent approximations due to its low cost design.

The Chosen Algorithm

The algorithm chosen for further approximation techniques was Approximate LLR. Piecewise decoding needs zero multiplications and radix encoding cannot be applied there. Approx LLR has a low complexity compared to the Exact with no exponential and logarithmic and a better BER performance than the third algorithm (Figure 5.1).

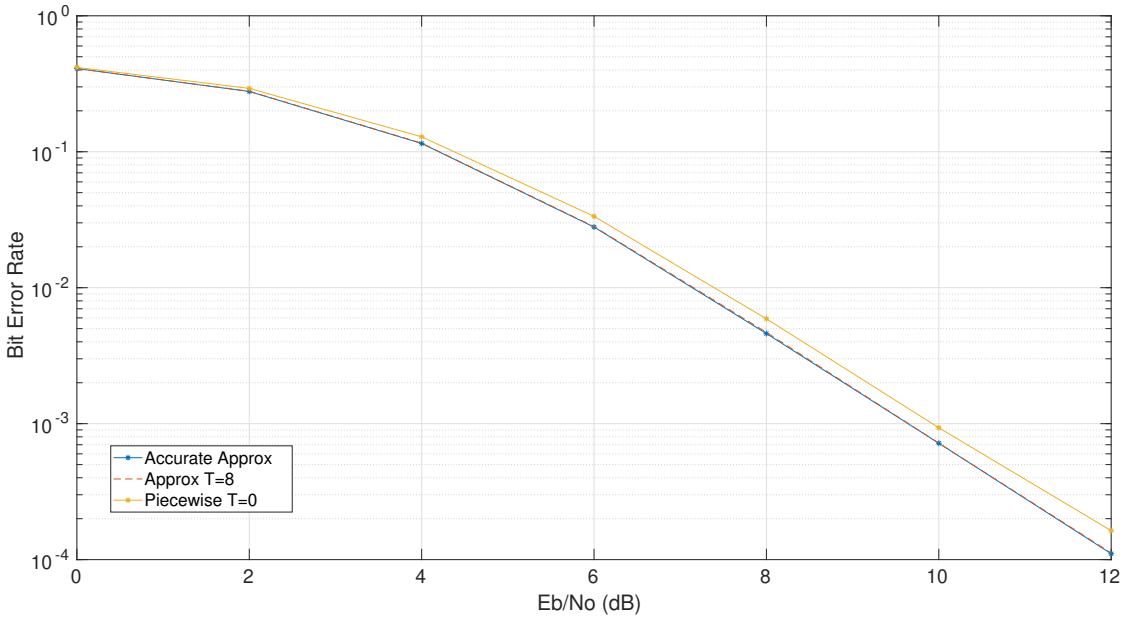


Figure 5.1: BER performance comparison between Accurate Approx, Approx $T = 8$ and Piecewise $T = 0$ in 64-QAM

In the next figure the hardware utilization for Approx LLR 64-QAM is derived using DSPs. It is visible that by using DSPs, LUTs are reduced by about 16%. It is worth mentioning that Approx LLR uses below 10% of the device's total resources.

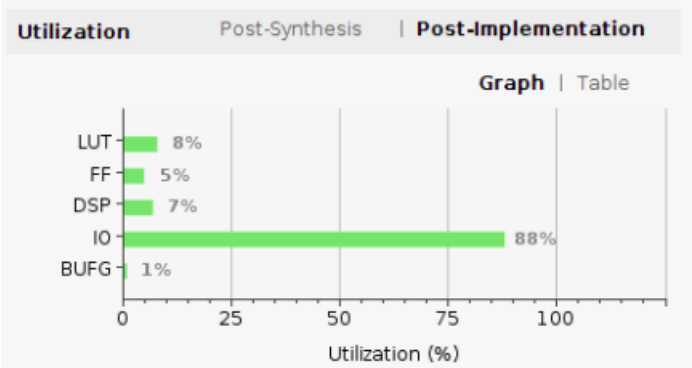


Figure 5.2: The utilization graph for the 64-QAM using DSPs as produced by the Vivado Implementation process

A schematic overview of the device utilization can be seen at the next Figure 5.3. The light blue areas symbolize the utilized fabric while the dark blue areas indicate the unused fabric. We can clearly see that our design has used least of the FPGA fabric.

