

AN OPTIMISED FIELD PROGRAMMABLE GATE ARRAY (FPGA) JACOBIAN LOGARITHM ARCHITECTURE FOR TURBO CODES AND SOFT BIT DEMODULATION

LIM LI LI, MSc.

Thesis submitted to the University of Nottingham

for the degree of Doctor of Philosophy

NOVEMBER 2018

ABSTRACT

Turbo codes remains as a main contender for future fifth generation (5G) mobile standard and is widely adopted in various standards for wireless communication including deep space communication. The challenges presented for future wireless mobile standards requires the achievements of decoder clock speed exceeding the 500 MHz threshold and ultra-low power utilisation which is directly linked to hardware complexity.

This research aims to develop a fresh approach in approximation the correction function within the Jacobian logarithm as applied in Turbo decoding and quadrature amplitude soft bit demodulation; fuelled by fact that the correction function is the most complex yet, most frequent computation kernel in iterative Turbo decoding and soft bit demodulation. The exact correction function in its original form, is riddled with complex computation involving logarithm of exponential sum function which is complex to implement on hardware. In order to mitigate this, the suboptimal Log-MAP solutions approximates the exact correction function with simpler functions for reduced complexity and low power utilisation.

This thesis also aims to develop novel suboptimal Log-MAP approximations to the Jacobian logarithm which is Field Programmable Gate Array (FPGA) implementation friendly. The choice of FPGA as hardware platform is rooted in the motivation for the future of Software Defined Radio (SDR), capable of swift reconfiguration with Applied Specific Integrated Circuit (ASIC) performance level.

In this thesis, the distribution of argument $|x|$ to the correction function is characterised with varying Signal to Noise Ratio (SNR) conditions and shown

to be concentrated to lower region of $|x|$ where the correction function is most effective at low SNR. Further, this work also shows that accurate approximations to the exact correction function facilitates good Bit Error Rate (BER) performance with close adherence to the ideal Log-MAP solution. A design criteria is established which focuses on high accuracy for lower $|x|$ regions of the correction function yet hardware friendly.

As a result, two approximations, the Hybrid Log-MAP (HLM) and Best Fit Linear (BFL) algorithm are proposed with R-Squared accuracy measure of 0.9732 and 0.9932 respectively. The proposed algorithms are shown to possess the two best goodness of fit (GOF) statistics among linear approximations reviewed in the literature. The proposed approximations are evaluated in Turbo decoder standards and is shown to possess the closest performance to the exact Log-MAP solution with performance within 3×10^{-4} dB and 5×10^{-4} dB each for the mobile and deep space wireless standard's Turbo decoder. Simulations under inaccurate channel estimation conditions also shows that the proposed approximations consistently provide for the closest BER performance to the ideal Log-MAP solution.

Hardware implementation of the HLM, utilising highly optimised digital signal processing modules within the FPGA recorded low power utilisation of 0.015 W, capable of running at 891 MHz speed. The BFL, also catered to fixed point processing however, recorded the lowest power utilisation of 0.001 W with pure combinational circuitry capable of faster clock speed exceeding the maximum limit of 891 MHz of the hardware platform while preserving computational accuracy to the exact correction function.

Additionally, the Jacobian logarithm application to soft bit demodulation was also evaluated for the BFL approximation against the exact Log-MAP and Max Log-MAP solution that omits the correction function altogether. The BFL was

shown to provide for close adherence to the exact Log-MAP solution within 1.6×10^{-3} dB for 256-Quadrature Amplitude Modulation (QAM) and 2.7×10^{-3} dB for 1024-QAM in the mobile wireless system; and 4.3×10^{4} dB for 16-QAM and 6×10^{-4} dB for 256-QAM in deep space communication standard.

ACKNOWLEDGEMENTS

I wish to firstly thank God for His wisdom, guidance, and the many blessings in my life. I wish to also thank my supervisor, Dr. David Lim Wee Gin. I am most grateful for his kindness and positive motivation leading up to this stage. I wish to also thank my thesis co-supervisor, Dr. Nafizah Goriman Khan for her kind advice, and guidance. I am also grateful to my thesis examiners, Prof. Dr. Amin Malek Mohammadi and Assoc. Prof Dr. Chow Chee Onn for their valuable feedbacks in improving the thesis.

A big thank you to my family for their unrelenting support and for being there for me throughout the duration of this research. I can never thank my husband, Pak Kwan, Lexie, and my dear friends enough for their kindness, companionship, and encouragements that enables me to be resilient in the face of challenges.

I wish to also thank my colleagues who are supportive, kind, and understanding. My alma mater, and the organisation that I now serve in, Tunku Abdul Rahman University College for sponsoring my postgraduate studies. To the University of Nottingham Malaysia, for providing a supportive environment for part-time postgraduate students and to my students who I still learn so much from. Last but not least, to all the teachers in my life who demand nothing less than excellence from me.

LIST OF TABLES

TABLE 1-1 COMPARISON BETWEEN TURBO, LDPC, AND POLAR DECODERS 3

TABLE 2-1 PIECEWISE LINEAR (PL) APPROXIMATION AND R-SQUARED MEASURE FOR EACH SECTION 46

TABLE 2-2 NUMBER OF COMPUTATION CYCLES FOR DIFFERENCE EP 48

TABLE 2-3 NUMBER OF MATHEMATICAL COMPUTATION OF APPROXIMATIONS TO THE CORRECTION FUNCTION 53

TABLE 2-4 KINTEX AND VIRTEX ULTRASCALE+ DEVICE RESOURCES 62

TABLE 2-5 SIMULATION SETUP FOR QAM CONSTELLATION DIAGRAM AND EYE DIAGRAM 66

TABLE 2-6 MODULATION CONSTELLATION MAPPING REFERENCES, RECEIVED CONSTELLATION SYMBOLS, AND EYE DIAGRAM BEFORE EQUALISATION 67

TABLE 3-1 SIMULATION SETUP FOR NORMALISED DISTRIBUTION OF $|\boldsymbol{x}|$ 79

TABLE 3-2 KEY GVIM COMMANDS FOR DATA CLEANING 80

TABLE 3-3 SIMULATION SETUP FOR CCSDS TURBO DECODER 86

TABLE 3-4 GOODNESS OF FIT STATISTICS FOR THE CONSTANT LOG-MAP WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 93

TABLE 3-5 GOODNESS OF FIT STATISTICS FOR THE LINEAR LOG-MAP WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 94

TABLE 3-6 GOODNESS OF FIT STATISTICS FOR THE MULTISTEP LOG-MAP WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 96

TABLE 3-7 GOODNESS OF FIT STATISTICS FOR THE LINEAR CONSTANT LOG-MAP WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 97

TABLE 3-8 GOODNESS OF FIT STATISTICS FOR THE SECOND ORDER APPROXIMATION WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 98

TABLE 3-9 CONFIDENCE BOUNDS AND INTERVAL FOR 95% TO 99.99% FOR $\boldsymbol{c}$ AND $\boldsymbol{m}$ IN EQ. 3-7 103

TABLE 3-10 GOODNESS OF FIT STATISTICS FOR LINEAR RANGE OF HLM, WITH $\boldsymbol{\Delta x = 0.01, 0 \leq x \leq 1.5}$ AGAINST OTHER APPROXIMATIONS 105

TABLE 3-11 GOODNESS OF FIT STATISTICS FOR THE OVERALL RANGE OF $\boldsymbol{x}$ WITH $\boldsymbol{\Delta x = 0.01, 0 \leq x < 4}$ FOR HLM AGAINST OTHER APPROXIMATIONS 107

TABLE 3-12 ITERATIVE SEARCH PARAMETERS TO ARRIVE TO FINAL OPTIMAL VALUES, $\boldsymbol{\Delta x = 0.01}$ 111

TABLE 3-13 CONFIDENCE BOUNDS AND INTERVAL FOR 95% TO 99.99% FOR $\boldsymbol{c}$ AND $\boldsymbol{m}$ FOR BFL $\boldsymbol{x \leq 1.4347}$ 114

TABLE 3-14 CONFIDENCE BOUNDS AND INTERVAL FOR 95% TO 99.99% FOR $\boldsymbol{c}$ AND $\boldsymbol{m}$ FOR BFL $\boldsymbol{x \leq 1.4347}$ 115

TABLE 3-15 GOODNESS OF FIT STATISTICS FOR THE BFL APPROXIMATION WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ BASED ON EQ. 3-11 116

TABLE 3-16 PARAMETER VALUES OF BFL_N FOR EQ. 3-12 WITH VARYING $\boldsymbol{N}$ ROUNDED TO FOUR DECIMAL PLACES 117

TABLE 3-17 NUMERATOR ADJUSTMENTS FOR FURTHER OPTIMISATION TO BFL_N ROUNDED TO FOUR POINTS DECIMAL 119

TABLE 3-18 GOODNESS OF FIT STATISTICS FOR BFL_N WITH $\boldsymbol{\Delta x = 0.01}$ FOR $\boldsymbol{0 \leq x \leq 4}$ 122

TABLE 3-19 XILINX FPGA PRODUCT SPECIFICATIONS 123

TABLE 3-20 RESOURCES USED FOR THE HLM DESIGN IN XSG 125

TABLE 3-21 RESOURCES USED FOR THE BFL4 DESIGN IN XSG 132

TABLE 3-22 RESOURCES USED FOR THE BFL6 DESIGN IN XSG 137

TABLE 3-23 QUANTIZATION EFFECTS TO BFL6 CONSTANTS WITH RADIX $\boldsymbol{N = 6}$ 138

TABLE 3-24 RESOURCES USED FOR THE BFL8 DESIGN IN XSG 142

TABLE 3-25 QUANTIZATION EFFECTS TO BFL8 CONSTANTS WITH RADIX $\boldsymbol{N = 8}$ 143

TABLE 4-1 SUMMARY OF 4G-LTE ADVANCED TURBO DECODER SIMULATION PARAMETERS 148

TABLE 4-2 SUMMARY OF CCSDS SIMULATION PARAMETERS 152

TABLE 4-3 ORGANISATION OF SNR MISMATCH RESULTS FOR THE 4G-LTE ADVANCED TURBO DECODER 157

TABLE 4-4 ORGANISATION OF SNR MISMATCH RESULTS FOR THE CCSDS TURBO DECODER 168

TABLE 4-5 HLM UTILIZATION REPORT 180

TABLE 4-6 HLM TIMING ANALYSIS REPORT 181

TABLE 4-7 HLM POWER ANALYSIS 182

TABLE 4-8 BFL4 UTILIZATION REPORT 183
TABLE 4-9 BFL6 UTILIZATION REPORT 183
TABLE 4-10 BFL8 UTILIZATION REPORT 183
TABLE 4-11 SUMMARY OF UTILIZATION REPORT FOR BFLN 183
TABLE 4-12 BFLN TIMING ANALYSIS 184
TABLE 4-13 BFLN POWER ANALYSIS 185
TABLE 4-14 VIVADO RESOURCE UTILISATION REPORT 186
TABLE 4-15 VIVADO TIMING SUMMARY REPORT 187
TABLE 4-16 VIVADO POWER SUMMARY REPORT 188
TABLE 4-17 SOFT BIT DEMODULATION SIMULATION SETUP FOR 4G-LTE ADVANCED TURBO DECODER 189
TABLE 4-18 SOFT BIT DEMODULATION SIMULATION SETUP FOR CCSDS TURBO DECODER 194
TABLE 5-1 SUMMARY OF RESULTS BASED ON RESEARCH OBJECTIVES 200
TABLE 5-2 SUMMARY OF RESULTS FROM SIMULATION AND FPGA IMPLEMENTATION 204

LIST OF FIGURES

FIGURE 1-1 A DIGITAL COMMUNICATIONS SYSTEM 1

FIGURE 1-2 RESEARCH OVERVIEW 11

FIGURE 2-1 TURBO ENCODER STRUCTURE 16

FIGURE 2-2 A RSC ENCODER 17

FIGURE 2-3 TRELLIS DIAGRAM OF A RSC ENCODER WITH GENERATOR POLYNOMIAL $\boldsymbol{G} = \{\mathbf{7}, \mathbf{5}\}$ 18

FIGURE 2-4 THE 4G-LTE ADVANCED TURBO ENCODER SCHEMATIC 20

FIGURE 2-5 THE CCSDS TURBO ENCODER SCHEMATIC 21

FIGURE 2-6 TURBO ENCODER OUTPUTS 25

FIGURE 2-7 TURBO ENCODER PARITY BITS AFTER PUNCTURING FOR RATE $\boldsymbol{r} = \mathbf{1}/\mathbf{2}$ 25

FIGURE 2-8 PUNCTURED AND MULTIPLEXED DATA FOR TRANSMISSION 26

FIGURE 2-9 TURBO DECODER STRUCTURE 26

FIGURE 2-10 TRELLIS TRANSITION WITH CONSTRAINT LENGTH $\boldsymbol{K} = \mathbf{3}$ 29

FIGURE 2-11 TRELLIS TRANSITION AND COMPUTATION FOR STATE METRICS 31

FIGURE 2-12 THE CONSTANT LOG-MAP PLOT AGAINST THE EXACT CORRECTION FUNCTION 41

FIGURE 2-13 THE LINEAR LOG-MAP PLOT AGAINST THE EXACT CORRECTION FUNCTION 42

FIGURE 2-14 THE MULTISTEP LOG-MAP PLOT AGAINST THE EXACT CORRECTION FUNCTION 43

FIGURE 2-15 THE LINEAR CONSTANT LOG-MAP PLOT AGAINST THE EXACT CORRECTION FUNCTION 45

FIGURE 2-16 BER PERFORMANCE FOR LC WITH BLOCK LENGTH 3000 AND 3 ITERATIONS 45

FIGURE 2-17 THE PIECEWISE LINEAR LOG-MAP PLOT AGAINST THE EXACT CORRECTION FUNCTION 47

FIGURE 2-18 BER PERFORMANCE FOR MTS FOR 3 AND 5 EP AGAINST RADIX 4 LOG-MAP AND RADIX 4 LUT METHOD 49

FIGURE 2-19 SECOND ORDER APPROXIMATION AGAINST THE EXACT CORRECTION FUNCTION 50

FIGURE 2-20 TURBO CODE AND SOFT BIT SIMULATION SYSTEM ENVIRONMENT 54

FIGURE 2-21 TURBO CODE OVER A FADING CHANNEL FOR THE WIRELESS MOBILE 58

FIGURE 2-22 BASIC FUNCTIONALITY OF THE DSP48E2 [93] 63

FIGURE 2-23 SYSTEM DESIGN FLOW 64

FIGURE 3-1 OVERVIEW OF CHAPTER 3: METHODOLOGY AND DESIGN 77

FIGURE 3-2 SNAPSHOT OF .TXT FILE IN GVIM DISPLAYING FORMAT OF RAW DATA FOR 1000 BITS SIMULATED 80

FIGURE 3-3 NORMALISED DISTRIBUTION OF $|\boldsymbol{x}|$ IN FORWARD STATE METRIC, $\boldsymbol{\alpha k}(\boldsymbol{s})$ COMPUTATION 82

FIGURE 3-4 NORMALISED DISTRIBUTION OF $|\boldsymbol{x}|$ IN BACKWARD STATE METRIC, $\boldsymbol{\beta k - 1}(\boldsymbol{s'})$ COMPUTATION 82

FIGURE 3-5 NORMALISED DISTRIBUTION OF $|\boldsymbol{x}|$ IN LLR, $\boldsymbol{L}(\boldsymbol{uk}|\boldsymbol{y})$ COMPUTATION 83

FIGURE 3-6 COMBINED NORMALISED DISTRIBUTION OF $\boldsymbol{x}$ FOR STATE METRICS AND LLR COMPUTATION 84

FIGURE 3-7 THE CLM AND LLM CORRECTION FUNCTION 85

FIGURE 3-8 BER PERFORMANCE OF SEVERAL SUBOPTIMAL LOG-MAP ALGORITHMS 87

FIGURE 3-9 ENLARGED GRAPH FOR LOW $\boldsymbol{Eb/N0}$ 87

FIGURE 3-10 THE EXACT CORRECTION FUNCTION, $\boldsymbol{fc}(\boldsymbol{x})$ 89

FIGURE 3-11 COMPUTATION OF MEAN ERROR DISTRIBUTION FOR AN APPROXIMATE CORRECTION FUNCTION, $\boldsymbol{fapprox}(\boldsymbol{x})$ 89

FIGURE 3-12 MEAN SSE ERROR DISTRIBUTION FOR THE CLM WITH $\boldsymbol{T = 1.5}$ AND $\boldsymbol{C = 1/2}$ 93

FIGURE 3-13 MEAN ERROR DISTRIBUTION FOR THE LINEAR LOG-MAP 94

FIGURE 3-14 MEAN ERROR DISTRIBUTION FOR THE MULTISTEP LOG-MAP 95

FIGURE 3-15 MEAN ERROR DISTRIBUTION FOR THE LINEAR CONSTANT LOG-MAP 96

FIGURE 3-16 MEAN ERROR DISTRIBUTION FOR THE SECOND ORDER APPROXIMATION 97

FIGURE 3-17 THE HYBRID LOG-MAP (HLM) APPROXIMATION DESIGN FLOW 101

FIGURE 3-18 LINEAR FIT MATLAB SCRIPT AND CONFIDENCE BOUNDS COMPUTATION 102

FIGURE 3-19 HLM APPROXIMATION AGAINST THE EXACT CORRECTION FUNCTION 104

FIGURE 3-20 MEAN ERROR DISTRIBUTION FOR THE HLM APPROXIMATION 106

FIGURE 3-21 SEARCH AREAS FOR BFL OPTIMAL VALUES 108

FIGURE 3-22 ITERATIVE SEARCH FOR $\boldsymbol{0}$, $\boldsymbol{COP}$, $\boldsymbol{BOP}$, $\boldsymbol{DOP}$, AND $\boldsymbol{FOP}$, $\boldsymbol{0}$ 110

FIGURE 3-23 BFL APPROXIMATION AGAINST THE EXACT CORRECTION FUNCTION 113

FIGURE 3-24 BFL CURVE FIT VERIFICATION FOR $x \leq 1.4347$ 113
FIGURE 3-25 BFL CURVE FIT VERIFICATION FOR $1.4347 < x < 3.6865$ 114
FIGURE 3-26 MEAN SSE ERROR DISTRIBUTION FOR THE BFL APPROXIMATION 115
FIGURE 3-27 FIXED POINT QUANTIZATION EFFECTS 117
FIGURE 3-28 PLOT OF BFL_4 CORRECTION FUNCTIONS 120
FIGURE 3-29 PLOT OF BFL_6 CORRECTION FUNCTIONS 121
FIGURE 3-30 PLOT OF BFL_8 CORRECTION FUNCTIONS 121
FIGURE 3-31 HLM APPROXIMATION AS IMPLEMENTED ON XSG 124
FIGURE 3-32 HLM TOP LEVEL SCHEMATIC IN VIVADO 127
FIGURE 3-33 HLM SCHEMATIC LOGIC BLOCK ARRAY LEVEL VIEW 129
FIGURE 3-34 BFL4 APPROXIMATION AS IMPLEMENTED ON XSG 131
FIGURE 3-35 BFL4 TOP LEVEL SCHEMATIC IN VIVADO 134
FIGURE 3-36 BFL4 SCHEMATIC LOGIC BLOCK ARRAY LEVEL VIEW 135
FIGURE 3-37 BFL6 APPROXIMATION AS IMPLEMENTED ON XSG 136
FIGURE 3-38 BFL6 TOP LEVEL SCHEMATIC IN VIVADO 139
FIGURE 3-39 BFL6 SCHEMATIC LOGIC BLOCK ARRAY LEVEL VIEW 140
FIGURE 3-40 BFL8 APPROXIMATION AS IMPLEMENTED ON XSG 141
FIGURE 3-41 BFL8 TOP LEVEL SCHEMATIC IN VIVADO 144
FIGURE 3-42 BFL8 SCHEMATIC LOGIC BLOCK ARRAY LEVEL VIEW 145
FIGURE 4-1 SUMMARY AND FLOW OF RESULTS PRESENTATION 146
FIGURE 4-2 4G-LTE ADVANCE SUBOPTIMAL LOG-MAP BER PERFORMANCE 149
FIGURE 4-3 ENLARGED HIGHLIGHTED REGION OF FIGURE 4-2 150
FIGURE 4-4 CCSDS SUBOPTIMAL LOG-MAP BER PERFORMANCE 153
FIGURE 4-5 ENLARGED AREA HIGHLIGHTED IN FIGURE 4-4 154
FIGURE 4-6 CCSDS SUBOPTIMAL LOG-MAP BER PERFORMANCE FOR $\boldsymbol{BFLN}$ 155
FIGURE 4-7 HLM, BFL, AND CLM SNR MISMATCH SIMULATION WITH $\sigma m2 = 0.1$ AT TRUE SNR OF 2.4 DB AND 2.8 DB 158
FIGURE 4-8 LLM, MLM, AND LC SNR MISMATCH SIMULATION WITH $\sigma m2 = 0.1$ AT TRUE SNR OF 2.4 DB AND 2.8 DB 159

FIGURE 4-9 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR $\boldsymbol{\sigma m2 = 0.1}$ AT 2.4 DB 161

FIGURE 4-10 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR $\boldsymbol{\sigma m2 = 0.1}$ AT 2.8 DB 162

FIGURE 4-11 HLM, BFL, AND CLM SNR MISMATCH SIMULATION WITH $\boldsymbol{\sigma m2 = 0.4}$ AT TRUE SNR OF $\boldsymbol{6.4}$ DB, $\boldsymbol{7}$ DB, AND 7.7 DB 163

FIGURE 4-12 LLM, MLM, AND LC SNR MISMATCH SIMULATION WITH $\boldsymbol{\sigma m2 = 0.4}$ AT TRUE SNR OF $\boldsymbol{6.4}$ DB, $\boldsymbol{7}$ DB, AND 7.7 DB 164

FIGURE 4-13 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR $\boldsymbol{\sigma m2 = 0.4}$ AT 6.4 DB 165

FIGURE 4-14 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR $\boldsymbol{\sigma m2 = 0.4}$ AT 7 DB 166

FIGURE 4-15 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR $\boldsymbol{\sigma m2 = 0.4}$ AT 7.7 DB 167

FIGURE 4-16 HLM, BFL, AND CLM SNR MISMATCH SIMULATION AT TRUE SNR OF $\boldsymbol{0.3}$ DB, 0.8 DB, AND 1.0 DB 169

FIGURE 4-17 LLM, MLM, AND LC SNR MISMATCH SIMULATION AT TRUE SNR OF $\boldsymbol{0.3}$ DB, 0.8 DB, AND 1.0 DB 170

FIGURE 4-18 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 0.3 DB 171

FIGURE 4-19 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 0.8 DB 172

FIGURE 4-20 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 1.0 DB 173

FIGURE 4-21 HLM, BFL, AND CLM SNR MISMATCH SIMULATION AT TRUE SNR OF $\boldsymbol{1.3}$ DB, 1.5 DB, AND 1.8 DB 173

FIGURE 4-22 LLM, MLM, AND LC SNR MISMATCH SIMULATION AT TRUE SNR OF $\boldsymbol{1.3}$ DB, 1.5 DB, AND 1.8 DB 174

FIGURE 4-23 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 1.3 DB 175

FIGURE 4-24 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 1.5 DB 176

FIGURE 4-25 BER DIFFERENCES OF SUBOPTIMAL LOG-MAP ALGORITHMS FOR THE CCSDS DECODER AT 1.8 DB 176

FIGURE 4-26 ***BFLN*** SNR MISMATCH SIMULATION AT TRUE SNR OF **0. 3** DB, 0.8 DB, AND 1.0 DB 177

FIGURE 4-27 ***BFLN*** SNR MISMATCH SIMULATION AT TRUE SNR OF **1. 3** DB, 1.5 DB, AND 1.8 DB 178

FIGURE 4-28 256-QAM SOFT BIT DEMODULATION BER FOR 4G-LTE ADVANCED TURBO DECODER 190

FIGURE 4-29 ENLARGED HIGHLIGHTED AREA OF 256-QAM SOFT BIT DEMODULATION BER FOR 4G-LTE ADVANCED TURBO DECODER 191

FIGURE 4-30 1024-QAM SOFT BIT DEMODULATION BER FOR 4G-LTE ADVANCED TURBO DECODER 192

FIGURE 4-31 ENLARGED HIGHLIGHTED AREA OF 1024-QAM SOFT BIT DEMODULATION BER FOR 4G-LTE ADVANCED TURBO DECODER 193

FIGURE 4-32 16-QAM SOFT BIT DEMODULATION BER FOR CCSDS TURBO DECODER 195

FIGURE 4-33 ENLARGED HIGHLIGHTED AREA OF 16-QAM SOFT BIT DEMODULATION BER FOR CCSDS TURBO DECODER 196

FIGURE 4-34 256-QAM SOFT BIT DEMODULATION BER FOR CCSDS TURBO DECODER 197

FIGURE 4-35 ENLARGED HIGHLIGHTED AREA OF 256-QAM SOFT BIT DEMODULATION BER FOR CCSDS TURBO DECODER 198

LIST OF COMMON ABBREVIATION AND ACRONYMS

MAP	Maximum A Posteriori
BER	Bit Error Rate
4G	Fourth Generation Mobile Telecommunication
LTE	Long Term Evolution
FPGA	Field Programmable Gate Array
RSC	Recursive Systematic Convolutional
LLR	Log Likelihood Ratio
SNR	Signal to Noise Ratio
QAM	Quadrature Amplitude Modulation
CCSDS	Consultative Committee for Space Data Systems
ASIC	Application Specific Integrated Circuit
DSP	Digital Signal Processing / Digital Signal Processor
LUT	Look Up Table
SDR	Software Defined Radio
EPROM	Erasable Programmable Read-Only-Memory
XSG	Xilinx System Generator
GOF	Goodness of Fit
CLM	Constant Log-MAP
LLM	Linear Log-MAP
MLM	Multistep Log-MAP
LC	Linear Constant Log-MAP
FLM	Fitted Log-MAP/Second Order Approximation
BFLN	Best Fit Linear Log-MAP (N Radix Points)
CCSDS	The Consultative Committee for Space Data Systems

TABLE OF CONTENTS

Abstract ..... **i**

Acknowledgements ..... **iv**

List of Tables ..... **v**

List of Figures ..... **viii**

List of Common Abbreviation and Acronyms ..... **xiii**

Table of Contents ..... **xiv**

1 Introduction **1**

1.1 OVERVIEW 1

1.1.1 Channel Encoder and Decoder *2*

1.1.2 Soft Demodulation *6*

1.1.3 Hardware Implementation for Channel Decoding *7*

1.2 PROBLEM STATEMENT AND RESEARCH MOTIVATION 8

1.3 RESEARCH AIM AND OBJECTIVES 10

1.4 SCOPE OF WORK 10

1.5 THESIS CONTRIBUTION 12

1.6 ORGANISATION OF THESIS 13

2 Literature Review **15**

2.1 INTRODUCTION 15

2.2 TURBO CODES: PRINCIPLES OF OPERATION 16

2.2.1 Turbo Encoding *16*

2.2.2 The Recursive Systematic Turbo Encoder (RSC) *17*

2.2.3 Turbo Encoding Standards for Wireless Communication *19*

2.2.4 Turbo Interleavers *22*

2.2.5 Puncturing and Multiplexing *24*

2.2.6 Turbo Decoding *26*

2.2.7 The MAP Algorithm *28*

2.2.8 The Max-Log-MAP Algorithm *34*

2.2.9 The Log-MAP Algorithm *37*

2.2.10 The Suboptimal Log-MAP Turbo Decoder *38*

2.2.11 Turbo Code Systems Simulation Environment *54*

2.2.12 The Turbo Decoder under SNR Mismatch Conditions *58*

2.3 OVERVIEW OF FPGA TECHNOLOGY 60

2.3.1 Review of Xilinx FPGAs *62*

2.3.2 System Generator for DSP, (XSG) Simulation Environment *63*

2.4 M-QAM Modulation and Soft Bit Demodulation for Wireless Communication 65
2.4.1 M-QAM Soft Demapping 73
2.4.2 Efforts to Simplify the ML Soft Demapping in Literatures 75

3 Methodology and Design 77

3.1 Introduction 77
3.2 Analysis of Argument $|x|$ Distribution with Varying SNR 78
3.2.1 BER Performance Analysis of Approximations to the Correction Function 85
3.3 Correction Function Approximation Accuracy Analysis 88
3.3.1 Computation of Mean Error Distribution 89
3.3.2 Computation of Goodness of Fit (GOF) Statistics 90
3.3.3 Accuracy Analysis of Approximations to the Correction Function 92
3.4 A Fresh Approach to the Development of Suboptimal Log-MAP Algorithm 99
3.4.1 Design Criteria for an Optimal Approximation to the Correction Function 99
3.4.2 The Hybrid Log-MAP (HLM) Approximation 101
3.4.3 The Best Fit Linear (BFL) Approximation 107
3.5 Design of FPGA Architecture for Proposed Approximations 122
3.5.1 HLM FPGA Architecture Design 124
3.5.2 BFL FPGA Architecture Design 130

4 Results and Discussion 146

4.1.1 Introduction 146
4.2 Suboptimal Log-MAP BER Performance 147
4.2.1 Performance of Various Suboptimal Log-MAP under the Eight States 4G-LTE Advanced Turbo Decoder 147
4.2.2 Performance of Various Suboptimal Log-MAP under the CCSDS Decoder 152
4.3 Performance under SNR Mismatch 156
4.3.1 SNR Mismatch Simulation under the 4G-LTE Advanced decoder 156
4.3.2 SNR Mismatch Simulation under the CCSDS decoder 168
4.4 FPGA Implementation Evaluation for Proposed Approximations 178
4.4.1 HLM Approximation 179
4.4.2 BFL Approximation 182
4.5 Comparative Study between Proposed Approximations and Existing Solutions 185
4.6 Soft Bit Demodulation Simulation Results 189
4.6.1 Performance of M-QAM soft bit demodulation with 8 states Turbo decoder 189
4.6.2 Performance of M-QAM soft bit demodulation with 16 states Turbo decoder 194

5 Conclusion and Further Work 199

5.1 Conclusion and Summary of Results 199
5.1.1 Overall Assessment, of Jacobian Logarithm Approximation for FPGA implementation in Turbo Decoding and Soft Bit Demodulation 202
5.2 Future Work 205

References **207**

List of Publications **218**

Appendix A **219**

Appendix B **221**

Appendix C **223**

Appendix D **227**

Appendix E **230**

Appendix F **232**

Appendix G **240**

Appendix H **245**

Appendix I **248**

Appendix J **250**

Appendix K **253**

Appendix L **257**

1 INTRODUCTION

1.1 Overview

In our modern era of wireless communication, the emphasis for connectivity among devices, low device power consumption, reception reliability, and data speed are ever demanding. The fifth generation wireless system (5G) is touted as the next big step in wireless communication that made its debut in the 2018, Pyeong Chang Winter Olympics which is the culmination of existing and developing solutions based on previous generations of wireless technology [1], [2]. 5G demands high throughput of up to 20 Gbps downlink, ultra-low latencies, ultra-reliable [3], [4] and low power consumption to include support for various application areas including broadband data, Internet of Things (IoT), vehicular communications, and cloud computing [5].

Figure 1-1 depicts the general block diagram of a wireless communication system that illustrates the processes that the input data has to undergo before wireless transmission over the channel and subsequently subjected to the reversed processes pre transmission to correctly decode the intended data bits sent.

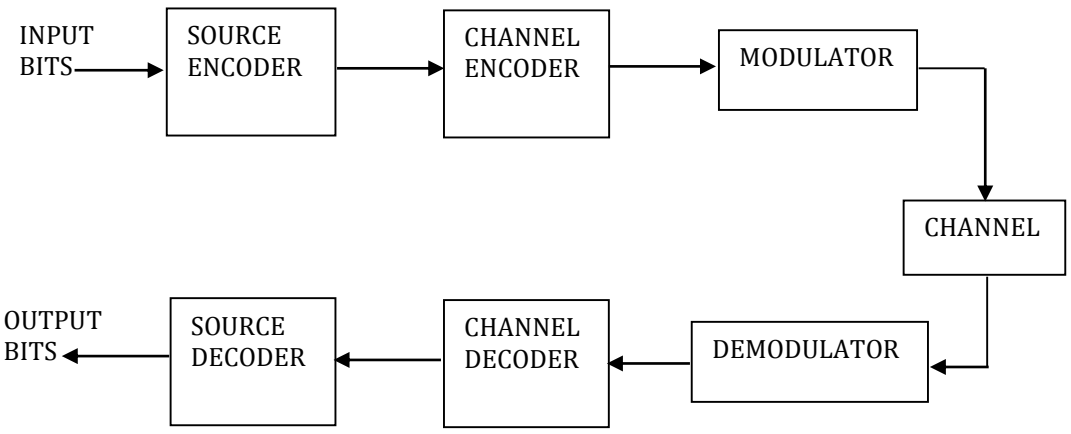


Figure 1-1 A digital communications system

1.1.1 Channel Encoder and Decoder

Channel Coding or Forward Error Correction (FEC) stems from Claude E. Shannon's pioneering work which predicts that a reliable communication system is achievable with the aid of channel coding. The channel encoder typically adds code bits to the transmission bit streams which are then exploited by the channel decoder at the receiving end to correct errors inflicted to the transmitted signal caused by noise, interference, and poor signal strength. Despite the addition of extra code bits, increased bandwidth to support higher transmission symbol rate over the channel, and greater hardware complexity to support complex decoding algorithms, channel coding remains the best option to elevate the system's performance approaching Shannon's limit.

The main requirements for 5G channel coding concerns with high throughput, low latency, low energy consumption, and low hardware complexity [6], [7]. At present, there are three main contenders for the 5G channel coding system, namely, the Turbo code , the Low Density Parity Check (LDPC) code, and Polar code [6]–[9].

Table 1-1 shows the relative comparison between the three contenders [10]. Table cells which are shaded in grey denotes deterring features of the relevant codes for ease of adaptation in future wireless communication systems.

Table 1-1 Comparison between Turbo, LDPC, and Polar decoders

Characteristic	Turbo decoder	LDPC decoder	Polar decoder
Maturity	Proven in 3G [11] and 4G [12]	Proven in WiFi [13]	Unproven [10]
Throughput and latency	Exceeding 20 Gbps [14]	Exceeding 20 Gbps [14]	Exceeding 20 Gbps [14]
Flexibility	Flexible [10]	Inflexible [10]	Inflexible [10]
Computation complexity	Higher for most coding rate [15]	Lower for most coding rate [15]	Lower for most coding rate [15]
Error correction capability	Similar [15]	Similar [15]	Similar to list decoding [15]
Backward compatibility	Yes [10]	No [10]	No [10]

Historically, Turbo codes has played a vital role in enabling 3G, and 4G systems [6] while the LDPC code were adopted in WiFi [16] and WiMax standards [17] along with the relatively new Polar code [18]. In terms of technology maturity, Turbo and LDPC codes are more established and adopted in several standards, while Polar codes remains relatively new and unproven except in publications and test scenarios [10]. In terms of error correction capabilities, throughput and latency, all codes have demonstrated capabilities exceeding 20Gbps with short latencies although with differing flexibilities to each implementation [19]–[21]. It has been shown that Turbo code allows full flexibility with regards to block length and coding rate making it fully compatible with older 3G and 4G standards [19]. However, the same cannot be said for both LDPC and Polar decoder as the performance is highly dependent to a particular block length, code rate for LDPC [10], [20], and pipelining approach especially in Polar codes [21] rendering it inflexible when compared with other decoding techniques. However, one drawback of the Turbo decoder when the coding rate is made constant across decoding techniques is higher computation complexity which is largely attributed to its iterative nature. Despite this, from an overall perspective, Turbo code remains attractive as the answer to future demands of 5G especially when considering backwards compatibility. Ideally the 5G

standard should be able to support older 3G and 4G technology lest devices are forced to have two separate pieces of hardware to support both 5G and older technologies [10]. This would incur an overhead since the channel decoder occupies up to 72% of baseband processor resources and energy consumption [22]. In addition, Turbo code offers shorter development time and reduced overall cost due to backward compatibility.

Additionally, Turbo codes are used widely in recent works [23]–[26] involving Internet of Things (IoT) application that has since been announced as part of the service category of NarrowBand-IoT (NB-IoT) in 5G by International Telecommunication Union- Radiocommunication (ITU-R) sector in [4]. Ultra low power design as reported in [26], and wireless security issues [25] are addressed with the implementation of Turbo codes for IoT applications. With these reasons, Turbo code is arguably the best candidate at present for 5G solutions.

1.1.1.1 A Brief History of Turbo Codes

The single Recursive Systematic Code (RSC) encoder based on Maximum A Posteriori (MAP) algorithm for Turbo code was first introduced by Bahl et al. [27] and then a breakthrough in the form of the iterative Turbo code was later introduced by Berrou et al. [28]. More specifically, the iterative Turbo scheme proposed by Berrou et al. features the use of parallel concatenation of two RSC encoders, and accommodates an interleaver between the two encoders. Since then, it has received much attention because of its performance in comparison with other types of codes at very low SNR. Although Turbo code was known to be powerful when it was first introduced, the implementation onto hardware was not popular. One of the main reason for this is its high complexity and lack of feasible platforms for practical implementation. Moreover, the Turbo code

algorithm was developed based on MAP with heavy computational cost and complexity due to multiple exponential computation.

However, the MAP algorithm received a renewed interest when it was realised that the complexity can be dramatically dropped without severely affecting its performance. The complexity of the MAP algorithm is reduced by transforming all calculations into the logarithm domain dubbed the Turbo Suboptimal Log-MAP algorithm [29]. In the natural logarithm domain, all exponential values are processed as though they are values in the linear domain, and multiplications turn into simple addition. However, one area where computation simplicity can be traded off with BER performance is in the approximation of an inevitable term that appears in the computation of forward metric, reverse metric, and the log-likelihood ratio (LLR) which can be quantized to give hard decision like performances. This notorious term is the log of sum of exponentials, where the exponential term appears as a result of the Gaussian nature of the additive noise. It will be shown that the log-sum of exponentials can be conveniently broken down into smaller nested units known as which only evaluates the log-sum of two (more commonly) [30] or a few exponential terms at a time [31], also known as the Jacobian logarithm. This is to be performed recursively until all the terms in the sums have been evaluated. At the other extreme, it is also possible to entirely eliminate the notorious log-sum of exponential terms at a cost of performance loss [32], [29]. In this research, the suboptimal Log-MAP solution will be the main focus since it offers a desirable performance compared to the Max-Log-MAP solution. The performance for Log-MAP is also included for comparison analysis.

1.1.2 Soft Demodulation

In reference to Figure 1-1, another area where the Jacobian logarithm occurs is at the demodulation stage. Quadrature Amplitude Modulation (QAM), is used in various wireless digital communications platform and would continue to play a key role in providing solutions for 5G standards. The order of modulation specifies the number of bits per that each symbol is carrying. Higher order QAM is preferred in most systems as it allows for more bits to be transferred for each symbol. The soft bit demodulator is an essential component that facilitates Turbo decoding where received symbols are mapped into log-likelihood-ratios (LLR) soft values where due to the Gaussian nature of the channel, also consists of the log-sum of exponential [33]. These soft values enable and enhances the function of the Turbo decoder with as opposed to the other extreme of hard decision bit demodulation without any likelihood indication a bit decision. Using these LLRs, the Turbo decoder is able to better exploit the decoded results based on soft bits with higher likelihood and with that, better ensures its decoded results through the Trellis structure [34].

With expectations in achieving 20Gbps throughput for 5G, the order of modulation scheme is expected to be 256-QAM up to 1024-QAM [35], [36]. Due to higher constellation size, the computation for the soft LLR values also incurs higher complexity when using the Maximum Likelihood (ML) Log-MAP value [35],[37]. Similarly as in the Turbo decoding, with the use of the Max Log-MAP or Min demapper, it reduces the hardware complexity but introduces performance loss [35], [38]. Hence, considerable efforts have been put in to develop optimal soft demodulators that approaches the performance of the ML Log-MAP at lower computational cost.

1.1.3 Hardware Implementation for Channel Decoding

At present, there are two main hardware platforms which are most commonly targeted for channel decoding applications i.e. the Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA). Often, in the hardware implementation aspect, the ASIC solution offers the best in terms of throughput, latency, and power consumption. However, ASIC implementation often requires longer design cycles, higher cost and often requires an implementation on configurable platforms such as FPGA for verification before mass production.

Enter Software Defined Radio (SDR), a platform that allows users to examine and verify new hardware developments of various evolving 5G standards before committing to an ASIC design for mass production [7], [39], [40]. Paradoxically, SDR is not only a practical platform for test and verification but it is quickly evolving into the platform of choice for the future of programmable wireless communication system [41], [42]. A truly programmable wireless solution allows for flexibility, instant update on hardware, and cost savings in the long run especially for deep space communication system where the hardware is often challenging and costly to access and replace. As such, FPGAs emerges as the platform of choice for SDR with powerful tools for implementation of modern wireless communication.

One of the areas focused on this thesis is in proposing a Jacobian logarithm hardware implementation on FPGA with optimisation on architecture complexity, latency, power consumption, and reliability or system performance; all of which are essential key factors of evolving technologies such as 5G. Generally, the hardware platforms and their interconnections plays a large role in the performance of the Turbo code system. FPGAs allows this architecture to be created such that it matches the algorithmic requirements

using existing high end, highly optimised, on board, DSP modules within the FPGA chip. The major themes for hardware implementation on FPGA centres around the programming environment, fixed point processing, and quality of the resulting design in terms of throughput, latency, power considerations, and circuit architecture complexity. Unfortunately, there is a lack in existing literatures with discussion on Turbo code implementation on FPGA as reviewed in [10]. A higher number of FPGA implementation are reported for LDPC and Polar decoders [10] compared to the Turbo decoder.

At the moment of writing, the leading state of the art FPGA solution provider is Xilinx with the world's first introduction of 16nm UltraSCALE+ based on Fin FET technology that leverages performance boost in performance/watt [43] which will be utilised as the main FPGA tool in this research.

1.2 Problem Statement and Research Motivation

As described in previous sections, channel decoding goals defined by IMT-2020 for 5G are ever more challenging with demands of increased throughput to approximately 100 Mbps data rates on average [41], reduced equipment complexity, ultra-low power consumption, and ultra-reliable communication systems [4]. The same aspirations are also imposed to all domains of wireless communications. Deep space communication standards governed by the CCSDS are defined in [44]. The main challenge for deep space communication is similar to mobile systems but with emphasis on technology upgrade for deep space communication's highly regulated equipment. Turbo codes remains a strong contender for channel coding candidate for 5G systems and was adopted and standardised for previous generations of mobile communications as well as deep space communications ensuring full compatibility with previous technologies.

In recent years, Software Defined Radio (SDR) based on FPGAs is emerging as the platform of choice for not only testing and evaluation of evolving methods, but also considered a likely replacement for ASIC solutions as FPGAs allows flexibility through its programmability feature, faster product development cycle, and rapid implementation. Present FPGAs are easily capable of pushing over the 500 MHz performance barrier for 5G applications [4] with increased logic densities, embedded processors and DSP modules [41]. Enhanced FPGA tools also eases design flows, place and route, timing, power, and complexity analysis to be implemented on verified, and fully characterised FPGA chips.

However, more research is needed on the implementation for Turbo decoding with SDR using the FPGA platform as reviewed in [10]. Key channel decoding and soft bit demodulation algorithms and computations implemented and evaluated on FPGA are also needed to meet the demands of future wireless communication systems. The Jacobian logarithm or the max^* operator within the Turbo decoder and used in soft bit demodulation would benefit from more research in optimised development of algorithms evaluated on FPGA implementation in order to facilitate full deployment with SDR.

In this thesis, the central focus is to develop an optimised algorithm and hardware based on FPGA for the Jacobian logarithm as applied to the suboptimal turbo decoding and soft bit demodulation in wireless communication. Aspects of computational accuracy, speed, and power consumption are a key factors determining optimised FPGA implementation. Additionally, the performance of the algorithms in practical channel conditions are also evaluated.

1.3 Research Aim and Objectives

The aim of this research is to design an optimal approximation of the Jacobian logarithm's correction function in suboptimal Turbo decoder for improved BER performance and FPGA implementation.

The research objectives in this thesis are:

1. Develop a Turbo decoding communication system model to investigate, analyse, and identify key factors affecting suboptimal Log-MAP Turbo decoder performance to develop an approach for optimised design.
2. Design optimised approximations to the correction function based on the design criteria developed.
3. Evaluate the implementation of the proposed suboptimal Turbo decoder through BER performance simulation with relevant, current solutions in the literatures. Simulations under SNR mismatch conditions are also conducted to evaluate the performance of the Turbo decoder.
4. Design the FPGA hardware implementations of the proposed approximations with a focus on low complexity, low latency, and low power consumption.
5. Evaluate the proposed Jacobian logarithm approximations as applied to M-QAM soft bit demodulation through simulation.

1.4 Scope of Work

The research scope of work can be summarised by Figure 1-2:

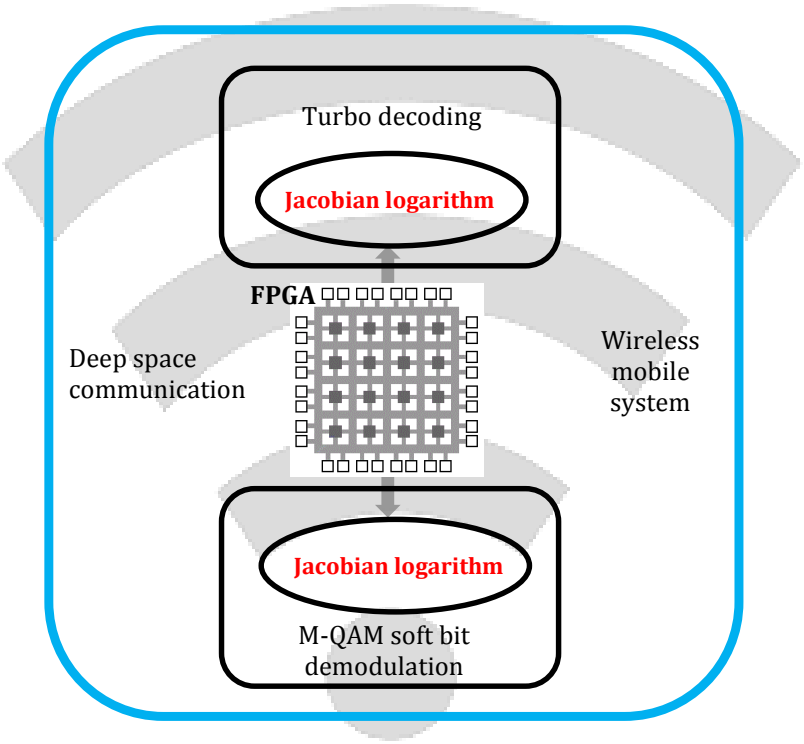


Figure 1-2 Research overview

The main theme of the thesis is to develop an alternate approach in approximating the Jacobian logarithm's correction function and propose new approximations that is optimised for FPGA implementation as applied to Turbo decoding, and M-QAM soft bit demodulation for SDR in wireless mobile and deep space communication system within the realm of wireless communication system.