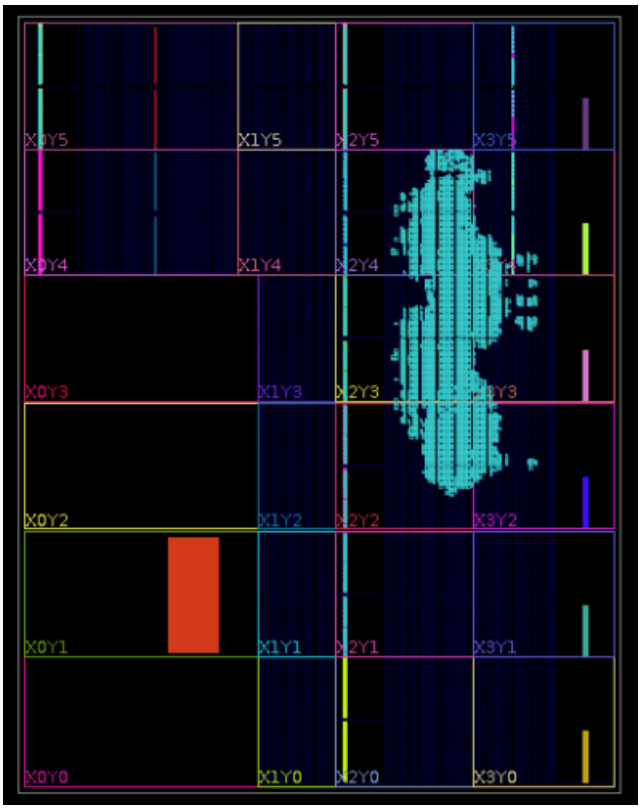


Figure 5.3: The FPGA device utilization as shown from the Vivado Implementation tool

5.2.2 Radix Approximation

At this point we exam the Radix Approximation on Approx LLR. We replace the accurate multipliers needed for the parallel euclidean distance calculation, with the radix multipliers. This supersession leads to energy savings in terms of cost and the results are presented below.

Table 5.10: Accuracy results and Resources Utilization of 64 QAM for Approx LLR using Radix Multipliers and $T = 0$, in comparison with full-precision algorithm.

	64-QAM			
	10DB		**15DB**	
	T=0	**K=10**	**T=0**	**K=10**
LLR Reversal Polarity	0.00075%	0.0058%	0.000083%	0.0013%
LLR Relative Error	0.14%	2.92%	0.04%	2.56%

DSP	0%	0%
LUTS	24.09%	16.37%
FF	4.69%	11.66%
LUTRAM	0%	1.13%

As it is shown from the Table 5.10 we sacrifice a little bit of precision, but the design cost has been highly decreased. The question that concerns is what its replica to BER performance.