This research focuses on the Jacobian logarithm as it is the most frequent and computationally complex kernel within the Turbo decoder and M-QAM soft bit demodulation of higher orders. The ability to optimise the Jacobian logarithm hardware implementation by reducing the complexity, latency, power and provide for a high computational accuracy indirectly promises lowered cost, higher throughput, and improved reliability. The FPGA is targeted as the main platform for implementation as it is the front runner of future enabling technology for SDR. At present, the application of Turbo decoding and M-QAM soft bit demodulation are found deeply engrained in wireless mobile and deep

space communication with standards for Turbo decoding established through 3G, 4G-LTE Advanced, and the Consultative Committee for Space Data System or CCSDS standard for mobile wireless and deep space communication respectively.

1.5 Thesis Contribution

Previous works in the implementation of Turbo decoding on FPGA largely focuses on the overall decoder's performance for throughput and energy efficiency with less emphasis on the key Jacobian logarithm computation where the correction function resides. Due to the iterative nature of the Turbo decoder, the Jacobian logarithm or max^* operator plays a major role in the decoder's complexity as it is the most complex computational kernel in the decoder due to the presence of the log exponential sum [30], [31], [45]–[48].

This thesis fills in the gap for FPGA implementation of the Jacobian logarithm in suboptimal Turbo decoders by identifying key parameters that influences performance. An investigation into the distribution of argument $|x|$ input to the correction function was conducted and linked to BER performance. This is followed by a study on the accuracy of suboptimal log-MAP solutions that approximates the correction function before a set of design criteria is established based on the overall computation accuracy for performance and ease of hardware implementation. Evaluations of the proposed approximation against solutions in the literature are also presented through simulation in the respective channel conditions and also under SNR mismatch. The proposed approximations were also extended to soft bit demodulation where the Jacobian logarithm also occurs.

The hardware implementation of the two new approximations were also developed for fixed point processing in FPGA. A study of complexity, timing and

power analysis of the new approximations along with relevant solutions in the literature was also conducted.

1.6 Organisation of Thesis

The organisation for the rest of the thesis along with their rationale are as follows:

Chapter 2: Literature Review

This chapter briefly introduces the principles of operation for the Turbo decoder leading to the Suboptimal Log-MAP Turbo decoder and presents a review of related works in the area of Jacobian logarithm's correction function approximation with a review on existing solution's BER performance and hardware implementation. A brief review on M-QAM soft bit demodulation along with current solutions in the literature is also included and linked to the application of Jacobian logarithm.

Chapter 3: Methodology and Design

This chapter is presented in 5 major sections. The first two section details the investigation and approach taken in proposing the new correction function approximation which allows the design criteria to be outlined in the third section. The fourth section details the hardware design for the proposed correction function approximation and finally the last section details the methodology for the proposed correction function's application in M-QAM soft bit demodulation for wireless communication.

Chapter 4: Results and Discussion

This chapter presents the simulation results for all suboptimal Log-MAP algorithm presented in the previous chapter under various channels and SNR mismatch conditions along with results of the Jacobian logarithm

implementation to soft bit demodulation. This chapter also includes performance comparisons between the algorithms and a focus on the BER performance for the proposed approximations. Further, the hardware implementation results with respect to complexity, timing, and power aspects are also presented.

Chapter 5: Conclusion and Future Work

The conclusion summarises the accomplishment and achievements with regards to each research objective with possible future work and improvements proposed.

2 LITERATURE REVIEW

2.1 Introduction

This chapter first presents the principle of operation for iterative Turbo codes and highlights the computational complexity of the Turbo MAP decoder algorithm. Subsequently, this has led to the introduction of the Log-MAP Turbo decoder, which greatly simplifies the Turbo MAP decoder by operating in the logarithm domain. However, one operation within the Log-MAP decoder known as the Jacobian logarithm continues to plague computational complexity for the Log-MAP Turbo decoder. Efforts have been made to simplify the Jacobian logarithm that gave rise to the Max-Log-MAP Turbo decoder by completely omitting the correction function, which consists of complex exponential sum computation in the Jacobian logarithm. By this omission, the Max-Log-MAP algorithm greatly simplifies the Turbo decoder but introduces a 10% performance loss as trade-off [46]. In order to mitigate the performance loss by the Max-Log-MAP Turbo decoder, research has been conducted to approximate the correction function in the Jacobian logarithm with simpler hardware implementations [30], [45], [46], [48]. This approach results in the Suboptimal Log-MAP Turbo decoder. Relevant approximation to the correction function is presented along with a review on each approximation's effectiveness in terms of BER performance and hardware complexity.

As previously highlighted in Chapter 1, there is a gap in research efforts to implement the Turbo decoder for SDR applications [10]. An overview of FPGA technology is presented in this chapter along with key modules and tools introduced for FPGA rapid prototyping to evaluate the implementation of the suboptimal solutions on FPGA.

Another area where the Jacobian logarithm occurs is the M-QAM soft bit demodulation. A brief explanation of the occurrence of the Jacobian logarithm is included along with a review relevant literatures are presented.

2.2 Turbo Codes: Principles of Operation

This section details the principle of operation of Turbo channel codes and the development of the Suboptimal Turbo decoding algorithm.

2.2.1 Turbo Encoding

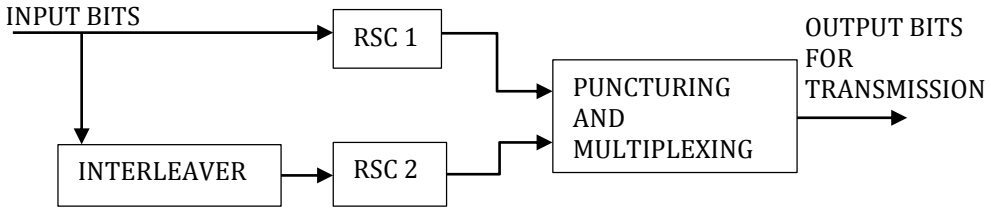


Figure 2-1 Turbo encoder structure

As illustrated in Figure 2-1, information bits are channel coded as means to protect the information being transmitted from adverse channel conditions. In Turbo codes, the information sequence is coded twice in the channel encoder using two Recursive Systematic Convolutional (RSC) encoders connected in parallel, and separated, with an interleaver that functions to provide protection from burst errors; and to provide statistical independence with increased code distance from each RSC. Often, reduced rate Turbo encoders are employed with the output parity bits from the RSCs punctured and subsequently transmitted along with the input bits or systematic bits.

2.2.2 The Recursive Systematic Turbo Encoder (RSC)

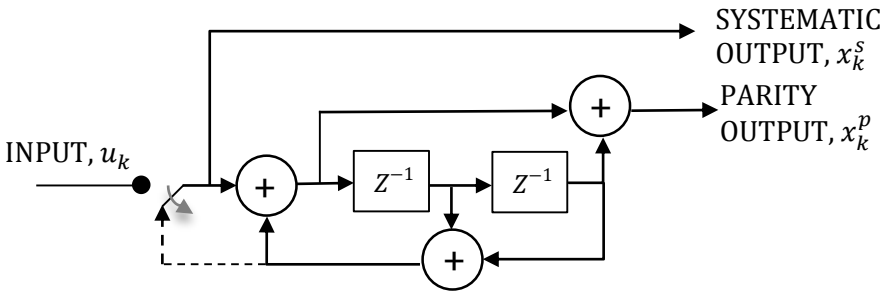


Figure 2-2 A RSC encoder

Figure 2-2 shows an example of a single two state RSC encoder with feedback/recursive path as depicted in Figure 2-1 presented by Berrou *et al*. In most Turbo encoding schemes [49]–[51], both RSC encoders shares the same structure with one input for information bits to be coded, $u_k = \{u_1, u_2, \dots, u_N\}$ as well as two outputs, consisting of the generated parity bits, $x_k^p = \{x_1^p, x_2^p, \dots, x_N^p\}$ and systematic bits which are identical to the input, denoted as $x_k^s = \{x_1^s, x_2^s, \dots, x_N^s\}$. The systematic output of the lower RSC encoder concatenated in parallel is usually not transmitted as it is identical to the upper RSC encoder, albeit in a different order due to interleaving. This gives the overall Turbo encoder rate (number of input/number of outputs) as $r = 1/3$. In this research work, the systematic bits from the upper encoder are transmitted along with the two parity outputs of both RSC encoders. However, the parity bits are not transmitted in full with alternate parity bits from both encoders punctured to achieve a higher bit rate of $r = 1/2$ [49]–[51].

The encoder is a convolutional code with constraint length $K = 3$, characterised by the number of delay registers present z^{-1} plus one, and generator polynomial $G = \{g_1, g_2\} = \{7,5\}$ which describes the feed forward and feedback circuit connection in octal form. The '+' symbol denotes logic XOR

operation. The complexity of the encoder increases with higher number of states which is also related to the constraint length.

2.2.2.1 The Trellis, Termination and Tail Bits

The Trellis diagram for the encoder in Figure 2-2 shows not only states, input/output, and state progression, but also the state transition with time index k. The nodes or circle in the Trellis represents the state of the encoder and the arrows indicate the next state depending on the incoming information bit u_k.

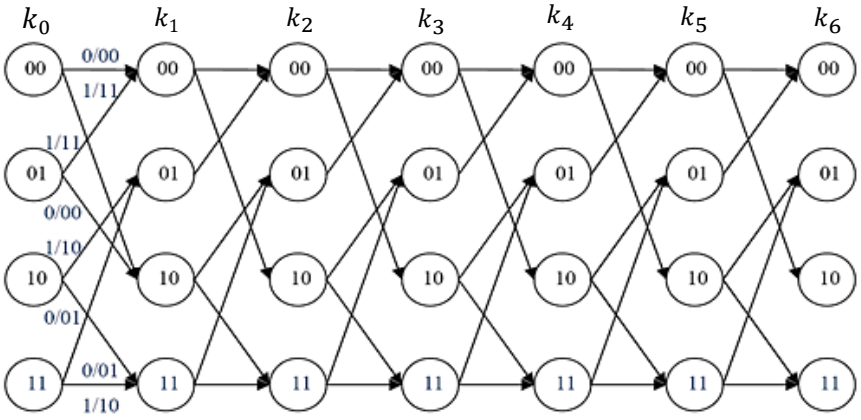


Figure 2-3 Trellis diagram of a RSC encoder with generator polynomial $G = \{7, 5\}$

Although the Trellis could begin from any valid state, in most standard Turbo encoders, the initial coding state is always set to zero, i.e. in this situation, state '00'. At the end of frame length, when all input information bits d_k have been encoded, the input is switched to receive the recursive path denoted by the dashed line connection in Figure 2-2. The feedback bits along with the XOR operations will reset the state to zero after a period of constraint length, producing tail bits from each encoder in the process. The tail bits are appended, multiplexed, punctured, and transmitted with the parity and systematic bits.

2.2.3 Turbo Encoding Standards for Wireless Communication

This Section details two other Turbo encoder structures used extensively in this research, namely, the 4G – Long-Term Evolution Advanced (4G-LTE Advanced) and Consultative Committee for Space Data Systems (CCSDS) standard. A core difference between these Turbo codes is the number of constraint length used to withstand the targeted channel that the data is being transmitted. With increase in the constraint length, the number of state representing the function of these encoders also increases, this, in turn requires higher hardware complexity at both the encoder and the decoder end. However, it is also important to note that an increase in constraint length allows for higher resilience of transmitted data fidelity against channel noise and lower number of iterations for decoding.

2.2.3.1 Eight States, 4G-LTE Advanced Turbo Decoder [12], [52]

At present, the Turbo decoding standard for 5G has yet to be defined. However, consistent with the Turbo decoding structures of past generations, the eight states Turbo decoder has been adopted as the standard Turbo decoder for mobile wireless communication and is expected to be maintained for the 5G standard. As such all simulation setup in this research adheres to the 3GPP 4G-LTE Advanced defined Turbo decoding scheme standard in[12], [52].

The 4G-LTE Advanced Turbo decoder is similarly setup as a parallel concatenated convolutional code [12], [52]. These standard targets applications in wireless voice telephony, mobile internet access, video calls, and mobile broadband [53]. The full schematic of the 4G-LTE Advanced Turbo encoder is illustrated in Figure 2-4 [12], [52]. The 4G-LTE Advanced Turbo encoding scheme features RSC encoders with constraint length $K = 4$, (8 states) and generator polynomial $G = \{13,15\}$ [12], [52].

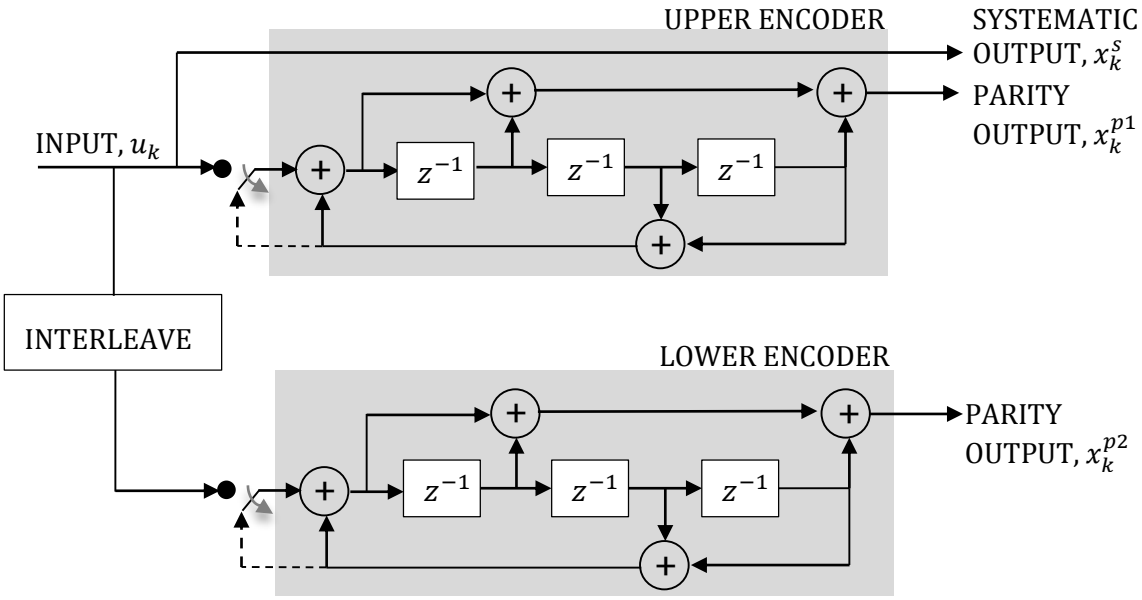


Figure 2-4 The 4G-LTE Advanced Turbo encoder schematic

Prior to encoding, both RSC encoders are set to zero state, i.e. all shift registers are set to logic '0'. The input bits, u_k are encoded by the upper encoder and interleaved before encoding by the lower decoder. The parity bits from each encoder, $x_k^{p1} = \{x_1^{p1}, x_2^{p1}, \dots, x_N^{p1}\}$ and $x_k^{p2} = \{x_1^{p2}, x_2^{p2}, \dots, x_N^{p2}\}$ from the upper and lower encoder respectively are multiplexed, punctured, and transmitted with the systematic output x_k^s. The size of the interleaver may range from as small as 40 to 5114 bits. After all input bits u_k are encoded, the encoders are brought back to all zero state by toggling of the switches connecting the feedback circuitry to the input denoted with the dotted lines. Tail bits generated by the upper and lower encoder are different due to the interleaver, and has to be dealt with separately. The overall encoding rate, $r = 1/2$.

2.2.3.2 Sixteen States, CCSDS Turbo Encoder [54]

The Turbo code for deep space communication was developed by the Jet Propulsion Laboratory (JPL) for the National Aeronautics and Space Agency (NASA)'s Deep Space Network. Turbo codes have been approved by the

Consultative Committee for Space Data Systems (CCSDS) for telemetry channel coding and included in spaceflight hardware [54],[55].

A higher number of constraint length is required for cutting edge, high fidelity telecommunication to establish a reliable virtual presence throughout an outer space mission. Deep Space Network (DSN) application is known to invest extensively on codes that could return even a slight additional coding gain to exploit the telecommunication link between probes and rovers in space missions . Since 2004, Turbo codes are implemented on the Messenger, Stereo, MRO and other missions up to present day [56], [56].

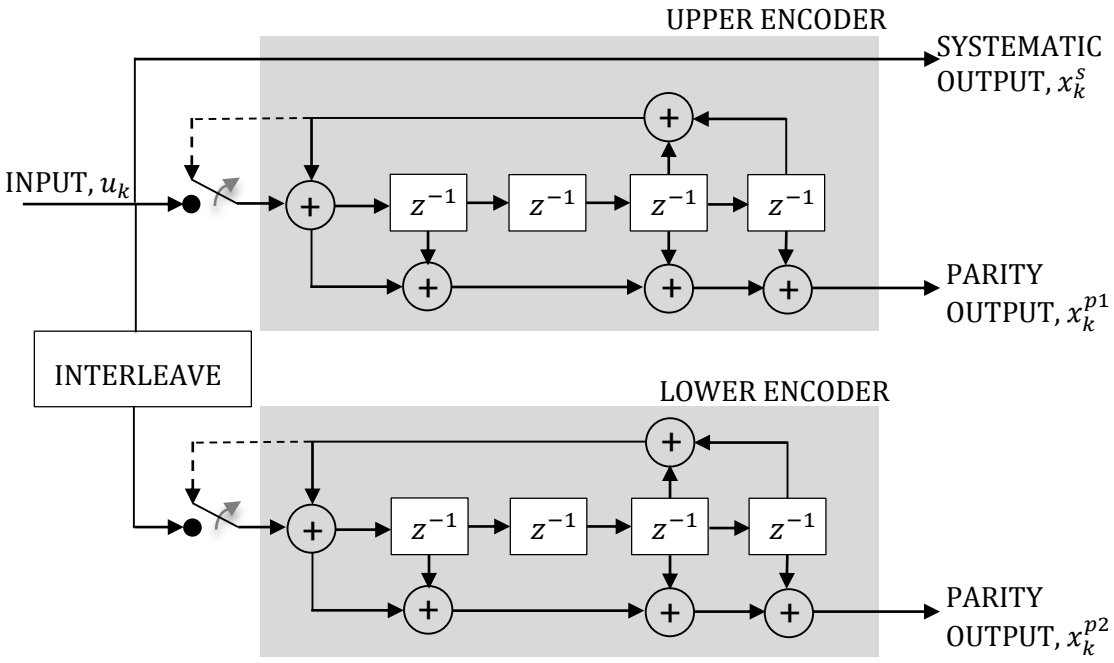


Figure 2-5 The CCSDS Turbo encoder schematic

Figure 2-5 illustrates the CCSDS Turbo encoder with constraint length $K = 5$, and generator polynomial $G = [23,33]$, [54]. The principle of operation for the CCSDS Turbo encoder is similar to that of the Turbo encoders discussed in the previous section with encoding rate $r = 1/2$. The interleaver size for CCSDS can range from 1784 to 16056 with 16 states [54], [57].

The motivation for the usage of the CCSDS standard lies in the increased constraint length for Turbo decoding which will intensify the computation frequency of the Jacobian logarithm and exacerbated with the iterative nature of Turbo codes. It is expected that changes implemented on to the Jacobian logarithm will affect Turbo decoders with higher states due to the increased frequency of its computation. Owing to this fact, investigation into the CCSDS Turbo decoding standard is necessary to fully characterise the effects of suboptimal Turbo codes for wireless standards.

2.2.4 Turbo Interleavers

The interleaver design has a profound effect on the performance of Turbo codes. The composition of type of encoder, interleaver design, and puncturing pattern determines the minimal free distance, d_{min} of Turbo codes, and subsequently, its performance.

Generally, the interleaver size is proportionate to the BER performance of the Turbo code [58]. In terms of hardware implementation, they are affordable and relatively simple to implement. However, the optimal choice of interleaver size also depends on the tolerable delay for a particular system. For a non-real-time system, a delay of 10^4 bits for example may be tolerable. In contrast, a real-time system, such as interactive speech and video transmission, the system can only tolerate a delay of approximately 100 bits. An interleaver exceeding 5000 bits size achieves its near optimum performance, but such systems only suits non-real-time applications [50]. The following sub Sections briefly reviews the different interleavers commonly used with various Turbo code systems.

2.2.4.1 Block Interleaver

Conceptually, the block interleaver consists of a rectangular matrix with R rows and C columns. Data is read into the rectangular matrix in a row by row sequence, and read out in a column wise fashion with interleaver size N [59], [60]. Despite their ease in hardware implementation, block interleavers are shunned due to large occurrence of error sequences that are self-terminating for both RSC encoders which produces low code weight sequences [58]. The de-interleaving process is simply the reverse of the interleaving process for block interleavers.

2.2.4.2 Pseudo Random Interleaver

The pseudo random interleaver maps the incoming bits in a 'randomly' generated sequence to the output as adopted in the CCSDS standard [60]. In other words, the output is a 'randomly' scrambled sequence of the input. The actual implementation of this interleaver however, features a pre-set random sequence generation seed for the 'random' scrambling operation as defined in the CCSDS standard [54]. Hence the term, pseudo random. The advantage of the pseudo random interleaver is its simple design and significantly improved BER performance over its block counterpart for medium or large sized interleaver [61]. For smaller sized interleavers ($N \leq 100$) however, the block interleaver holds an advantage in BER performance over the pseudo random interleaver. Another disadvantage of the pseudo random interleaver is its inability to guarantee the minimum spreading properties which is linked directly to the system's BER performance.

2.2.4.3 S-Random Interleaver

The S-Random (Semi-Random) interleaver is an improvement over the pseudo random interleaver where it is designed to guarantee high spreading properties and is commonly used for mobile communication standards such as 4G-LTE Advanced [12], [52]. The S-Random permutation is done by generating a set of random integers i input positions ($1 \leq i \leq N$). Each randomly selected output position i is compared to the S previously selected integer. Should the current selection be equal to any of the S previous selection within a range of $\pm S$, then the current selection is rejected. This process is repeated until all N positions are selected. The drawback of the S-Random interleaver is the increase of searching time with S. A suggested optimisation is to set $S < \sqrt{N/2}$ for a reasonable search time [62]. An alternate option in hardware implementation to reduce processing time would be to store S-Random interleaver mappings but would take up valuable memory especially when catered for a wide variety of packet lengths.

2.2.5 Puncturing and Multiplexing

A standard Turbo encoder output generally consists of the systematic output, x_k^s and parity outputs. Depending on the selected mode, there can be two, or more sets of parity bits generated by the encoder [57]. Consistent with the scope and standards used in this research, the mode for puncturing and multiplexing shall be limited to the case of raw rate $r = 1/3$ where the encoder generates the systematic bits, and two sets of parity bits (x_k^{p1} and x_k^{p2}) as illustrated in Figure 2-6 [50].

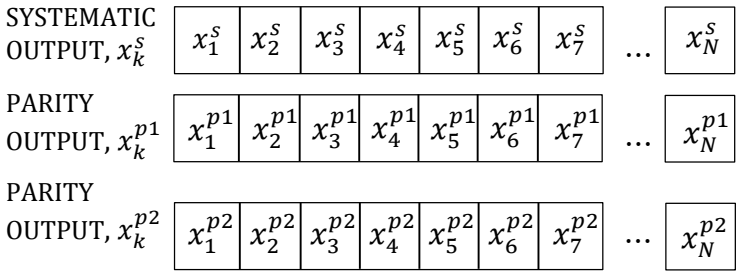


Figure 2-6 Turbo encoder outputs

Puncturing involves the removal of parity bits following a pre-defined pattern after encoding. Puncturing can improve the code rate for transmission without increasing the complexity of the system. Figure 2-7 shows the puncturing pattern for the encoder taking only odd parity bits from the upper encoder and even parity bits from the lower decoder. Punctured bits are marked with greyed regions in Figure 2-7 achieving an overall increased rate of $r = 1/2$. As a rule, the systematic bits are never punctured as it will degrade the overall performance of the system more significantly than puncturing the parity bits. The inverse process or de-puncturing is done at the receiver according to similar principle of operation.

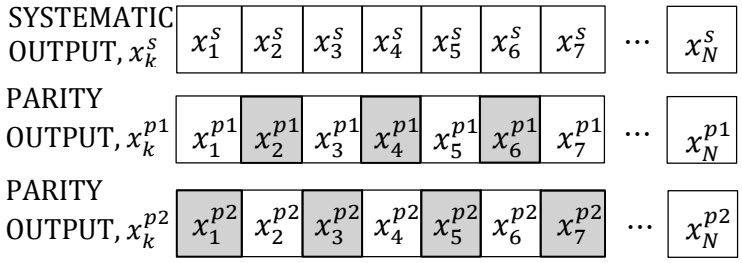


Figure 2-7 Turbo encoder parity bits after puncturing for rate $r = 1/2$

Following the puncturing of parity bits, the systematic bits are multiplexed with the remaining parity bits and transmitted over the channel. Multiplexing of the remaining bit streams for pairs each systematic bit with alternate parity bits

form the upper and lower encoder respectively as shown in Figure 2-8 before transmission over the channel.

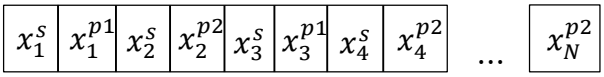


Figure 2-8 Punctured and multiplexed data for transmission

2.2.6 Turbo Decoding

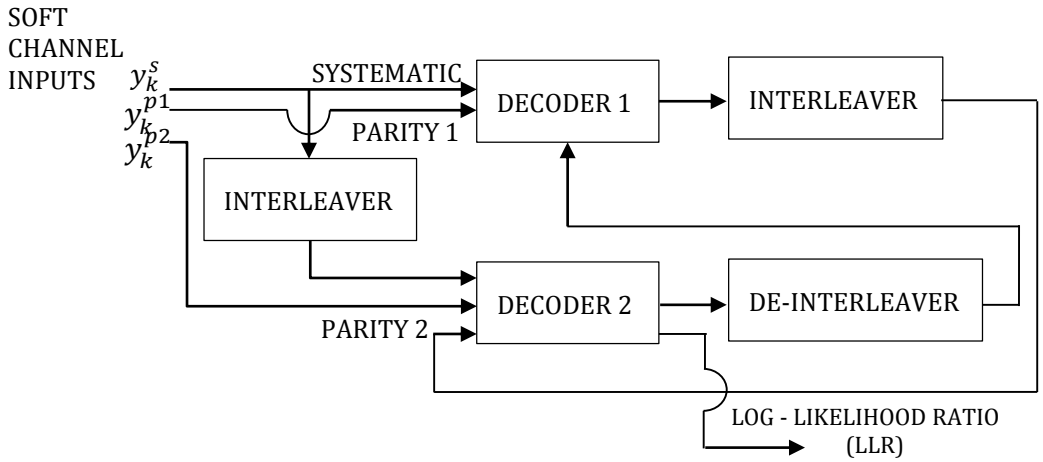


Figure 2-9 Turbo decoder structure

The transmitted signal are subjected to channel conditions and received at the upper decoder as soft channel inputs as illustrated in Figure 2-9 [49]–[51]. The received signal, are de-multiplexed into systematic bits, y_k^s and parity bits, (y_k^{p1} and y_k^{p2}). Subsequently, the Turbo decoder breaks the decoding problem into two decoders to produce locally optimal solutions to be shared with each other in an iterative manner. Each decoder produces soft outputs in the form of a-posteriori probabilities of the received bits, and the log-likelihood ratio or LLR. The lower decoder then receives the interleaved, a-posteriori information as a-priori information from the upper decoder and in turn generates its own a-posteriori information for the next decoder iteration. This manner of decoding

is continued until the pre-set number of iteration is complete and the final LLR will be taken from the lower decoder.

2.2.6.1 Log- Likelihood Ratio (LLR)

The LLR of a decoded data bit, $L(u_k)$ is defined to be the log ratio probability of the data bit taking either $+1$ or -1, denoting a '1' or '0' respectively as follows [49]–[51]:

$$L(u_k) \triangleq ln\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \quad \text{EQ. 2-1}$$

where, $L(u_k)$ is the Log likelihood ratio (LLR) and $P(u_k = \pm 1)$ the probability of data bit u_k takes on value of ± 1.

The sign, as well as magnitude of $L(u_k)$ indicates whether the information bit, u_k is more likely to be $+1$ or -1. When $L(u_k)$ is larger than 0, $P(u_k = +1) \gg P(u_k = -1)$ indicates that u_k is more likely to be $+1$. This also applies conversely for $L(u_k)$ smaller than 0, making u_k more likely to take -1. When $L(u_k) \approx 0.5$, $P(u_k = +1) \approx P(u_k = -1) \approx 0.5$, u_k is uncertain.

2.2.6.1.1 LLR Based on Conditional Probabilities

In channel coding, the probability that the data bit, $u_k = \pm 1$ based on a received signal $\underline{y}$ are known as the a-posteriori probability [49]–[51]:

$$L\left(u_k|\underline{y}\right) \triangleq ln\left(\frac{P(u_k = +1|\underline{y})}{P(u_k = -1|\underline{y})}\right) \quad \text{EQ. 2-2}$$

where, $\underline{y}$=received signal.

Apart from the conditional LLR, $L(u_k|\underline{y})$, another conditional LLR based on the probability that the receiver's matched filter output would be y_k given that the transmitted bit x_k was either $+1$ or -1 is defined as follows [49]–[51]:

$$L(y_k|x_k) \triangleq ln\left(\frac{P(y_k|x_k=+1)}{P(y_k|x_k=-1)}\right) \quad \text{EQ. 2-3}$$

where, x_k =channel coded bits for transmission.

Assume that the transmitted bit $x_k = \pm 1$ has been sent over the Gaussian or fading channel with BPSK modulation, the probability of the matched filter output y_k can be written as [49]–[51]:

$$L(y_k|x_k=+1) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{E_b}{2\sigma^2}(y_k-a)^2\right) \quad \text{EQ. 2-4}$$

where, E_b is the transmitted energy per bit; σ^2 the noise variance; and a the fading amplitude ($a = 1$ for non-fading AWGN channels).

Similarly, for the case of $x_k = -1$, [49]–[51]:

$$L(y_k|x_k=-1) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{E_b}{2\sigma^2}(y_k+a)^2\right) \quad \text{EQ. 2-5}$$

Substituting EQ. 2-4 and EQ. 2-5 into EQ. 2-3 gives [49]–[51],

$$\begin{aligned} L(y_k|x_k) &= ln\left(\frac{exp\left(-\frac{E_b}{2\sigma^2}(y_k-a)^2\right)}{exp\left(-\frac{E_b}{2\sigma^2}(y_k+a)^2\right)}\right) \\ &= L_c y_k \end{aligned} \quad \text{EQ. 2-6}$$

where $L_c = 4a\frac{E_b}{2\sigma^2}$ also known as the channel reliability value which depends on the SNR and fading amplitude of the channel. In the case of BPSK over a Gaussian channel, the conditional LLR $L(y_k|x_k)$ is also the soft output of the channel, y_k multiplied with the channel reliability value, L_c.

2.2.7 The MAP Algorithm

The Maximum-A-Posteriori or MAP algorithm is implemented within each Turbo decoder to compute soft input and output for iterative decoding. First introduced by Bahl et al. or better known as the BCJR algorithm for estimating

the a-posteriori probabilities of states and later modified by Berrou et al. for iterative Turbo decoding. The MAP algorithm gives the probability that a decoded bit, u_k takes the value of $+1$ or -1, given the received symbol sequence $\underline{y}$ i.e. $L(u_k|\underline{y})$ [49]–[51]

$$L(u_k|\underline{y}) = ln\left(\frac{P(u_k = +1|\underline{y})}{P(u_k = -1|\underline{y})}\right) \quad \text{EQ. 2-7}$$

where, $L(u_k|\underline{y})$ is the likelihood ratio of data bit, and $u_k = +1$ or -1 given received symbols, $\underline{y}$.

2.2.7.1 Trellis Decoding Algorithm

Consider the Trellis diagram of the decoder described in Figure 2-3 with constraint length $K = 3$, and four possible states with S_{k-1} and S_k denoting the previous and next state respectively [49]–[51].

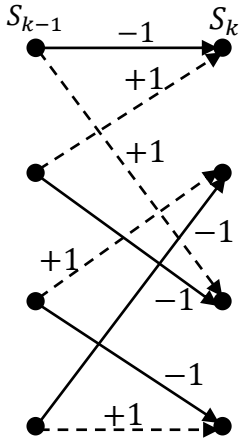


Figure 2-10 Trellis transition with constraint length $K = 3$

Each state has only two possible transition with input values of either $+1$ or -1 for a binary code. The solid lines represents the -1 input transition, and the dotted lines for $+1$. The transition from the previous state, S_{k-1} to the present state, S_k will give the value of input bit u_k. The probability that $u_k = +1$ is represented by one of the four dotted lines. These transitions are mutually

exclusive and only one transition is possible at any time. Hence, the occurrence probability of any of these transitions is equivalent to the sum of their individual probabilities. EQ. 2-7 can be rewritten as

$$L(u_k|\underline{y}) = ln\left(\frac{\sum_{(s',s)\Rightarrow u_k=+1} P(S_{k-1}=s', S_k=s, \underline{y})}{\sum_{(s',s)\Rightarrow u_k-1} P(S_{k-1}=s', S_k=s, \underline{y})}\right) \quad \text{EQ. 2-8}$$

where, s' is the previous state; s the next state; $(s',s) \Rightarrow u_k = +1$ denotes trellis transitions for $u_k = +1$; $(s',s) \Rightarrow u_k = -1$ denotes trellis transitions for $u_k = -1$; and $P\left(S_{k-1}=s', S_k=s, \underline{y}\right)$ is the probability of transition between previous and next state given $\underline{y}$.

For simplicity, $P\left(S_{k-1}=s', S_k=s, \underline{y}\right)$ is written as $P(s',s,\underline{y})$ for future discussions. The received sequence $\underline{y}$ can be divided into three parts; the present, $\underline{y}_k$; the past or received sequence prior to the present transition, $\underline{y}_{j<k}$; and the future, for after the current transition, $\underline{y}_{j>k}$. Substituting this into the individual $P(s',s,\underline{y})$ yields [49]–[51]:

$$P\left(s',s,\underline{y}\right) = P(s',s,\underline{y}_{j<k},\underline{y}_k,\underline{y}_{j>k}) \quad \text{EQ. 2-9}$$

where, $\underline{y}_k$ is the present received signal, $\underline{y}_{j<k}$ the past received sequence, and $\underline{y}_{j>k}$ the future received sequence.

The probability that the transition from state $S_{k-1}=s'$ to state $S_k=s$ given the received sequence $\underline{y}$, can be divided to the product of forward, branch, and backward state metric or $\alpha_{k-1}(s')$, $\gamma_k(s',s)$, and $\beta_k(s)$ respectively. Assuming a memoryless channel with future sequence, $\underline{y}_{j>k}$ only depending on the present state, and using Bayes rule repeatedly to EQ. 2-9 gives [49]–[51]:

$$P\left(s',s,\underline{y}\right) = \beta_k(s)\cdot\gamma_k(s',s)\cdot\alpha_{k-1}(s') \quad \text{EQ. 2-10}$$

where, $\beta_k(s)$ is the backward state metric, $\gamma_k(s', s)$ the branch state metric, and $\alpha_{k-1}(s')$ the forward state metric.

Concretely, $\alpha_{k-1}(s')$ denotes the probability that the Trellis is in state s' at time $k-1$ and that the received channel sequence is up to $\underline{y}_{j<k}$; $\beta_k(s)$ denotes the probability that the Trellis is in state s at time k with the future received channel sequence will be $\underline{y}_{j>k}$; and, $\gamma_k(s', s)$ denotes the probability that the Trellis was in state s' at time $k-1$, it moves to state s and the received channel sequence for this transition is $\underline{y}_k$. The significance of these metrics and position in the Trellis are illustrated in Figure 2-11 [49]–[51].

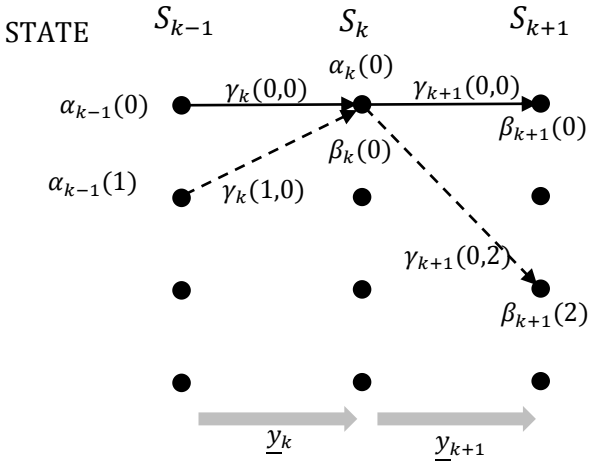


Figure 2-11 Trellis transition and computation for state metrics

The MAP algorithm computes the state metrics $\alpha_k(s)$ and $\beta_k(s)$ for all states, s throughout all time index $k = 0, 1 \cdots N-1$, with $\gamma_k(s', s)$ for all transitions branch with the following recursive calculation [49]–[51]. These values are then used to find the probabilities $P(S_{k-1} = s', S_k = s, \underline{y})$ to give the LLR, $L(u_k|\underline{y})$ for each decoded bit, u_k.

2.2.7.1.1 Forward State Metric

The computation for forward state metric, $\alpha_k(s)$ is more specifically computed as [49]–[51]:

$$\alpha_k(s) = \sum_{all\ s'} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \qquad \text{EQ. 2-11}$$

where, $\gamma_k(s', s)$ is the branch metric from previous state to present state and $\alpha_{k-1}(s')$ represents the previous forward state metric associated with the state transition.

All forward state metric, $\alpha_k(s)$ values are calculated recursively with initial conditions that the Trellis begins from state 0, with $\alpha_0(S_0 = 0) = 1$, and $\alpha_0(S_0 = s) = 0$ for all $s \neq 0$. This is illustrated in Figure 2-11 for a binary trellis with only two previous states.

2.2.7.1.2 Backward State Metric

Similarly, the backward state metrics $\beta_{k-1}(s')$, recursive computation can be calculated using EQ. 2-12 [49]–[51]:

$$\beta_{k-1}(s') = \sum_{all\ s} \beta_k(s) \cdot \gamma_k(s', s) \qquad \text{EQ. 2-12}$$

where, $\beta_k(s)$ is the present state's backward state metric associated with the state transition.

Similarly, with all $\gamma_k(s', s)$ known, a backward recursion can be used to calculate all $\beta_{k-1}(s')$ from $\beta_k(s)$ as illustrated in Figure 2-11. The initial conditions for $\beta_N(0) = 1$ and $\beta_N(s) = 0$ for all $s \neq 0$ in a Trellis terminated to the all zero state.

2.2.7.1.3 Branch Metric

The branch metrics, $\gamma_k(s', s)$ can be calculated from the received channel sequence [49]–[51].

$$\gamma_k(s', s) = P(\underline{y}_k|\underline{x}_k) \cdot P(u_k) \quad \text{EQ. 2-13}$$

where, $P(\underline{y}_k|\underline{x}_k)$ is the probability of received symbols given transmitted code word $\underline{x}_k$, and $P(u_k)$ the a-priori probability of bit u_k.

Assuming a Gaussian, memoryless channel using BPSK and updating EQ. 2-13 gives:

$$\gamma_k(s', s) = \prod_{l=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{E_b}{2\sigma^2}(y_{kl} - ax_{kl})^2\right) \cdot \left(\frac{e^{-\frac{L(u_k)}{2}}}{1 + e^{-L(u_k)}}\right) e^{(u_k L(u_k)/2)} \quad \text{EQ. 2-14}$$

where, n is the number of bits in $\underline{y}_k$ and $\underline{x}_k$; x_{kl} the individual bits transmitted; y_{kl} the individual received bits; and $L(u_k)$ represents the log likelihood ratio (LLR) of bit u_k.

2.2.7.1.4 Log Likelihood Ratio (LLR)

The conditional MAP, LLR of u_k, given the received sequence $\underline{y}_k$ can be written as [49]–[51]:

$$L\left(u_k\middle|\underline{y}\right) = ln\left(\frac{\sum_{(s',s)\Rightarrow u_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)}{\sum_{(s',s)\Rightarrow u_k-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)}\right) \quad \text{EQ. 2-15}$$

The MAP algorithm as described in this Section is extremely complex owing to the multiplications, logarithms, and exponential functions for the recursive computation of $\alpha_k(s', s)$ and $\beta_k(s', s)$.

2.2.7.2 Iterative MAP Turbo Decoding

In summary, the final am of the Turbo decoder is to compute the a posteriori LLR that can be divided to three terms i.e. the a priori information $L(u_k)$, $L_c y_{ks}$,

and extrinsic information $L_e(u_k)$ all derived from equations presented thus far in MAP decoding. The a-priori LLR term, $L(u_k)$ is also known as the intrinsic information which is received before the decoding process begins from the counterpart decoder. At the first iteration however, all of the a-priori values are set to 0 which corresponds to $P(u_k) = 0.5$, [49]–[51]. The $L_c y_{ks}$ term is the soft output of the channel representing the systematic bit, u_k which was transmitted, and received as y_{ks}. When the channel SNR is high, the channel reliability value L_c will be high and provide a large influence on the a-posteriori LLR, $L(u_k, \underline{y})$. This applies conversely for when the SNR is low. The final term, $L_e(u_k)$ is also called the extrinsic information that will be interleaved and sent to the other decoder. This process is repeated until the total number of pre-set iteration is completed and the a posteriori LLR can be obtained.

2.2.8 The Max-Log-MAP Algorithm

The MAP algorithm was challenging to implement when it was first introduced due to the complexity required on hardware with exponential sum computations, multiplications, and logarithmic operation to compute the eventual a posteriori LLR. However, the MAP algorithm received a renewed interest when it was modified to operate in the logarithm domain thanks to the Jacobian logarithm approximation first introduced by Erfanian *et al*. In the log domain, exponential terms disappear, and multiplication operations reduces to additions. The Max-Log-MAP algorithm significantly reduces the complexity of the MAP algorithm by transferring all state and branch metric computation to the logarithm domain, and applying the max approximation of the Jacobian algorithm as follows [31], [45]:

$$ln\left(\sum_i e^{x_i}\right) \approx \max_i(x_i) \qquad \text{EQ. 2-16}$$

where $\max_i(x_i)$ means the maximum value of x_i. The following subsections details the application of this approach to the metric computations.

2.2.8.1 Max-Log-MAP: Forward State Metric

By transforming the forward state metric computation in EQ. 2-11 to the log domain and adopting EQ. 2-16, the forward state metric is updated as follows [49]–[51]:

$$\alpha_k(s) = ln\left(\sum_{all\ s'} \alpha_{k-1}(s') \cdot \gamma_k(s', s)\right)$$

$$\alpha_k(s) \approx \max_{s'}(A_{k-1}(s') + \Gamma_k(s', s)) \qquad \text{EQ. 2-17}$$

where, $A_{k-1}(s') = \ln(\alpha_{k-1}(s'))$; $\Gamma_k(s', s) = \ln(\gamma_k(s', s))$; and $\max_{s'}(\,)$ is the maximum value of all $A_{k-1}(s') + \Gamma_k(s', s)$ leading to a state.

The impact of this approach can be explained with the Trellis diagram. In reference to Figure 2-11, for a binary Trellis, there are only two paths leading into the present stage, $S_k(s) = s$ from the previous stage, $S_{k-1}(s') = s'$. The branch metric, $\Gamma_k(s', s)$ is added to the previous state $A_{k-1}(s)$ for the $A_k(s)$ in that particular path. Based on the approximation in EQ. 2-17, the next $\alpha_k(s)$ value will be the maximum of all $A_k(s)$ paths leading to the state. This selection process can be thought of as selecting a surviving path and discarding other paths leading to the state. As a result, the Max-Log-MAP gives the probability of the most likely path through the Trellis rather than considering all paths throughout the Trellis to a particular state which causes performance degradation. On the other hand, the recursive forward state metric calculation has been reduced to simple addition and comparison operations to determine the maximum value.

2.2.8.2 Max-Log-MAP: Backward State Metric

Similarly, for the backward state metric, the equations can be rewritten in the log domain as [49]–[51]:

$$\beta_{k-1}(s) = ln\left(\sum_{all\ s} \beta_k(s) \cdot \gamma_k(s', s)\right)$$

$$\beta_{k-1}(s) \approx \max_{s}(B_k(s) + \Gamma_k(s', s)) \qquad \text{EQ. 2-18}$$

where, $B_{k-1}(s') = ln\big(\beta_{k-1}(s')\big)$; and $\max_{s'}()$ is the maximum value of all $B_{k-1}(s') + \Gamma_k(s', s)$ leading to a state.

The backward recursion calculation of $\beta_{k-1}(s)$ is equivalent to the maximum of all $\Gamma_k(s', s) + \mathrm{B}_k(s)$ paths to the state.

2.2.8.3 Max-Log-MAP: Branch Metric

Finally, the branch metric computation as described by EQ. 2-14 in the log domain can be re-cast, simplified, and rewritten as [49]–[51]:

$$\gamma_{\mathrm{k}}(\mathrm{s}', \mathrm{s}) = ln\left(e^{u_k L(u_k)/2} \cdot exp\left(\frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl}\right)\right) \qquad \text{EQ. 2-19}$$

$$\gamma_k(s', s) = \frac{1}{2}u_k L(u_k) + \frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl} \qquad \text{EQ. 2-20}$$

2.2.8.4 Iterative Max-Log-MAP Turbo Decoding

Finally for the a-posteriori LLRs, $L(u_k|\underline{y})$ for the Max-Log-MAP algorithm can be updated as [49]–[51]:

$$L\left(u_k\middle|\underline{y}\right) = ln\left(\frac{\sum_{(s',s)\Rightarrow u_k=+1} exp\big(A_{k-1}(s') + B_k(s) + \Gamma_k(s', s)\big)}{\sum_{(s',s)\Rightarrow u_k-1} exp\big(A_{k-1}(s') + B_k(s) + \Gamma_k(s', s)\big)}\right) \qquad \text{EQ. 2-21}$$

$$L\left(u_k\middle|\underline{y}\right) \approx \max_{(s',s)\Rightarrow u_k=+1}[A_{k-1}(s') + B_k(s) + \Gamma_k(s',s)] - \max_{(s',s)\Rightarrow u_k=-1}[A_{k-1}(s') + B_k(s) + \Gamma_k(s',s)] \quad \text{EQ. 2-22}$$

In the Max-Log-MAP algorithm for each bit u_k, the a-posteriori LLR, $L(u_k|\underline{y})$ is calculated as the maximum value of every state and branch metric in the transition from S_{k-1} to S_k. The transitions are isolated in the numerator for all transitions corresponding to $u_k = +1$ and for u_k=-1 in the denominator. A comparison then ensues, selecting the maximum value of $A_{k-1}(s') + \mathrm{B}_k(s) + \Gamma_k(s',s)$ within the present state. For a binary Trellis, there will be $2 \cdot 2^{K-1}$ transitions at each stage, where K is the constraint length of the encoder. Although greatly simplified, the Max-Log-MAP algorithm introduces an overall 10% drop in performance compared to the MAP algorithm [46].