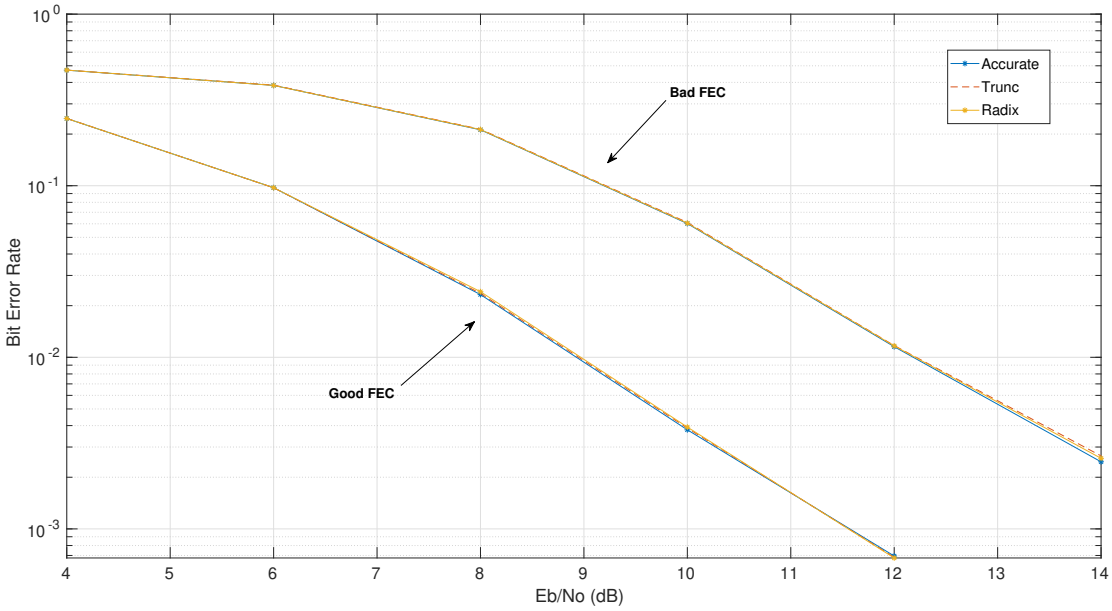


Figure 5.4: BER performance for Approx LLR of Accurate, Truncated (T=8) and Radix technique for different FEC encoders.

The answer is that the BER performance degretation is negligible and irrespective to the FEC encoder, as we can see from Figure 5.4. Both the truncation method and Radix approximation have almost the same behavior in terms of bit error rate. This happens and seems logic because we have a predetermined and sufficient length of fractional part in our inputs. In the next section, this number of bits will be decreased in order to draw a conclusion about which technique outperforms the other.

5.3 Approximation Techniques Comparison

Our goal here is to determine in which conditions Radix technique fits better than the Bit Truncation. For that purpose, we had to decrease the bit-length of input so that we notice a remarkable difference in BER performance. The fractional part is now 6 bits and the accuracy results are the following.

Table 5.11: Accuracy results and Resources Utilization of 64 QAM for Approx LLR at 10db with different number of fractional part (6 bits). LLR is compared to the Full-Precision and LUTS to the accurate algorithm (T=0).

	64-QAM		
	K=6	T=1	T=2
LLR Reversal Polarity	0.58%	1.18%	1.78%
LLR Relative Error	20.91%	11.23%	54.2%
Relative LUTS-gain	15.36%	9.01%	20.83%

In this situation, when the bit-length of input is decreased we notice a rapid increase in LLR values as concern the Truncation method. Contrariwise, the Radix approximation has a more stable change in its values, combining a remarkable gain in LUTs. Although the difference between the two techniques shown in Figure 5.5 is quite small (0.2-0.3db), we can draw a qualitative conclusion regarding how much we can lower the SNR requirement if we use FEC [17]. Depending on the FEC that we use, one technique may offer a better performance either in resources deduction or in better accuracy than the other.

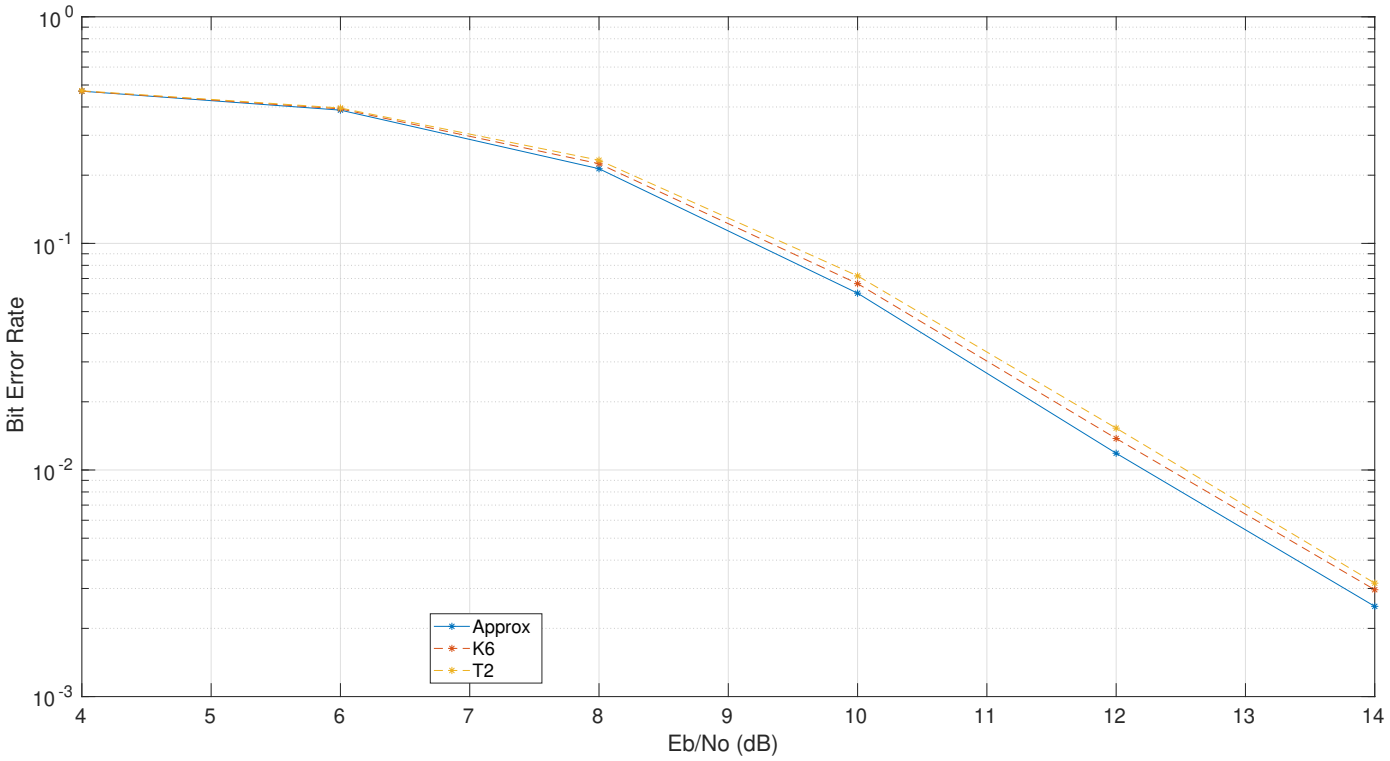


Figure 5.5: BER comparison of Truncation (T=2) and Radix method (K=6) for Approx LLR.

5.4 Overall Comparisons in Approximate LLR

In this section we evaluate both of the approximation techniques over the Approx LLR, in order to demonstrate the benefits of replacing accurate algorithms with the approximated ones. At this point, an error evaluation metric and a cost reduction metric are proposed, being called LLR mean relative error (LMRE) and relative LUTs reduction (RLR). The expressions for the last metrics are the following:

$$LMRE = \frac{\sum_{i=1}^{N} \frac{|LLR(i)_{accur} - LLR(i)_{approx}|}{|LLR(i)_{accur}|}}{N} \cdot 100\% \tag{5.1}$$

$$RLR = \frac{LUTS_{acc} - LUTSapprox}{LUTS_{acc}} \cdot 100\% \tag{5.2}$$

where i indicates the ith sample of the total number of N.

Figure 5.6 shows a comprehensive comparison of all situations by considering both RLR and LMRE. The purpose of the diagram is to extract the most efficient designs in terms of low cost-error and depending on the length of our inputs.

Generally, Truncation method attains the biggest RLR value and should be preferred when input's fractional length is high (e.g. 14 bits). However, the smaller this length is, the more efficient Radix method becomes. In this case, when error is of high importance Radix multiplier is more preferable, as truncating the LSB will lead to a massive reduction in accuracy. Hence, both the error and the utilization of resources depend on the examined application and the fixed-point representation we want to apply.