2.2.9 The Log-MAP Algorithm

Robertson et al. first introduced the Log-MAP algorithm in that gives identical performance to the MAP algorithm. This is facilitated with the replacement of the approximation used in EQ. 2-16, with the Jacobian logarithm. Considering only two variables involved, the log sum of exponential can be conveniently expressed as the max^* operation:

$$\ln(\mathrm{e}^{\mathrm{x}_1} + \mathrm{e}^{\mathrm{x}_2}) = max^*(x_1, x_2) \quad \text{EQ. 2-23}$$

$$max^*(x_1, x_2) = max(x_1, x_2) + ln(1 + e^{|x_1 - x_2|}) \quad \text{EQ. 2-24}$$

$$max^*(x_1, x_2) = max(x_1, x_2) + f_c(|x_1 - x_2|). \quad \text{EQ. 2-25}$$

where $f_c(x)$ or $f_c(|x|)$ is known as the correction function term with argument $x = |x_1 - x_2|$. The argument x or $|x|$ is used interchangeably in this thesis where it the input to the correction function should always be considered as $|x|$. In the case where more than two terms of log sum of exponentials are involved,

EQ. 2-23 can be nested to accommodate the exact max^* operation for a more elaborate log sum of exponential terms [31].

$$f(x_1, x_2, x_3 \cdots, x_n) = max^*(x_n, max^*(x_{n-1} \cdots max^*(x_3, max^*(x_2, x_1)) \cdots)) \quad \text{EQ. 2-26}$$

This allows for a higher order Trellis decoding with the Max-Log-MAP or Log-MAP algorithm. Although greatly simplified in the log domain, it is observed that the max^* operation with an exponential sum function is retained as described in EQ. 2-24 within the correction function and has to be computed in all states or nodes of the forward and backward state metric. As such, it constitutes a large role in hardware complexity. With the Max-Log-MAP approximation in EQ. 2-16, the correction function in the Jacobian logarithm is completely ignored, simplifying the log sum of exponentials to maximum operations using comparators as a trade-off for overall performance drop. However, retaining the correction function would require a higher hardware complexity due to the exponential and log function in $f_c(x)$. Retaining the correction function however, will give the exact performance to the ideal MAP algorithm. The solution to this problem will be expounded in the following chapter on Suboptimal Log-MAP Turbo decoders.

2.2.10 The Suboptimal Log-MAP Turbo Decoder

In the Log-MAP domain, exponential functions recede to linear equations and multiplications translates to simple addition. However, the initial addition operation in the MAP algorithm throughout the Trellis for forward and backward state metric computation remains as addition in the inverse logarithm domain and is indicated with $\boxplus$ in the computation of state metrics. To illustrate this, the natural log of the forward state metric, $\alpha_k(0)$ computation in the MAP algorithm in reference to Figure 2-11 is given as:

$$\alpha_k(0) = \alpha_{k-1}(0)\gamma_k(0,0) + \alpha_{k-1}(1)\gamma_k(1,0) \quad \text{EQ. 2-27}$$

When computed in the log domain, the following applies [49]–[51]:

$$ln[\alpha_k(0)] = \underbrace{(ln[\alpha_{k-1}(0)] + ln[\gamma_k(0,0)])}_{A} \boxplus \underbrace{(ln[\alpha_{k-1}(1)] + ln[\gamma_k(1,0)])}_{B} \quad \text{EQ. 2-28}$$

As observed in EQ. 2-28, the multiplication operations in EQ. 2-27 has been reduced to addition in the log domain as illustrated in A and B. However, as explained, inverse logarithm for addition operation in EQ. 2-27 remains, and is denoted as $\boxplus$. Therefore, in order to solve EQ. 2-28, the following is required;

$$ln[\alpha_k(0)] = ln(e^A + e^B) \quad \text{EQ. 2-29}$$

which is the crux of the problem as detailed in suboptimal Log-MAP methods. EQ. 2-29 can be re cast with the Jacobian logarithm or max^* as detailed in EQ. 2-23 to EQ. 2-26. The exact max^* is complex to implement in hardware and the Max Log-MAP algorithm implements a simplified version of the max^* by omitting the correction function, $f_c(|x|)$ entirely. This eliminates exponential functions for hardware implementation at a cost of approximately 10% BER performance degradation [46] that can be improved with the Suboptimal Log-MAP algorithm which will be presented in the following sections.

Exponential functions, along with trigonometric and logarithmic functions, are categorised as unary functions due to their non-linear properties which are challenging for implementation on hardware.

Traditionally, the correction function is implemented using the look up table (LUT) by Robertson *et al.* or LUT in FPGA with 2^b addresses, where b denotes the address bit width. LUTs offers a simple solution on hardware but sacrifices accuracy (highly dependent of the number of addresses of the LUT for more accurate representation of the correction function and radix precision), introduces high latency, and takes up memory sizes. In order to reduce memory

size and latency, a simple solution is to partition the memories into two smaller LUT [48]. However, with recent advancement in DSP modules within the FPGA, a more optimal solution is possible. A brief discussion of the hardware resources involved for each algorithm is discussed in the following sub sections before outlining and prioritising the design considerations for a highly optimised max^* FPGA implementation.

In each of the following sections, each Suboptimal Log-MAP will be introduced and discussed. The Suboptimal Log-MAP algorithm attempts to mitigate the adverse effects of the Max Log-MAP by approximating the correction function, f_c.

2.2.10.1 Constant Log-MAP (CLM)

The Constant Log-MAP (CLM) first introduced by Gross *et al.* in [63] and implemented in recent works [30], [45], [64], [65] offers a simple solution by approximating the correction function with a two value look up table. In other words, $f_c(x)$ is approximated with a step function:

$$f_{CLM}(x) = \begin{cases} C, & x \leq T \\ 0, & x > T \end{cases} . \qquad \text{EQ. 2-30}$$

Two variations of the CLM algorithm are proposed by Gross *et al.* with $CLM_{2,0.375}$ $\{T = 2, C = 3/8\}$ and $CLM_{1.5,0.5}$ $\{T = 1.5, C = 1/2\}$. The CLM plots against the exact correction function, f_c are illustrated in Figure 2-12 [45], [64], [65].

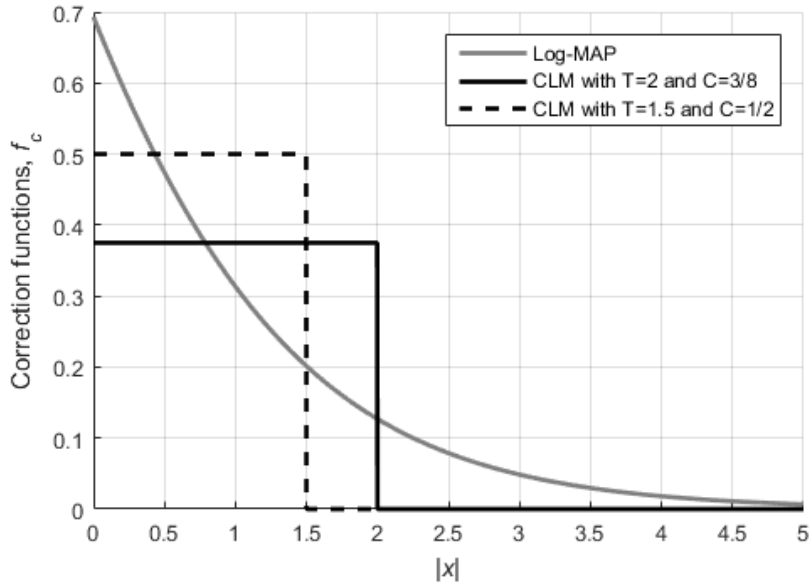


Figure 2-12 The Constant Log-MAP plot against the exact correction function

Although two variations are proposed, it has been shown that the $CLM_{1.5,0.5}$ $\{T = 1.5, C = 1/2\}$ approximation is more popular due to its overall resilience with various block sizes to channel impairments compared to other variations as shown by Classon *et al*. in [66] and in recent works reported in [45]. Based on this, and in reference to recent works implementing the CLM, the $CLM_{1.5,0.5}$ $\{T = 1.5, C = 1/2\}$ is adopted for CLM approximation throughout this research.

Complexity Analysis

The CLM crudely approximates the correction function with low accuracy that degrades BER performance as reported in [30], [64], [65]. However, the CLM is also the simplest approximation to implement next to the 8 address LUT by Robertson *et al.* [45]. The only implementation cost to hardware incurred by the CLM is a comparator, multiplexer, and a two address LUT table.

2.2.10.2 Linear Log-MAP (LLM)

First introduced in [67] and later improvised in [48], the Linear Log-MAP (LLM), also implemented in recent works [45], [64], [65] approximates the correction function with a single linear function and a constant value of 0 for larger values of $|x|$ as follows:

$$f_{LLM}(x) = max\left(0, ln\,2 - \frac{1}{2}x\right). \qquad \text{EQ. 2-31}$$

The LLM is based on the observation that the Log-MAP correction function is most effective for smaller arguments of $|x|$ around zero. Therefore, the Maclaurin series expansion was employed to approximate $f_c(x)$; and by neglecting the expansion series for orders greater than one, the approximation given in EQ. 2-31 is reduced to a linear computation which is hardware friendly. The LLM, f_{LLM} is illustrated in Figure 2-13 [45], [48], [64], [65]:

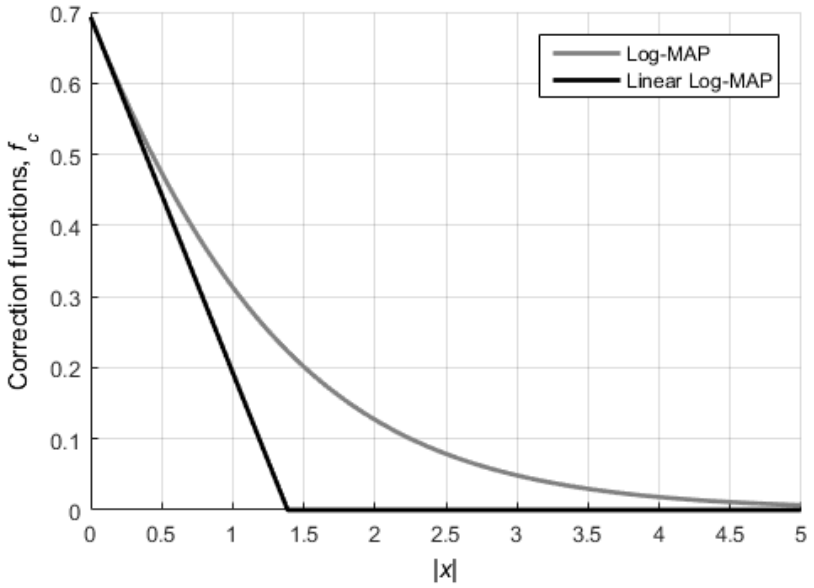


Figure 2-13 The Linear Log-MAP plot against the exact correction function

Complexity Analysis

The hardware implementation of the LLM is more complex than the CLM requiring a comparator, multiplexer, multiplier, adders, and LUT for

implementation [48]. However, it also offers improved BER performance by approximating the correction function more effectively at lower regions of $|x|$ where it is more effective. Hence giving a more accurate approximation to the exact correction function.

2.2.10.3 Multistep Log-MAP (MLM)

The Multistep Log-MAP (MLM) was introduced by Wang *et al.* [30] and also implemented in recent works [45], [64], [65] at an attempt to provide a better fit to the Log-MAP algorithm and offer reduced hardware complexity. The approximation is given as:

$$f_{MLM}(x) = \frac{ln\,2}{2^{\lfloor x+0.5 \rfloor}} \qquad \text{EQ. 2-32}$$

where $\lfloor x \rfloor$ denotes the largest integer that is smaller or equal to x. The approximation is generally more accurate than both CLM and LLM over all values of $|x|$ as shown in Figure 2-14 .

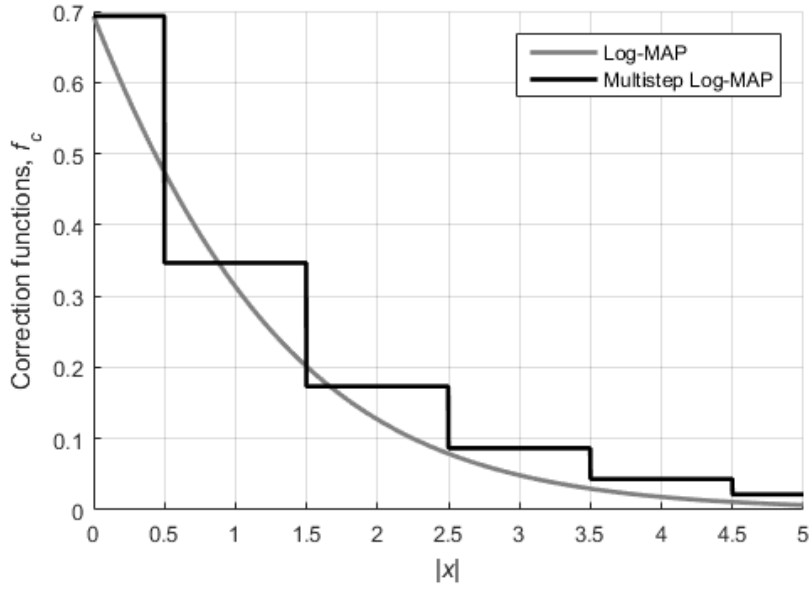


Figure 2-14 The Multistep Log-MAP plot against the exact correction function

Complexity Analysis

The hardware implementation aspect was discussed in [30] employing the use of barrel shift registers to facilitate division by 2 resulting in the multi-step function as shown. Aside from that, only an adder and a LUT to store the constant of $\ln 2$ is required. However, with the use of shift registers, each division by 2 has to be facilitated by a clock edge requiring multiple system clock cycles to compute the values for larger $|x|$ in the correction function. This complicates hardware implementation as the barrel shift register is limited to the system's maximum clock frequency and cause delays for computation of larger $|x|$ values unless a faster clock is implemented for the shift register that meets the maximum system clock frequency timing requirements.

2.2.10.4 Linear-Constant Log-MAP (LC)

First introduced in [47], the Linear Constant (LC) algorithm approximates the correction function with a linear and step function as described in EQ. 2-33.

$$f_{Lc}(x) = \begin{cases} -0.24x + 0.596, & x \leq 2.45 \\ 0.048, & 2.45 < x \leq 3.5 \\ 0, & x > 3.5 \end{cases} \qquad \text{EQ. 2-33}$$

The LC approximation features a linear function for small regions of x, followed by a constant of 0.048 for $2.45 < x \leq 3.5$ and 0 for values of x larger than 3.5. The LC approximation is illustrated in Figure 2-15 [47]:

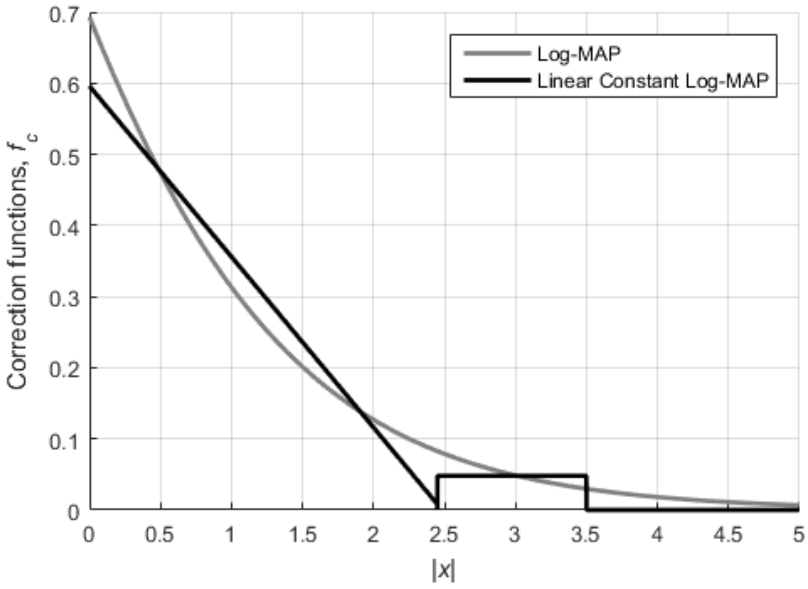


Figure 2-15 The Linear Constant Log-MAP plot against the exact correction function

The BER performance of the LC method extracted from [47] with block length of 3000, and 3 iterations using the UMTS Turbo code is referred to in Figure 2-16.

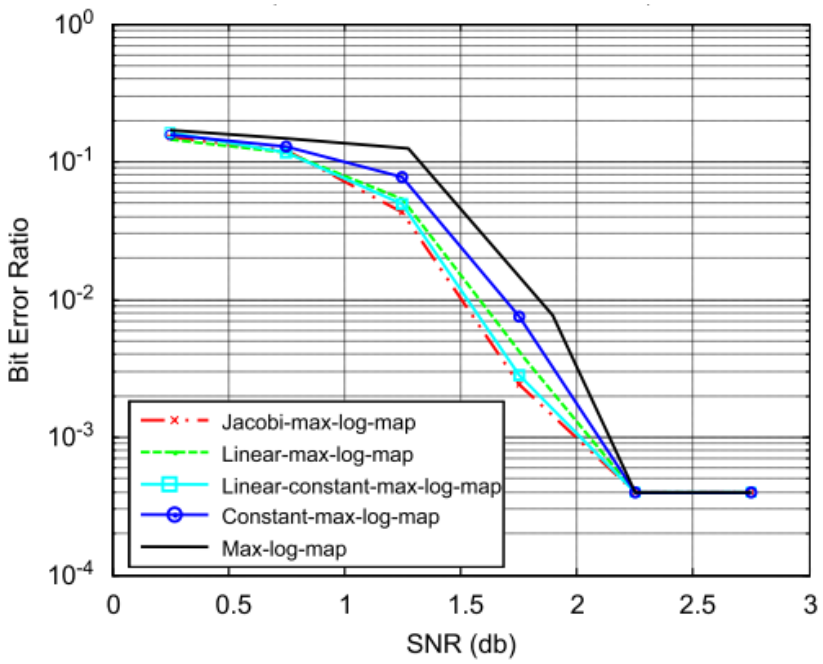


Figure 2-16 BER performance for LC with block length 3000 and 3 iterations

Complexity and Performance Analysis

The hardware implementation aspect was discussed based on implementation on Texas Instrument TMS320C6713 DSP processor with no logic component breakdown for the correction function implementation. However, from the perspective of implementation on the FPGA, the correction function can be implemented with use of LUT, multiplexer, multiplier, adder, and comparators.

The performance of the LC approximation is very close to the exact Log-MAP in lower regions of $|x|$ and continues to adhere closely to the exact Log-MAP after 3 iterations. It is also reported that the LC approximation BER performance is an improvement over the CLM and an older version of the linear Log-MAP proposed by [67].

2.2.10.5 Piecewise Linear (PL)

A recent linear approximation that is proposed in [45] breaks the correction function into 5 sections for linear approximation. The approximation is described in Table 2-1 and illustrated as follows [45]:

Table 2-1 Piecewise Linear (PL) approximation and R-Squared measure for each section

PL Approximations	$\|x\|$ Sub ranges	a_n	b_n	R-Squared
$f_{PL0}(\|x\|) = a_0x + b_o$	0:1	-0.3792	0.6754	0.996
$f_{PL1}(\|x\|) = a_1x + b_1$	1:1.5	-0.2229	0.5327	0.999
$f_{PL2}(\|x\|) = a_2x + b_2$	1.5:2	-0.1483	0.4213	0.998
$f_{PL3}(\|x\|) = a_3x + b_3$	2:3	-0.0773	0.2758	0.991
$f_{PL4}(\|x\|) = a_4x + b_4$	3:4	-0.0300	0.1362	0.993

In [45] where the PL approximation is introduced, the R-Squared goodness of fit values are also presented for each sub ranges of $|x|$. It is noted that for each of the sub range the R-Squared measure for PL is consistently good over 0.99.

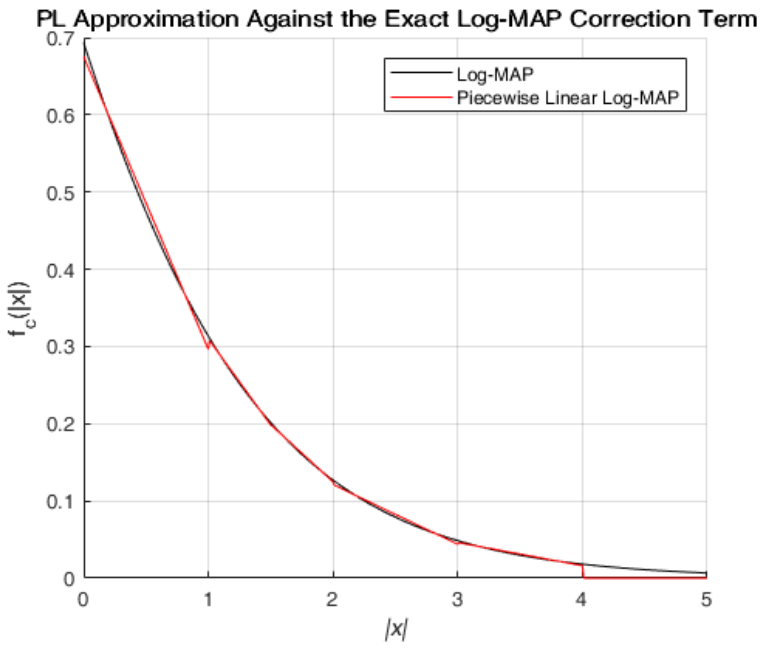


Figure 2-17 The Piecewise Linear Log-MAP plot against the exact correction function

The illustrated PL approximation to the exact correction function based on Table 2-1 is given in Figure 2-17 which is very close to the exact Log-MAP.

Complexity Analysis

However, in this work, there are no discussions on hardware implementation aspect of the PL approximation and limited BER performance analysis made. By dividing the region of $|x|$ into 5 regions, more serially cascaded comparators are required before the respective multiplication and addition can be done and the results, selected through the multiplexer. This inadvertently incurs more complexity, cost, and delay throughout the PL computation.

2.2.10.6 Multivariable Taylor Series (MTS)

Another recent development in the approximation to correction function is presented in [31] using multivariable Taylor series to approximate the correction function. In this work, up to four $|x|$ argument are considered as input to the correction function or max^* operation. The four arguments are firstly ranked. For brevity, $x_1 > x_2 > x_3 > x_4$. The minimum value, x_4 is simply

discarded and the correction function approximation for multivariable Taylor series is given as [31]:

$$max^*(x_1, x_2, x_3, x_4) \approx max^*(x_1, x_2, x_3) \quad \text{EQ. 2-34}$$

$$max^*(x_1, x_2, x_3) = log(e^{x_1} + e^{x_2} + e^{x_3}) \quad \text{EQ. 2-35}$$

$$= x_1 + log(1 + e^{x_2 - x_1} + e^{x_3 - x_1}) \quad \text{EQ. 2-36}$$

$$= x_1 + f_{PL}(x_2 - x_1, x_3 - x_1) \quad \text{EQ. 2-37}$$

with (six) expansion points of the correction function,

$$max^*(x_1, x_2, x_3, x_4) \approx max(0, k_1 x_1 + l x_2 + m_1 x_3 + C_1, \ldots, k_i x_1 + l_i x_2 + m_i x_3 + C_i) \quad \text{EQ. 2-38}$$

where k_i, l_i, m_i and C_i are real values computed at (six) expansion points (EP) based on $(|x|, |y|)$, where $|x|$ and $|y|$ represents the difference values of $|x_2 - x_1|$ and $|x_3 - x_1|$ respectively. Computational complexity can be reduced with fewer expansion points as defined in Table 2-2 [31]:

Table 2-2 Number of computation cycles for difference EP

No. of EP	Approximation Results
3	$max(x_1, 1.0986 + 0.3333 * (x_1 + x_2 + x_3), 0.8853 + 0.4683 * (x_1 + x_2) + 0.0634 * x_3, 0.6655 + 0.787 * x_1 + 0.1065 * (x_2 + x_3))$
5	$\max(x_1, 1.0986 + 0.3333 * (x_1 + x_2 + x_3), 0.8853 + 0.4683 * (x_1 + x_2) + 0.0634 * x_3, 0.7389 + 0.4954 * (x_1 + x_2) + 0.0092 * x_3, 0.6655 + 0.787 * x_1 + 0.1065 * (x_2 + x_3), 0.4411 + 0.8668 * x_1 + 0.1173 * x_2 + 0.0159 * x_3)$

The BER simulation results for 3EP and 5EP are extracted from [31] as follows:

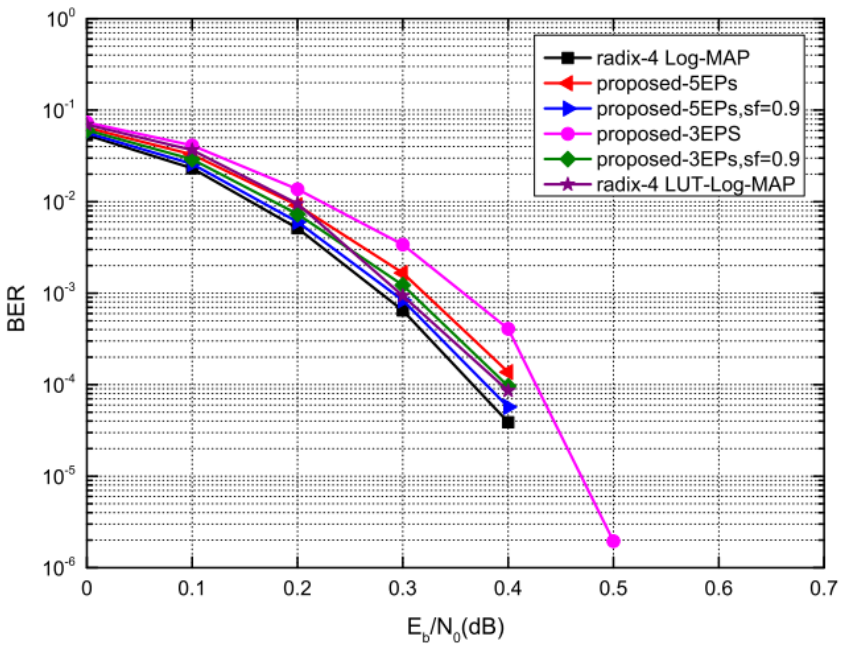


Figure 2-18 BER performance for MTS for 3 and 5 EP against radix 4 Log-MAP and radix 4 LUT method

The MTS method for 5 EPs and $sf = 0.9$, which refers to the scaling factor for extrinsic information in order to reduce the influence of the channel conditions and rely more on the Trellis for decoding as reported in [68] shows the closest performance to the radix 4 exact Log-MAP Turbo decoder while the 3 EPs version of the MTS suffers slightly but benefits from a trade off with complexity.

Complexity and Performance Analysis

In this paper, although there are some discussions and comparisons with the 4 radix Log-MAP and LUT Log-MAP method, there was no discussion on the implementation of the MTS method on hardware. Based on the approximation results in Table 2-2 , the hardware implementation is linear and can be implemented using comparators for the sorting and max functions, multiplications, adders, and LUTs for computation of the max^*.

However, in the development of the MTS approximation, the lowest magnitude of x_4 was abandoned which effectively renders the approximation into a 3 variable Taylor's series approximation. The essence to the argument $|x|$ of

correction function or max^* operator does not lie on the smallest value among x_1, x_2, x_3, and x_4 but rather, the *magnitude difference* between them. By discarding the smallest value of x_4, the decoder loses the information contributed by it especially at low SNR conditions as will be shown in the following chapters. This is evident in the case of 3 EP approximation of the MTS where the number of expansion points are lowered to reduce computational complexity. The BER performance for 5 EP however is an improvement over the 3 EP due to increase expansion points for more accurate results but requires higher computation complexities that incurs cost and potentially delays.

2.2.10.7 Second Order Approximation / Fitted Log-MAP (FLM)

The polynomial order two, fitted Log-MAP approximation, f_{FLM} to the Log-MAP correction function is a quadratic function for regions of $0 \leq x \leq 4$ and is derived from the optimal uniform Remez approximation given in EQ. 2-39 and illustrated in Figure 2-19 [46]**:**

$$f_{FLM}(x) = 0.6782 - 0.392x + 0.0576x^2 \quad x \leq 4 \qquad \text{EQ. 2-39}$$

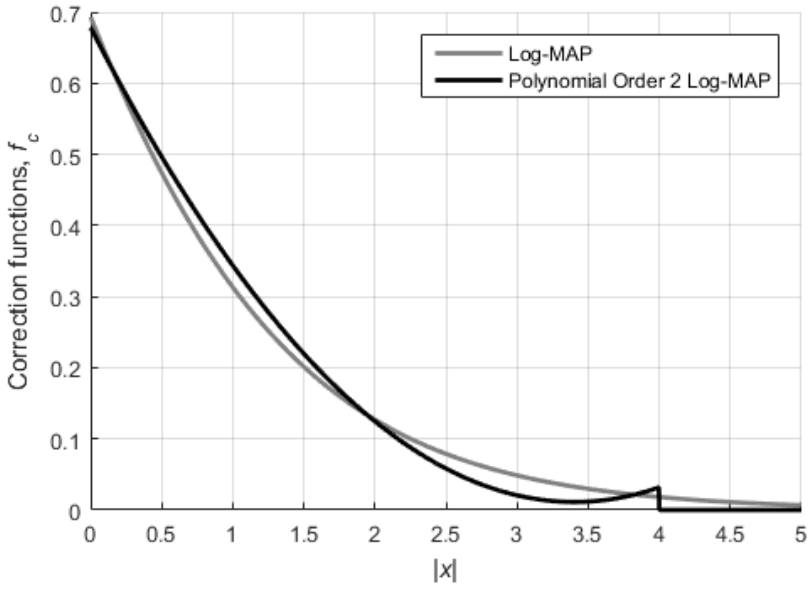


Figure 2-19 Second order approximation against the exact correction function

The f_{FLM} dips to zero for values of $x \geq 4$ consistent with the work reported in [45] where the correction function is approximated to zero where the correction function is less effective. The use of a second order polynomial fit provides a better approximation than other linear solutions which features a quadratic function that employs comparators, multipliers and adders.

Complexity Analysis

However, in striving for a closer fit, polynomial curve fits beyond the second order is not practical as the cost on hardware for higher order function will increase. This is especially significant in Turbo decoding as the max^* operation occurs at each state metric computation and complexity of the decoder exacerbates with increased constraint length and frame rate. Although the FLM presents a good fit to the Log-MAP correction function, resources on hardware for the computation of x^2 and $-0.392x$ can be costly as a minimum of two multipliers are required. Therefore, the FLM algorithm is omitted from hardware implementation in this research but included in selected simulation and analysis to provide for a more comprehensive review and comparative study.

2.2.10.8 Higher Order Polynomial Approximations

Similar to the FLM, recent polynomial regression (PR) based on Taylor series approximations are also omitted in this work due to polynomial of order from two to six as reported in [65] and [64]. While the complexity for polynomial order 2 for the PR approximation rivals that of the FLM, higher order polynomial would incur much higher complexities.

2.2.10.9 Discussions on Reviewed Suboptimal Log-MAP Solutions

In general, each approximation reviewed shows a consistent pattern of trade-off between complexity and accuracy for BER performance. Table 2-3 shows the number of mathematical operations associated with each approximation equation reviewed in this section. Approximations that are shaded are considered complex as there is a need for repeated multiplication operations which is the most complex operation when compared to the comparator, adder, or the usage of the shift register for the correction function computation. Although each of the approximation is listed to contain only one look up table (LUT), in truth, when implemented on logic circuits, the number of LUT may be more due to the various radix precisions of constants stored, and number of addresses required for storage. The LUT is estimated based on the assumption that all parameters are fixed to only one uniform format of fixed processing within a single LUT for fair comparison. The same assumption is also extended for multiplication operation. Multiplication and addition operation on FPGA can be done efficiently using the highly optimised DSP48E2 module in Xilinx FPGAs as reviewed in the next chapter but it is also limited to a fixed length. Should the precision of the results are emphasised, more of such DSP modules are needed. Also, it is observed that there is a lack of evaluation of these approximations based on fixed point processing. This thesis aims to propose an accurate, yet simple to implement approximation for FPGA hardware that is evaluated in aspects of complexity, latency, and power utilisation.

Table 2-3 Number of mathematical computation of approximations to the correction function

Approximations	Look up table	Comparator	Multiplication	Addition/ Subtraction	Barrel Shift Register
CLM [63]	1	1	0	0	0
LLM [48]	1	1	1	1	0
MLM [30]	1	0	0	1	1
LC [47]	1	2	1	1	0
PL [45]	1	4	5	5	0
MTS3 [31]	1	3	5	9	0
MTS5 [31]	1	5	10	15	0
FLM/PR order 2 [46], [64], [65]	1	1	4	2	0
PR order 3 [64], [65]	1	1	8	3	0
PR order 4 [64], [65]	1	1	13	4	0
PR order 5 [64], [65]	1	1	19	5	0
PR order 6 [64], [65]	1	1	26	6	0

Based on the comparison from Table 2-3, the ranking for complexity are as follows: CLM, MLM, LLM, LC, FLM, PL, MTS3,and PR order3 to 6 with CLM recording the least complexity for implementation. As such, further work for analysis, performance, and hardware comparison shall only be based on CLM, MLM, LLM, LC, FLM to keep the approximation complexity low for iterative Turbo decoding operations.

The effect of complexity indirectly affects the area and ultimately power utilisation for implementation on hardware. Accordingly, power utilisation for these algorithms are expected to emulate the complexity of each approximation algorithm.

In terms of latency, multi-regions approximations are shunned as the identification of regions can only be decided once the input $|x|$ has gone through all comparators before the respective approximation computation can be done. This results in further delay on top of the mathematical computation operation time. It is also for this reason that multi regions approximation such as the PL are excluded from further comparative work.

2.2.11 Turbo Code Systems Simulation Environment

In consecutive chapters, Turbo code systems simulations will be conducted to gain insights, as well as to evaluate the suboptimal Log-MAP algorithms. This section is dedicated to describe the system simulation environments that are implemented in this research. The overall simulation environment is summarised in Figure 2-20.

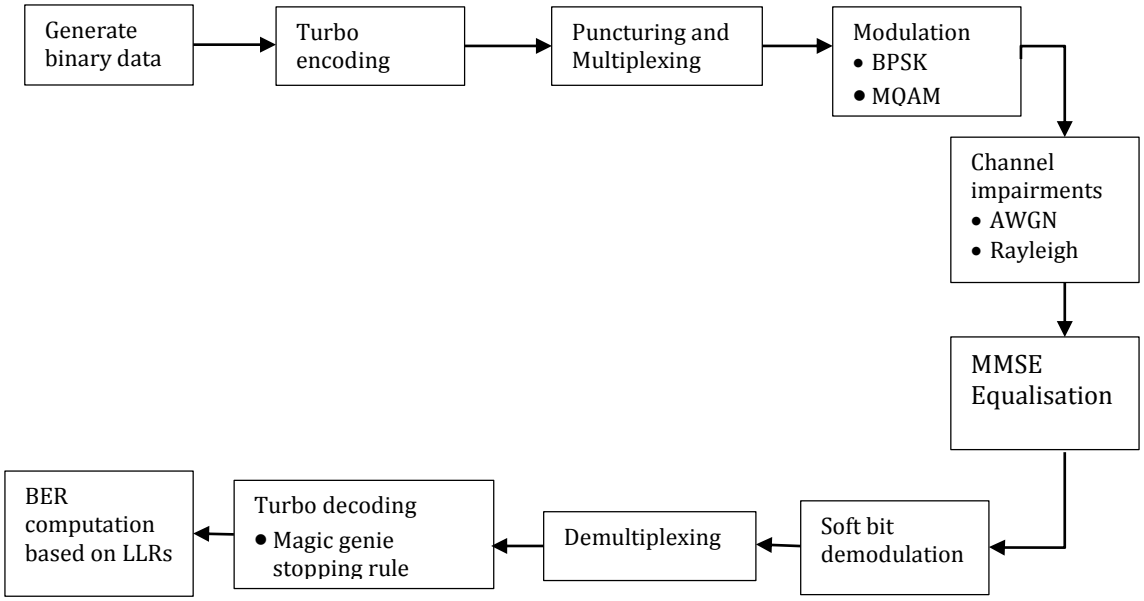


Figure 2-20 Turbo code and soft bit simulation system environment

The binary data is first generated within MATLAB and Turbo encoded according to the 4G-LTE Advanced or CCSDS Turbo encoder as detailed in Section 2.2.3. Following that, the encoded bits are then punctured for reduced transmission rates and multiplexed before modulation.

2.2.11.1 Modulation Scheme, Transmit Power and SNR Measurement

The encoded bits are modulated to Binary Phase Shift Keying (BPSK) or M-QAM with order from 16-QAM to the average 1024-QAM [36], [69], [70] and up to as high as 4096-QAM [71] for 5G mobile wireless communication environment. There are even demonstrated efforts by Magnacom-Qualcomm that achieves

the modulation order up to 16384-QAM for 5G which is also currently being championed by ZTE [72]. Whereas the deep space environment regulates the QAM order of 16 up to 256 under CCSDS standard [54]. Both standards implements gray code symbol mapping [44], [73], [74] and is accordingly adopted in this research. All transmitted symbols are energy normalised to unity to ensure a fair comparison across different modulation scheme.

The transmit power for each symbol is energy normalised to provide a fair comparison between modulation schemes. The SNR used in simulation are referred to accordingly impair the transmitted symbols and verified with MATLAB's `awgn` function as identical. All SNRs are measured in dB at the receiver end and accordingly simulated in all scenarios. Further details of the modulation scheme and constellation plots, as well as eye diagrams can be referred in Section 2.4.1 where the soft bit demodulation scheme is detailed.

2.2.11.2 Bit Error Rate Computation

The Turbo code and soft bit demodulation system simulated in this research are all based on the Monte Carlo method with a minimum of 10^6 bits simulated for each bit error rate (BER) result in a single continuous simulation scenario with uniform seeding to ensure fair comparison across different settings. The BER computation is done via bit by bit comparison. To illustrate, consider the total number of bits simulated for a given scenario is 10^6 and out of these bits (after bit by bit comparison between the data source and final decoded results) 100 bits are found to have been received in error. The BER for the particular E_b/N_0 would then be $100/10^6 = 10^{-4}$ where, E_b is the transmission energy per bit and N_0 is the noise spectral density (i.e. noise power in a 1Hz bandwidth). The bit by bit error computation is easily facilitated in MATLAB using the `biterr`

function. In certain situations, the Signal to Noise Ratio (SNR) is used and defined as $SNR = E_s/N_0$ where E_s represents the energy per symbol.

2.2.11.3 Channel Models

In this research, two of the most commonly used channels in wireless communication link are considered i.e. the Additive White Gaussian Noise (AWGN) and Rayleigh channel.

In AWGN channels, the transmitted symbol is often attenuated and noise, added with a Gaussian probability density function (p.d.f), $p(n)$ given in EQ. 2-40 [75]–[77]:

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{n^2}{2\sigma^2}\right) \quad \text{EQ. 2-40}$$

where, σ^2 is the noise variance dependent on SNR.

Channel models for satellite and deep space communication frequently consists of only AWGN noise as modelled in [78]–[80]. Hence, accordingly, all CCSDS Turbo coding simulations are simulated in pure AWGN channel.

However, for mobile communication systems, the AWGN noise is often coupled with fading impairments. The fading amplitude, h for a transmitted signal is given as [78]–[80]:

$$h = \sqrt{h_I^2 + h_Q^2} \quad \text{EQ. 2-41}$$

where, h_I and h_Q are the in phase and quadrature fading.

The Rayleigh channel fading model is used to impair transmitted symbols with (p.d.f) of [78]–[80]:

$$p(h) = \frac{h}{\sigma^2} exp\left(-\frac{h}{2\sigma^2}\right) \quad \text{EQ. 2-42}$$

where, σ^2 is the variance of the Gaussion random variable of a_I and a_Q.

In this research a memoryless channel is assumed, with constant gain and linear phase response. In other words, the channel is modelled after the flat fading Rayleigh channel similar to [81]. Since the overall aim of the thesis is focused on the Jacobian logarithm, more complex modelling for multipath scenarios are not considered to enable the use of simpler equalisation methods and reduce simulation time. The simulation time to obtain a single BER performance in this research based on custom coded Turbo encoding and decoding, can take from six to ten hours for just eight points of SNR in BER simulation, the simulations are also very dependent on number of iterations used, and M-QAM order used. Optimisation of the MATLAB codes were done wherever possible to reduce simulation time, however, with complex computation involving exponential functions such as the exact Jacobian logarithm done iteratively, the simulation time remains a challenge.

2.2.11.4 Channel Equalisation

The receiver assumes perfect knowledge of Rayleigh channel coefficient in this work and employs the minimum mean square error (MMSE) method for equalisation which are well established for mobile wireless communication systems and considered for 5G systems [82]–[84]. The MMSE approach mitigates noise enhancements effect and minimizes total errors by computing the weighting matrix, $\boldsymbol{W}$ (filter) given as follows [75]–[77], [85]:

$$\boldsymbol{W} = [\boldsymbol{H}^{\boldsymbol{H}}\boldsymbol{H} + N_0\boldsymbol{I}]^{-1}\boldsymbol{H}^{\boldsymbol{H}} \qquad \text{EQ. 2-43}$$

where, $\boldsymbol{H}$ is the matrix of channel coefficients, $\boldsymbol{H}^{\boldsymbol{H}}$ the Hermitian transpose of $\boldsymbol{H}$, $\boldsymbol{I}$ the identity matrix with size (number of transmit × number of receive antenna), and N_0 is the channel noise.

2.2.11.5 Turbo Iteration stop criteria

Due to time consuming iterative processes, this research also adopts a 'magic genie' method to stop the iteration process once the soft LLR signs matches the data source bits. This allows for the iteration process to halt before completing the pre-set number of iterations once the decoded LLRs are a match with the data frame. Accordingly, for higher SNRs, the number of iteration can be reduced before reaching the maximum number of iteration set.

2.2.12 The Turbo Decoder under SNR Mismatch Conditions

In this research there are basically two aspects for analysis on channel mismatch conditions. Firstly, the performance of the suboptimal Log-MAP algorithms are also tested in channels with SNR offsets which often occurs in practise due to inaccurate SNR estimation [86]–[88]. And secondly, the aspect of when the channel coefficients estimated at the receiver is also misestimated [89]. The system under simulation is summarised and illustrated in Figure 2-21 [89].

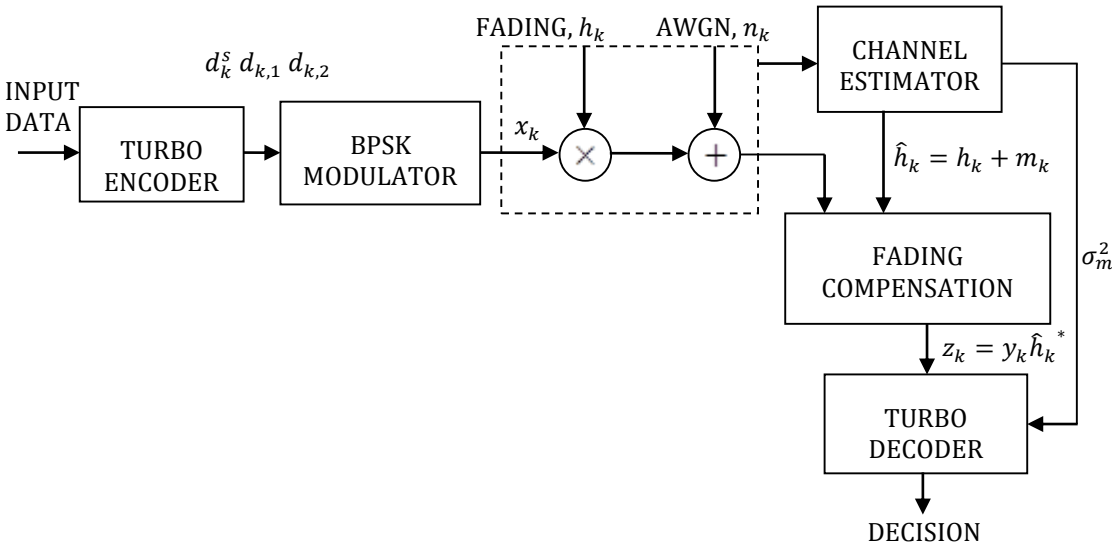


Figure 2-21 Turbo code over a fading channel for the wireless mobile

In wireless mobile systems, the input data are encoded, modulated and put through a Rayleigh faded channel with AWGN noise. Often enough, the SNR estimated at the receiver is inaccurate and is termed henceforth as SNR mismatch represented by SNR offsets in simulations. Also, at the receiver, fading compensation is done with the often erroneous estimated channel coefficient, $\widehat{h_k}$ along with erroneous channel coefficient estimation linked variance σ_m^2. The receiver performs fading compensation to give the decision variables, z_k that is decoded iteratively to give the decision bits.

It has been reported in various works that the suboptimal Log-MAP Turbo decoder is sensitive to SNR mismatch while the Max Log-MAP is mostly unaffected by it [86], [87], [90] except for conditions when the channel coefficient, $\widehat{h_k}$ is inaccurate. In [89], it is shown that the Max Log-MAP is indeed sensitive to SNR offsets when the channel coefficient is incorrect. Furthermore, it is also reported that the sensitivity of the Log-MAP and suboptimal Log-MAP Turbo decoder is heightened when the encoder memory length is larger or in other words, the constraint length is higher as in the case for the CCSDS Turbo decoder with 16 states Turbo decoding [91]. In this research, in addition to considering mismatch in channel coefficient estimation, an investigation into mismatch for SNR estimation is also done.

In the CCSDS standard, the channel is simpler with only AWGN noise. A mismatched decoding situation in the CCSDS decoder is where the decoder uses an erroneous SNR which is offset from the true SNR.

This provides for a motivation to evaluate all approximations to the correction function under channel mismatch conditions. The simulation results using the 4G-LTE Advanced and CCSDS Turbo decoder shall be presented in Chapter 4.

2.3 Overview of FPGA Technology

The FPGA first emerged in 1985 with XC2064™ family from Xilinx Inc. based on conventional static RAM technology with erasable programmable read-only-memory (EPROM) for programme storage. With the advent of faster, smaller, and more affordable transistors; the FPGA market continues to thrive with more vendors such as Altera (later acquired by Intel), Actel, Lattice, Crosspoint, Algotronix, Prizm, Plessey, Toshiba, Motorola, and IBM. Recent changes in the market saw smaller players in FPGA technology disappearing from the market due to technology rationalisation as well as emergence of the then; expensive and dominant static random-access memory SRAM technology for FPGA. The market is now dominated by Xilinx and Intel-Altera, but more importantly, FPGA technology has also grown from simple glue logic component to a complete System on Programmable Chip (SoPC) with physical and soft processors, dedicated DSP hardware, memory, and high speed I/O. The FPGA is normally configured using hardware description languages such as VHDL or Verilog.

The modern FPGA generally possesses the following features:

- ***Configurable Logic Block*** - Configurable logic tiles are the core logic structure that consist slices of look-up tables (LUTs), carry chains, and registers. These slices can be configured to perform logical functions, arithmetic functions, memory functions, and shift register functions. [92]
- ***Memory Organisation*** - For on chip storage
- ***DSP Processing Slices*** - Supports multiplication, MAC, multiply add, three input add, barrel shifting, wide-bus multiplexing, magnitude comparator, bit-wise logic functions, pattern detect, and wide counter. Multiple slices can be configured to form a wider range of DSP functions. [93]

- ***Clock Networks and PLLs*** - Clock distribution, shifting as well as frequency synthesis.
- ***I/O and External Memory Interfaces*** - Interfaces FPGA to external input and outputs and external memory.
- ***Routing resources*** - Interconnects all logic blocks and IPs
- ***Intellectual Property (IP) macros*** - Predesigned functional blocks known as IP cores that allows for enhanced productivity applied to a range of implementations, from dedicated circuit layout design, efficient code for programmable DSP or RISC processor, or core descriptions in hardware description language (HDL).

In this research, the implementation platform of choice will be Xilinx FPGAs influenced by Xilinx design solutions, popularity of Xilinx in the industry, and Xilinx's comprehensive solutions ranging from FPGA devices and advanced software integration with MATLAB/Simulink for rapid prototyping.