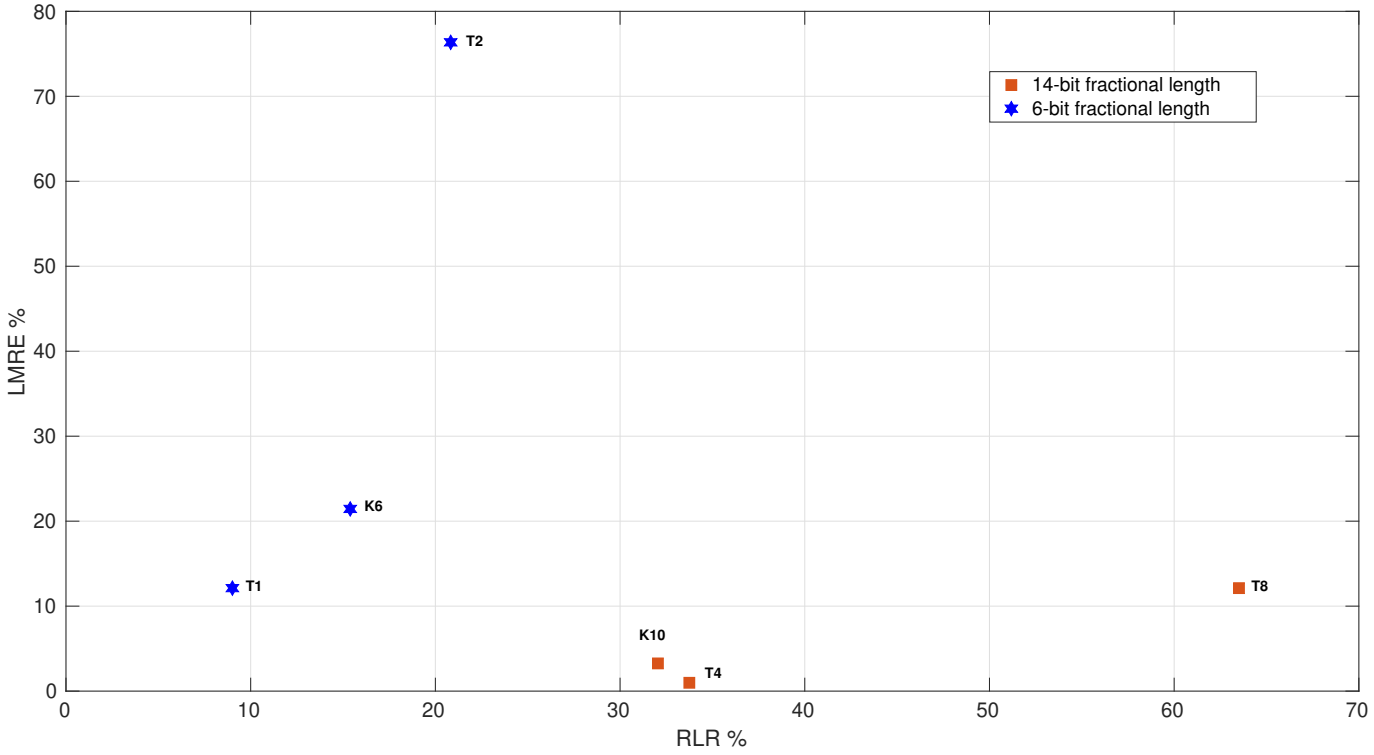


Figure 5.6: RLR-LMRE tradeoff of the examined approximated algorithms.

Chapter 6

Conclusions and Future Work

In the current thesis some soft decision decoding algorithms with approximate techniques were presented, assessed through simulations and finally implemented on an FPGA platform. Many telecommunication systems use also ASIC technology where effective solutions with minimum possible hardware overhead are needed. For that purpose, the present analysis focused on utilization of resources and the reduction of them using approximate techniques.