Other platforms such as Intel's Altera FPGA range was considered at the beginning of hardware implementation stage. The key difference between the choices in platforms are the technology used for which at the moment of writing favours the Xilinx FPGA with the 16nm FinFET technology. However, since the proposed implementation boils down to DSP modules and other commonly used logics, the difference between the two platforms are minimal. There is only one study known to have compared compatible 28nm fabrication technology between Xilinx and Intel's Altera for which the FPGA boards are tested in cryogenic temperatures. The overall performance in the test are generally positive and equal with faster operations however, it was reported that the Xilinx FPGA is more tolerant to low temperature operations [94].

2.3.1 Review of Xilinx FPGAs

The FPGAs based on the 16nm FinFET also known as the Ultrascale+ range offers higher clock rate capabilities, enhanced DSP blocks that are key for high throughput radio systems up to 825MHz speed grade [95]. The Ultrascale+ technology utilises '3D' transistors that allows for smaller device footprint, higher speed grade, and low power consumption [95], [96]. Xilinx also offers complete tooling with the Ultrascale+ with the Vivado 2017.4 design suite,

There are two ranges of FPGAs based on the 16nm FinFET within the Ultrascale+ family offered Xilinx, namely the Kintex Ultrascale+ and the Virtex Ultrascale+. Both FPGAs share the same core architecture and recipe for manufacturing by TSMC (Taiwan Semiconductor Manufacturing Company) semiconductor foundry. The only difference between both ranges of products are in terms of resource count as shown in Table 2-4 [96]:

Table 2-4 Kintex and Virtex Ultrascale+ device resources

Features	Kintex Ultrascale+	Virtex Ultrascale+
System Logic Cells (K)	356–1,143	862–3,780
Block Memory (Mb)	12.7–34.6	23.6–94.5
DSP (Slices)	1,368–3,528	2,280–12,288
DSP Performance (GMAC/s)	6,287	21,897
Transceivers	16–76	32–128
I/O Pins	280–668	208–832

2.3.1.1 DSP Slice (DSP48E2)

Central to future discussions on the research's hardware implementation, the UltraScale+ features the DSP48E2 as the DSP slice for computation at the maximum speed of 891 MHz [97]. The UltraScale+ devices features many highly optimised, low power DSP slices that combines high speed with small footprint and design flexibility. The DSP48E2 primitive's basic functionality is shown in Figure 2-22 [48]. Improved difference from its predecessor includes improved

multiplier width, increased length and flexibility in pre-adder, fourth operand addition to the ALU, and wider XOR in multiplexers.

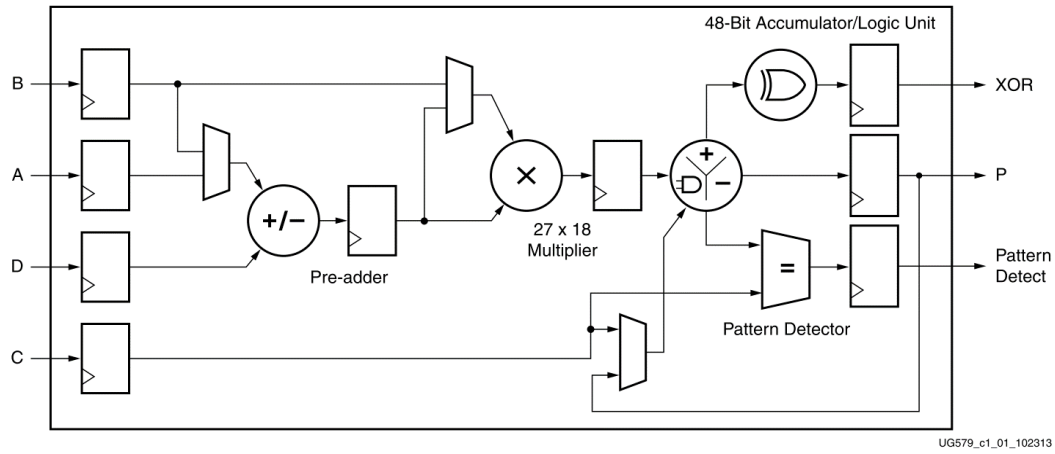


Figure 2-22 Basic functionality of the DSP48E2 [93]

Features supported by the DSP48E2 includes:

- Multiply
- Multiply accumulate (MAC)
- Multiply add
- Four input add
- Barrel shift
- Wide-bus multiplexing
- Magnitude comparator
- Bitwise logic functions
- Wide XOR
- Pattern detect
- Wide counter

The architecture also supports cascading multiple DSP48E2 slices to form wider computation functions without the need to build custom arithmetic functions from general logic.

2.3.2 System Generator for DSP, (XSG) Simulation Environment

Xilinx's System Generator (XSG), is integrated into MATLAB's Simulink tool and equipped with powerful cores for high level DSP modelling and design. It features a schematic based design tool with graphic modelling and simulation of FPGAs. An algorithm can be described directly in fixed point arithmetic and translated into hardware implementation with a Simulink library called Xilinx Blockset. The blockset consists of DSP functions and algorithms and fundamental DSP blocks such as adders, multipliers, and registers.

The XSG automatically produces command files for FPGA synthesis, HDL simulation, and implementation tools; allowing the designer to work in a fully graphical environment. XSG blocks works together with other libraries in Simulink. However, only XSG blocks will be converted to hardware. This is done by mapping the blocks to highly optimised Intellectual Property (IP) cores that generates a hierarchical HDL netlist. If a required block is not included in XSG, the algorithm can be developed using MATLAB code and later translated into HDL. The modular approach using XSG also allows for the designer to exploit the flexibility and parallel computing capabilities that are inherent to FPGA platforms. The overall design flow for FPGA implementation in future chapters are summarised in Figure 2-23.

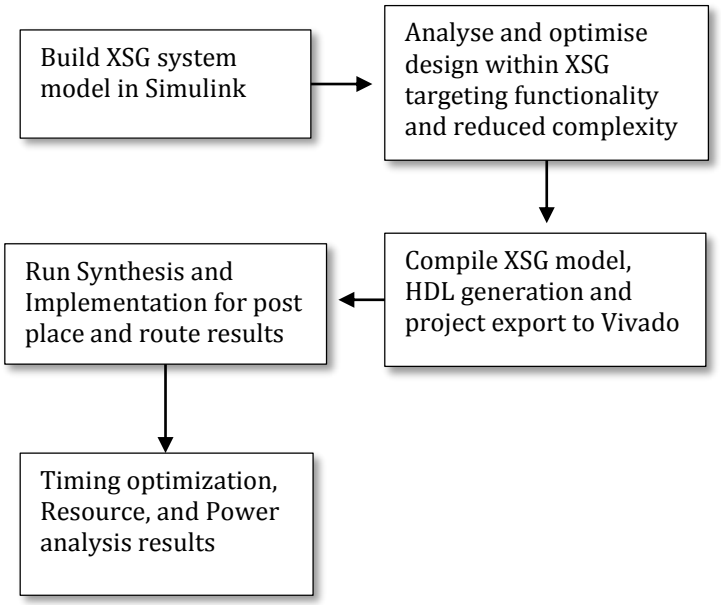


Figure 2-23 System design flow

The process begins with a model being built in Simulink, then analysed, and optimised to ensure that the design meets requirements for functionality and complexity. Once the model is optimised, the design is then elaborated for FPGA implementation. Optimisation for FPGA implementation is also done to ensure fixed point data types are suitable, and that difference between the Simulink and XSG model are acceptable. The HDL from finalised XSG model is then

generated and ported to Vivado for Register Transfer Level (RTL) synthesis tools to translate Electronic Design Interchange Format (EDIF) netlist files to FPGA programming files. The existence of several optimisation exercises in the design flow, particularly at the early stage helps in minimizing errors, thus shortening the design time from the beginning. The graphical modelling nature of the design flow not only eases design effort but allows design optimisation for complex, and parallel architectures.

2.4 M-QAM Modulation and Soft Bit Demodulation for Wireless Communication

As indicated in the previous sections, the Jacobian logarithm can be adopted for use in M-QAM soft bit demodulation. In this research, only square QAMs are considered for simulation for use in mobile wireless communication as explained in Section 2.2.11.1. The symbols are Gray coded mapped, energy normalised, and extracted from the research's simulation exercises for the 4G-LTE Advanced Turbo decoding simulation environment. The MATLAB codes to generate the following results can be referred in Appendix A.

The respective modulation scheme symbol mapping reference, simulated constellation mapping for 1000 symbols and their respective eye diagrams are presented in Table 2-6.

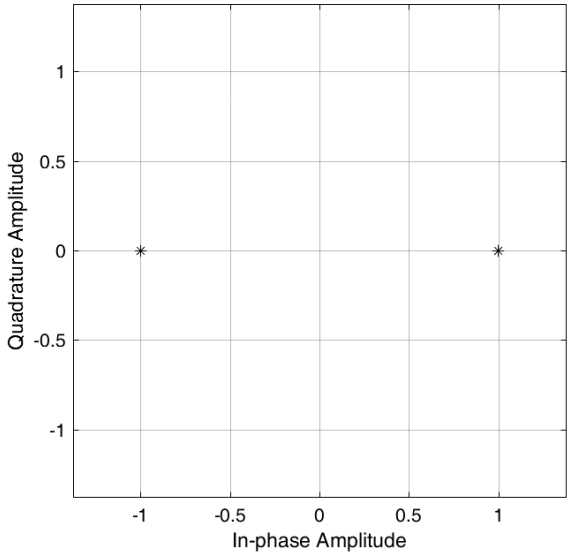
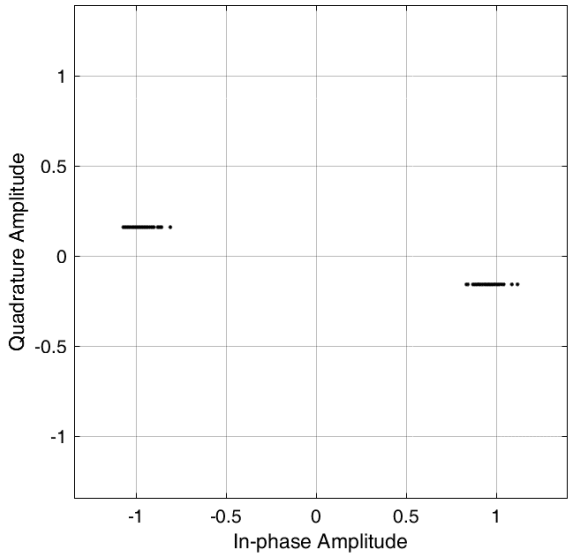
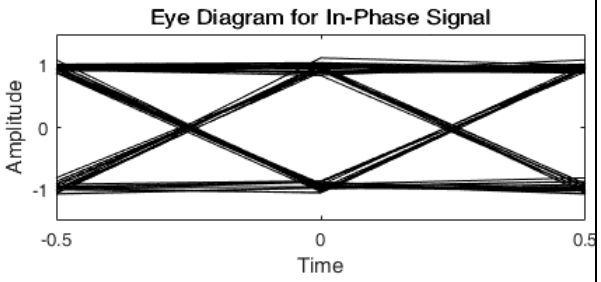
Simulation Setup

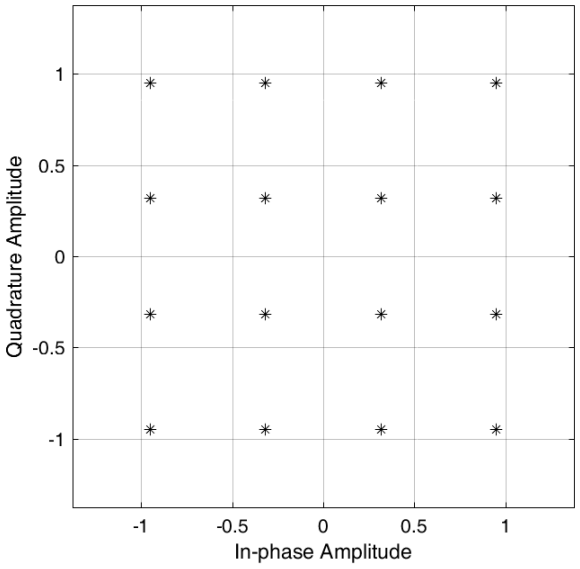
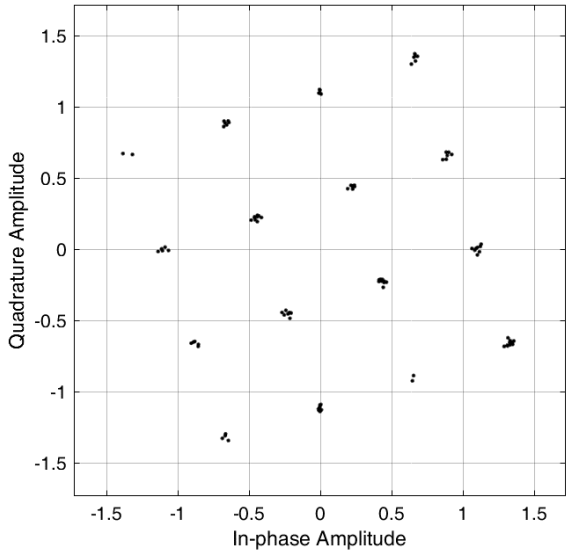
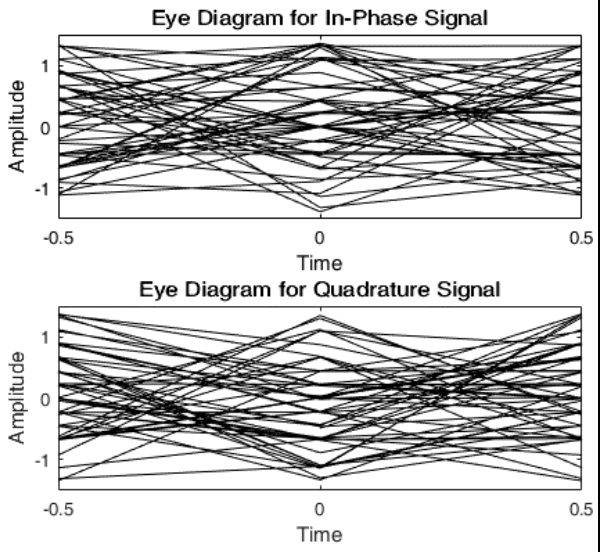
The simulation setup for the results in Table 2-6 is listed in Table 2-5 [12], [52].

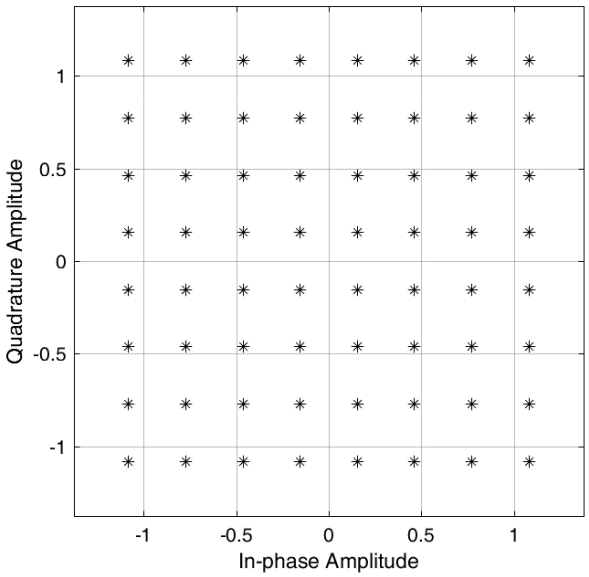
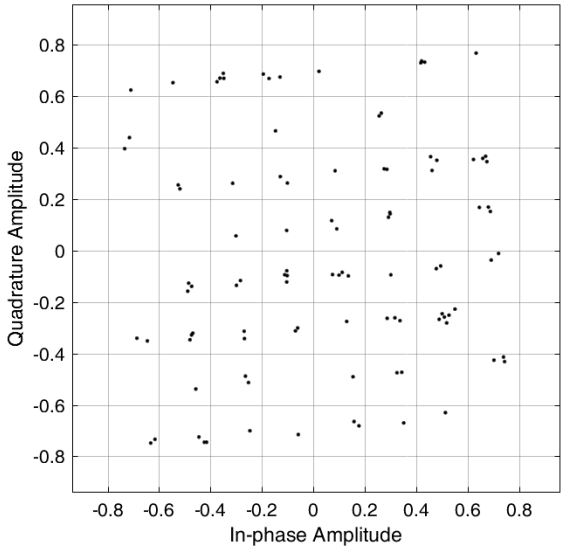
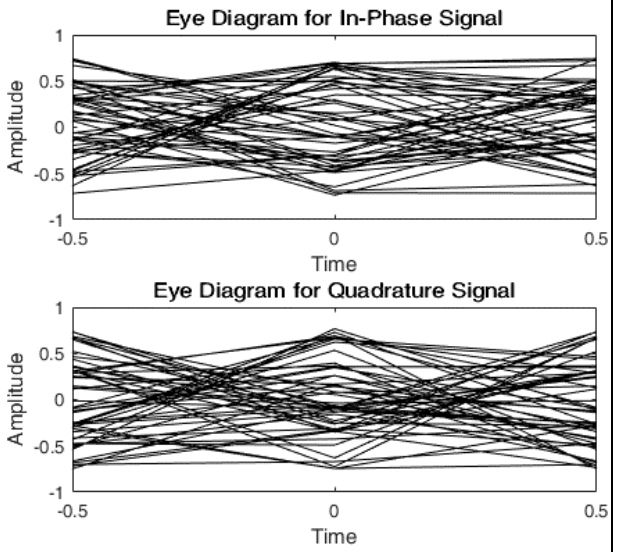
Table 2-5 Simulation setup for QAM constellation diagram and eye diagram

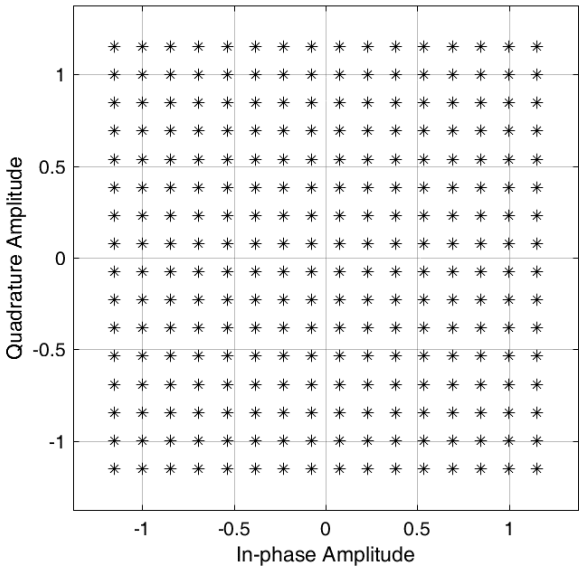
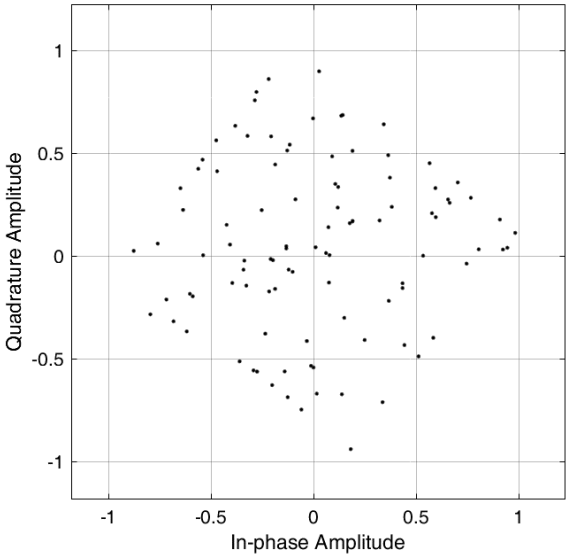
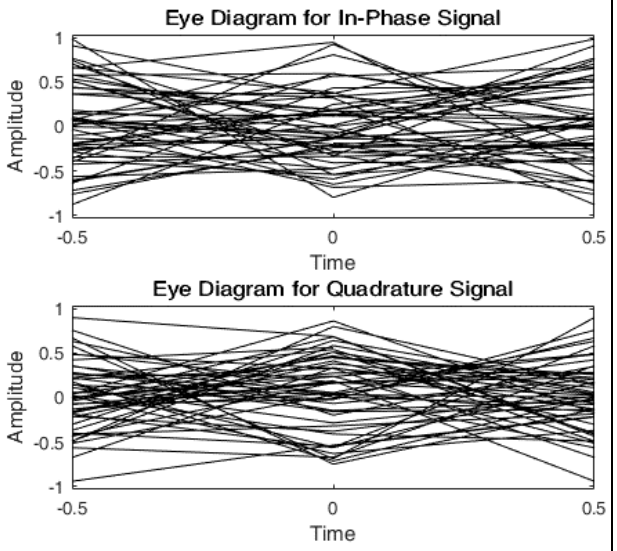
Parameter	Measure
Total symbols simulated	50
Channel	AWGN, Rayleigh flat-fading
Modulation type	BPSK,16, 64, 256, 1024, 4096
E_b/N_0 simulated for all modulation	25 dB
Received signal condition	Pre-MMSE Equalisation
Number of samples for each eye diagram trace	2
Channel coding scheme	Turbo
Rate	1/2
Generator polynomial	g[13,15]

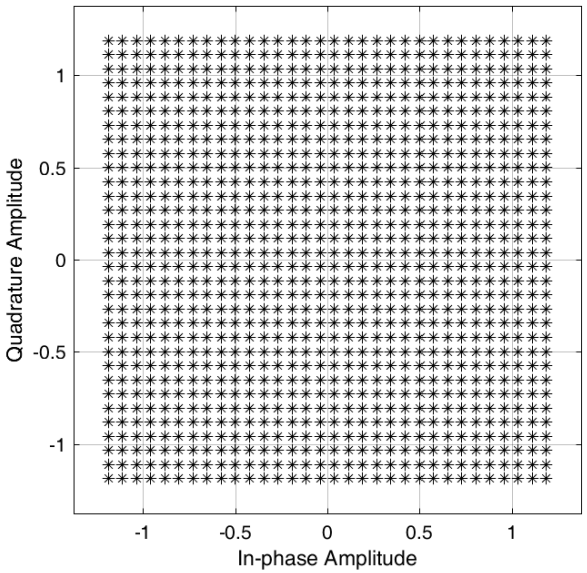
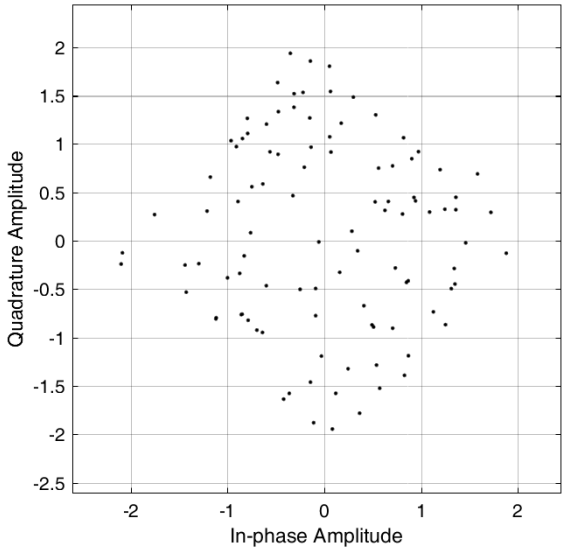
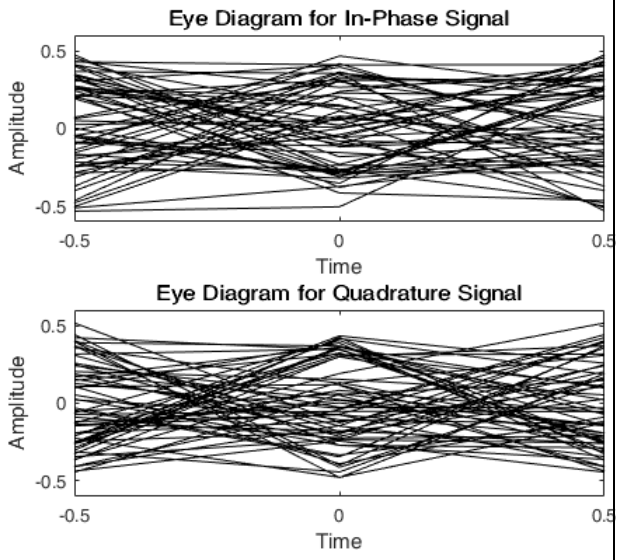
Table 2-6 Modulation constellation mapping references, received constellation symbols, and eye diagram before equalisation