Approximate computing forms a design alternative that exploits the intrinsic error resilience of various applications and produces energy-efficient circuits with small accuracy loss. In this thesis we picked the most commonly used soft decision algorithms and tried to achieve a low cost circuit for a small error in accuracy. The algorithms that tested, simulated and implemented were Exact LLR, Approximate LLR and Piecewise LLR. We proved that although the last method is clearly lower in complexity than the others, the cost of Approx LLR design can be significantly reduced using approximate techniques. In addition, it can be approximated with a negligible loss in BER performance. The approximations that implemented were Bit Truncation and Radix multipliers. The first one showed that is generally more efficient when inputs in an application have big bit-length. The second one becomes more preferable when error is of a high important. This means that this method can withstand a telecommunication system it uses a bad-low cost FEC encoder. Thus, depending on the parameters, when implementing such a system some techniques can be more desirable as they perform just as well in any FEC encoding.

A future project could be an actual FPGA implementation of the approximated algorithms and embodiment to experiment with real telecommunication data and noise. Additionally, a more extensive exploration could be done including different FEC encoders and parameters in a telecommunication system.

Bibliography

[1] S. Chandrashekar. Advantages of fpga design methodologies. 2004.

[2] F. Didactic. *Quadrature Amplitude Modulation (QAM/DQAM)*, October 2016.

[3] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *IEEE European Test Symposium (ETS)*, pages 1–6, May 2013.

[4] M. Imani, R. Garcia, S. Gupta, and T. Rosing. RMAC: Runtime configurable floating point multiplier for approximate computing. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 12:1–12:6, Jul. 2018.

[5] M. Imani, D. Peroni, and T. Rosing. CFPU: Configurable floating point multiplier for energy-efficient computing. In *Design Automation Conference (DAC)*, pages 1–6, Jun. 2017.

[6] H. Jiang, J. Han, and F. Lombardi. A comparative review and evaluation of approximate adders. In *Great Lakes Symposium on VLSI*, pages 343–348, May 2015.

[7] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Design Automation Conference (DAC)*, pages 820–825, Jun. 2012.

[8] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris, and K. Pekmestzi. Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers. In *Design Automation Conference (DAC)*, pages 160:1–160:6, Jun. 2019.

[9] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi. Approximate hybrid high radix encoding for energy-efficient inexact multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):421–430, March 2018.

[10] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi. Walking through the energy-error pareto frontier of approximate multipliers. *IEEE Micro*, 38(4):40–49, Jul./Aug. 2018.

[11] Liu and Peiya. An open architecture for digital communication systems. In *IEEE Multimedia*, pages 79–84, 1994.

[12] D. Llamocca and C. Agurto. A fixed-point implementation of the natural logarithm based on a expanded hyperbolic cordic algorithm. In *XII Workshop IBERCHIP*, 2006.

[13] D. Llamocca and C. Agurto. A fixed-point implementation of the expanded hyperbolic cordic algorithm. In *Latin American applied research*, January 2007.

[14] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):62:1–62:33, May 2016.

[15] A. Morello and U. Reimers. Dvb-s2, the second generation standard for satellite broadcasting and unicasting. In *International Journal of Satellite Communications and Networking*, pages 249–268, 2004.

[16] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC)*, pages 1–6, Jun. 2015.

[17] Säckinger and Eduard. Forward error correction. In *Analysis and Design of Transimpedance Amplifiers for Optical Receivers*, pages 475–482, October 2017.

[18] Tasissa and Abiy. Function approximation and the remez algorithm. Technical report, 2020.

[19] F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved cofdm with application to hiperlan/2. In *2002 IEEE International Conference on Communications*, page 3, 2002.

[20] Viterbi and J. Andrew. An intuitive justification and a simplified implementation of the map decoder for convolutional codes. In *IEEE Journal on Selected Areas in Communications*, pages 260–264, February 1998.

[21] P. J. Woong, S. M. Hoon, K. P. Soo, and C. Dae-Ig. Multi-level modulation soft-decision demapper for dvb-s2. In *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, pages 13–17, 10 2009.

[22] Yoon and Eunchul. Maximum likelihood detection with a closed-form solution for the square qam constellation. In *IEEE Communications Letters*, pages 829–832, 4 2017.