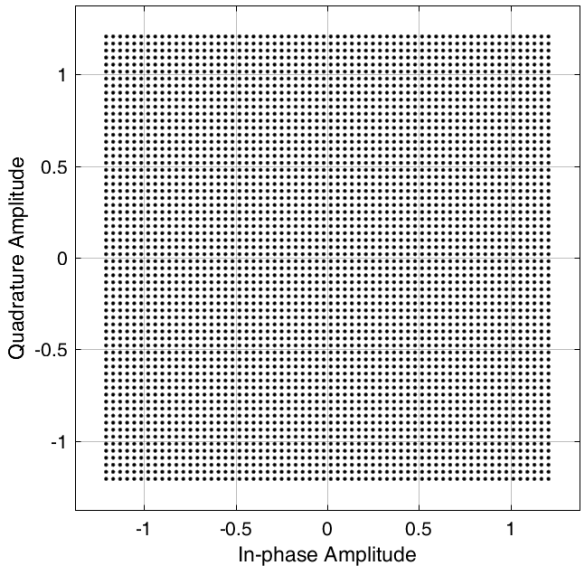
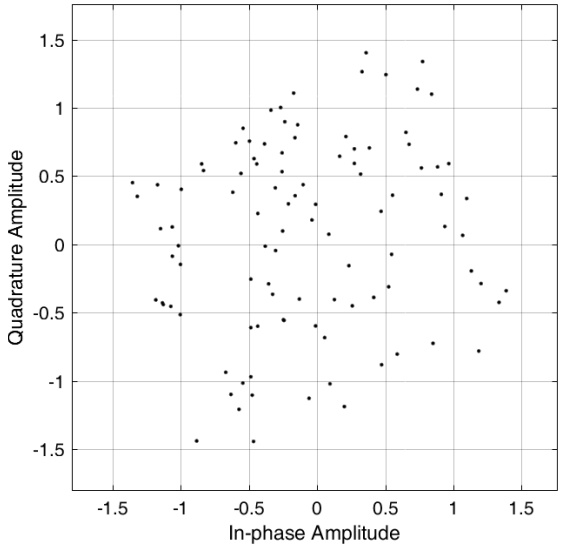
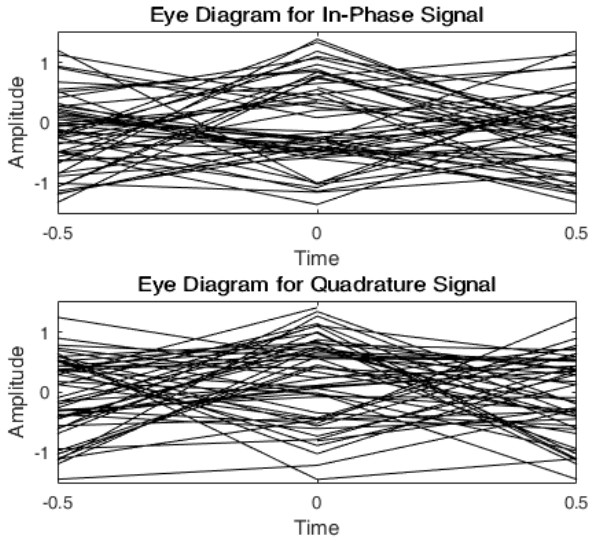
M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
BPSK 		

M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
16-QAM 		

M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
64-QAM 		

M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
256-QAM 		

M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
1024-QAM 		

M-QAM constellation map	Received Signal Scatter Plot (50 symbols, $\frac{E_b}{N_0} = 25\ dB$)	Eye diagram
4096-QAM 		

From the results tabled, it is evident that with increased order of modulation, higher transmit power is necessary for the decoder to overcome the effects of channel impairments. The received symbol eye diagram are the results from 1000 transmitted symbols which shows increasingly severe channel impairments indicated by closure of the eye diagram as the order of modulation increases with $\frac{E_b}{N_0} = 25\ dB$ fixed for all scenarios. Hence, the importance of equalisation and channel coding in recovering the transmitted data.

2.4.1 M-QAM Soft Demapping

Higher order M-QAM modulation is a key technique in achieving throughputs necessary for targeted 5G speed. Soft bit demodulation is an essential technique to enhance the function of Turbo decoding by providing soft decision bits or log likelihood ratio (LLR) of bits to the decoder [35], [98]–[101].

2.4.1.1 Maximum Likelihood Soft Demapping

The maximum likelihood (ML) soft demapping scheme computes brute force search for soft decision bit information and chooses the highest probability for maximum performance. However, the complexity for this technique increases exponentially along with the use of higher QAM modulation order. This is exacerbated with the nature of the LLR equation which is influenced by the AWGN channel and takes on exponential functions in its computation. The ML soft demapper method calculates the a posteriori, L values for each k bit as follows [100]:

$$L_k = ln\left(\frac{Pr\{b_k = 1|r_k\}}{Pr\{b_k = 0|r_k\}}\right), k = 0,1, \dots m-1 \qquad \text{EQ. 2-44}$$

where, b_k is bit '1' or '0' in the Gray code mapping scheme; L_k the bit wise LLR for the k-th bit; m the number of bits per M-QAM symbol, $2^m = M$; and r_k=k-th baseband symbol received.

Assuming the Gaussian noise channel, the bit wise LLR for ML technique are expanded as follows [34], [100]:

$$L_k = ln\left(\frac{\sum_{b_k=1} exp\left(-\frac{|r_k - s_k|^2}{2\sigma^2}\right)}{\sum_{b_k=0} exp\left(-\frac{|r_k - s_k|^2}{2\sigma^2}\right)}\right), k = 0,1, \dots m-1 \qquad \text{EQ. 2-45}$$

where, s_k is the modulation symbol s carrying m bits; r_k the baseband symbol received corresponding to s_k; and σ^2 the variance of complex Gaussian noise random variable.

Specifically, for QAM symbols with in phase, I and quadrature components,Q [34]:

$$L_k = ln\left(\frac{\sum_{b_k=1} exp\left(-\frac{|r_I - s_I|^2}{2\sigma^2}\right)}{\sum_{b_k=0} exp\left(-\frac{|r_I - s_I|^2}{2\sigma^2}\right)}\right), k = 0,1, \dots m-1 \qquad \text{EQ. 2-46}$$

$$L_k = ln\left(\frac{\sum_{b_k=1} exp\left(-\frac{|r_Q - s_Q|^2}{2\sigma^2}\right)}{\sum_{b_k=0} exp\left(-\frac{|r_Q - s_Q|^2}{2\sigma^2}\right)}\right), k = 0,1, \dots m-1 \qquad \text{EQ. 2-47}$$

where, $\boldsymbol{r_I}$ and $\boldsymbol{r_Q}$ represents the in phase and quadrature components of $\boldsymbol{r_k}$, and s_I and s_Q represents the in phase and quadrature components of s_k.

The ML method uses the L_k values to pass on to the decoder as soft bits. In a hard decoding approach, $b_k = 1$ if $L_k \geq 0$ and vice versa [100]. From EQ. 2-46 and EQ. 2-47, the log of exponential sum is present and increases in number of terms with higher order modulations. Such computation is still manageable when the order of modulation is small, however, with higher modulation

orders, the computation will be very complex with up to M number of log exponential sum terms for QAM modulation.

2.4.2 Efforts to Simplify the ML Soft Demapping in Literatures

This section gives a brief review of methods used in literatures to simplify the computation of the ML soft bit demapper.

2.4.2.1 Max-Log-MAP or Min Approach

The most common approximation to the ML soft bit demodulator is the Max-Log MAP approximation that was presented in Section 2.2.8, and specifically defined in EQ. 2-16 and applied into EQ. 2-46 [34], [35], [98], [100]–[104]:

$$L_k = ln\left[max_{b_k=1}\left(exp\left(-\frac{|r_k - s_k|^2}{2\sigma^2}\right) + \cdots\right)\right] - ln\left[max_{b_k=\mathbf{0}}\left(exp\left(-\frac{|r_k - s_k|^2}{2\sigma^2}\right) + \cdots\right)\right] \quad \text{EQ. 2-48}$$

$$for\ \ k = 0,1, \ldots m-1$$

Which equivalently translates to the minimum, or *min* approximation [34], [35], [98], [100]–[104]:

$$L_k = \frac{1}{2\sigma^2}\left[\underset{b_k=0}{min}(|r_k - s_k|^2) - \underset{b_k=1}{min}(|r_k - s_k|^2)\right] \quad \text{EQ. 2-49}$$

The Max Log-MAP is a fairly accurate approximation of the Log-MAP at high SNR region but introduces performance loss in the lower SNR region [105]. It is clear from EQ. 2-48 that the log sum of exponential using Jacobian logarithm or max^* operation and approximations can be applied in much of the same way as the Turbo decoder. Specifically, from EQ. 2-48 [34], [35], [98], [100]–[104]:

$$L_k = \frac{1}{2\sigma^2} max^*_{b_k=1}(-|r_k - s_k|^2, \ldots) - \quad \text{EQ. 2-50}$$

$$\frac{1}{2\sigma^2} max^*_{b_k=\mathbf{0}}(-|r_k - s_k|^2, \dots), k = 0,1, \dots m-1$$

where the multiple max^* operation can be executed two by two terms as specified in EQ. 2-26, that further decomposes into the max operation along with the correction function $f_c(|x|)$ detailed in EQ. 2-25.

2.4.2.2 Simplifications to the Max Approach in QAM Soft Bit Demodulation

Although the Max Log-MAP provides for a simpler implementation, there remains to be 2^m max operation to be computed for each symbol before a decision is made [102]. In order to reduce the need for exhaustive search by the Max Log-MAP approach, several works have been done to selectively restrict the QAM constellation region for LLR computation as reported in [35], [98], [101], [106]. The restricted regions for LLR computation are generally dependent on the channel conditions following a set of rules to restrict the search area.

In this research, the suboptimal Max approaches are not considered since the aim is to develop and evaluate a closer correction function approximation to the exact Jacobian logarithm. For this purpose, it is sufficient to limit the scenario to the exhaustive search method using Max-Log-MAP, exact Log-MAP, and with proposed solutions from this research.

3 METHODOLOGY AND DESIGN

3.1 Introduction

An overview of this chapter is summarised by Figure 3-1 as follows:

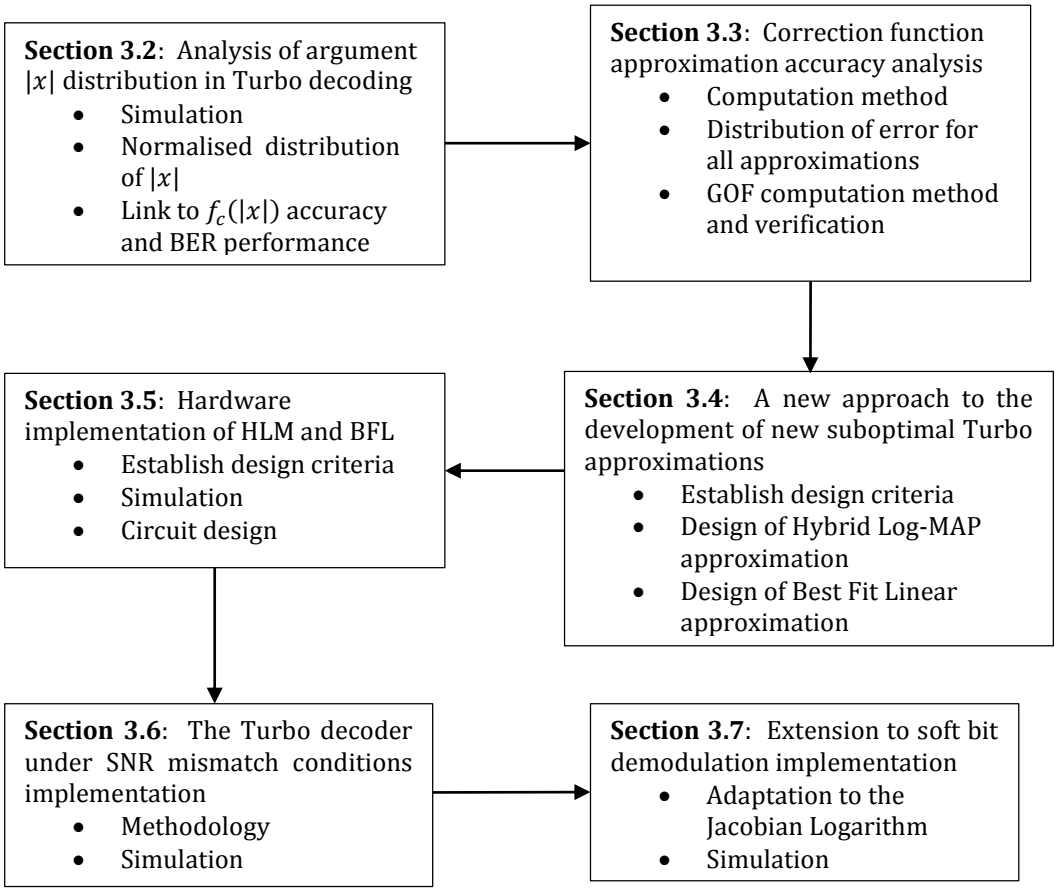


Figure 3-1 Overview of Chapter 3: Methodology and Design

The occurrence of the Jacobian logarithm or max^* operator in suboptimal Turbo decoder and in QAM soft demodulation was discussed in the previous chapter. In this chapter, the design for two alternate Jacobian logarithm or max^* operator approximations are detailed. The chapter starts off with an investigation and analysis on the distribution of argument $|x|$ in the correction function over varying SNRs. Insights gained from this investigation will allow the identification of crucial areas along $|x|$ that requires high accuracy for

improved BER performance under certain SNR conditions. This is followed by an accuracy computation and analysis of each of the two term Jacobian or max^* operator presented in the literatures along with a discussion of its effects to performance.

Another aspect to the design of the suboptimal Log-MAP solution is the hardware implementation in FPGA. Two new approximations are proposed, designed, and analysed based on hardware implementation aspects. Given the frequent occurrence of the max^* operator in decoding operations, it is crucial for the suboptimal Log-MAP solution to be optimised in terms of accuracy for reliability, complexity, latency, and power consumption to meet the requirements for future wireless systems [3]–[6], [10]. The hardware design of the new approximations are also presented.

Additional study on the Turbo decoder's performance under SNR mismatch are also included and finally, the extension of the proposed approximations to M-QAM soft bit demodulation are also discussed.

3.2 Analysis of Argument $|x|$ Distribution with Varying SNR

To recap, in each of the two term Jacobian logarithm or max^* operator, the *exact* correction function, $f_c(|x|)$ occurs as follows [45], [47], [31]:

$$ln(e^A + e^B) = max^*(A, B) = max(A, B) + f_c(|x|) \qquad \text{EQ. 3-1}$$

where, $f_c(|x|) = \ln\left(1 + e^{|x|}\right)$ and $|x| = |A - B|$ is approximated and applied to the suboptimal Turbo decoder as well as M-QAM soft demodulation techniques.

The max^* operator dominates the overall Turbo decoder computation, appearing in the computation of forward, and backward state metrics as well as the Log-Likelihood Ratio (LLR).

In this section, the distribution of argument $|x|$ in the correction function with varying SNRs are analysed before developing new approximations to the correction function, $f_c(x)$ within the max^* operator. As introduced in the literatures, there are currently two main standards that are adopted for Turbo decoding, which is the 4G-LTE Advanced and the CCSDS Turbo decoder. In the effort to determine the distribution of argument $|x|$ values with varying SNRs, the CCSDS standard is adopted in this investigation as it consists of decoders with constraint length of $K = 4$ or 16 states decoding where the occurrence of the max^*operator more than double the max^*operator occurrence in the 4G-LTE Advanced standard. With the increase of the number of states, the CCSDS Turbo decoder's performance is more sensitive towards the computation of the max^*operator.

Simulation Setup and Methodology

Using the CCSDS turbo decoder, with frame size, $N = 1000$, 5 decoding iterations, and rate, R=1/2 over the AWGN channel as specified in Table 3-1 [54] with the exact max^*operator for decoding, modulation scheme of BPSK, and total bits simulated of 10^4 bits; the normalised distribution of argument $|x|$ in f_c to compute the forward, backward, and LLR metrics are then recorded.

Table 3-1 Simulation setup for normalised distribution of $|x|$

CCSDS Turbo decoder	Parameters
Number of Frames	7
Frame Size	1000
Total bits simulated	70, 000
Constraint length	$K = 5$
Generator polynomial	$G = [23,33]$
Number of iteration	5
Rate	1/2
Channel	AWGN
Modulation scheme	BPSK
max^*operator	Log-MAP

The values of $|x|$ that is referred to the correction function computation is recorded within MATLAB using the `diary on/diary off` function. The resulting $|x|$ values are recorded in an ASCII (.txt) data file of approximately 23 million lines of data including empty lines as shown in Figure 3-2. The simulation code can be referred in Appendix B.

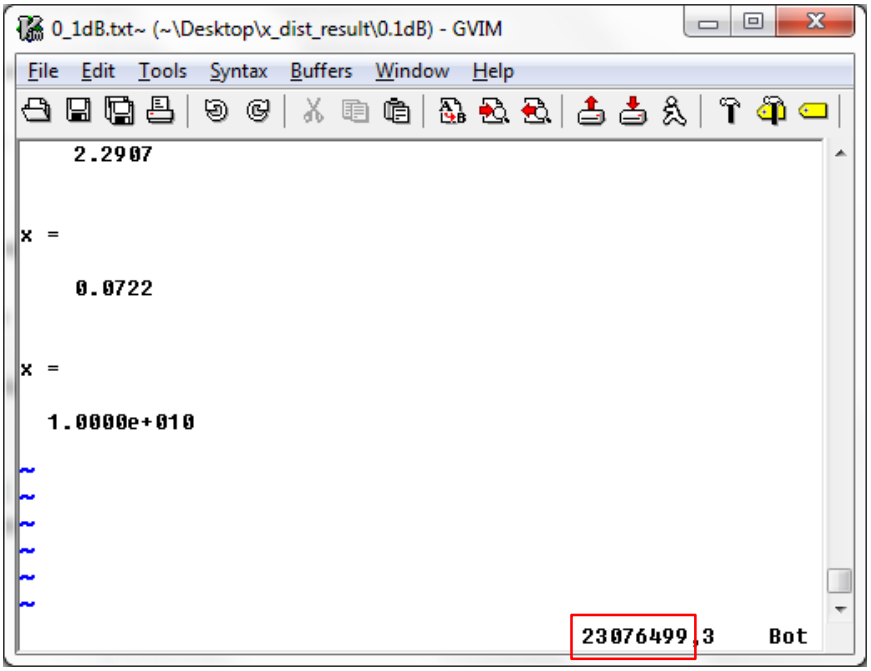


Figure 3-2 Snapshot of .txt file in GVim displaying format of raw data for 1000 bits simulated

Due to the size of the data files, normal word processing application are not effective in cleaning up the data in the text file. The use of GVim, a Unix origin text editor is employed to quickly, and effectively remove blank lines to extract pure data values before further processing is done in MATLAB using the following key commands:

Table 3-2 Key GVim commands for data cleaning

GVim Syntax Commands	Function
`:.,$!sort`	Sort lines to group white space, numbers, and blank lines
`:g/^\s*$/d`	Delete empty lines including white space
`:g/x =/d`	Delete lines with " `x =` "

It is also for this reason that the number of simulated bits are kept relatively low at 70,000 bits to ease data cleaning and processing but yet, able to comprehensively represent the distribution of $|x|$ over various SNRs. Since the purpose of this investigation is purely to investigate the distribution of $|x|$, it has been determined through trials of 10,000, 50,000, and 100,000 bits that the simulation of 70,000 bits is sufficient to adequately represent the distribution of $|x|$.

In MATLAB, the data is sorted, and only data for $|x| < 15$ are considered for analysis as the correction function values for $|x| > 4$ [46], [45] can be approximated to 0. Hence, a range of $|x| < 15$ is more than sufficient for analysis. After removal of values of $|x| \geq 15$, the data is binned using the standard Freedman-Diaconis method to determine the number of suitable bins, h of up to 567 according to the following equation [107]:

$$h = 2\frac{IQR(x)}{\sqrt[3]{n}} \qquad \text{EQ. 3-2}$$

Where, h =bin size, IQR =interquartile range or H-spread of data x, and n=number of samples in x.

The normalised mean probability distribution plot of $|x|$ for different SNRs in dB with two, y axis is generated; on the left (in black) shows the normalised $|x|$ distributions against the exact correction function in blue on the right for reference. Three normalised $|x|$ distribution graphs involving the forward, backward, and the LLR decision metric are illustrated in Figure 3-3, Figure 3-4, and Figure 3-5 respectively.

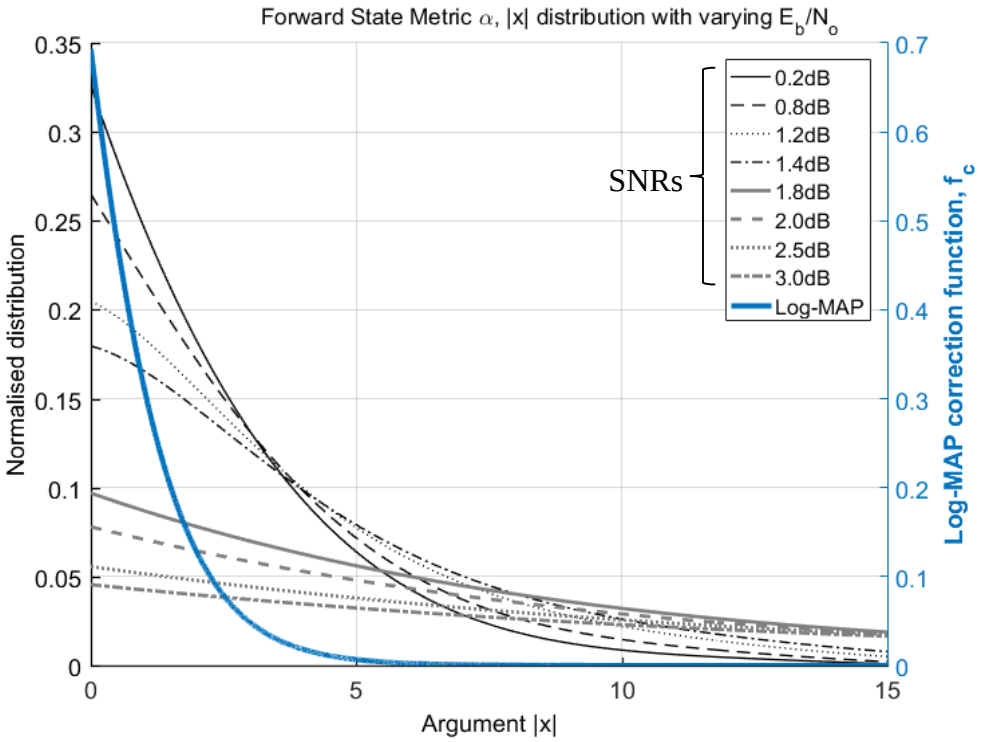


Figure 3-3 Normalised distribution of $|x|$ in forward state metric, $\alpha_k(s)$ computation

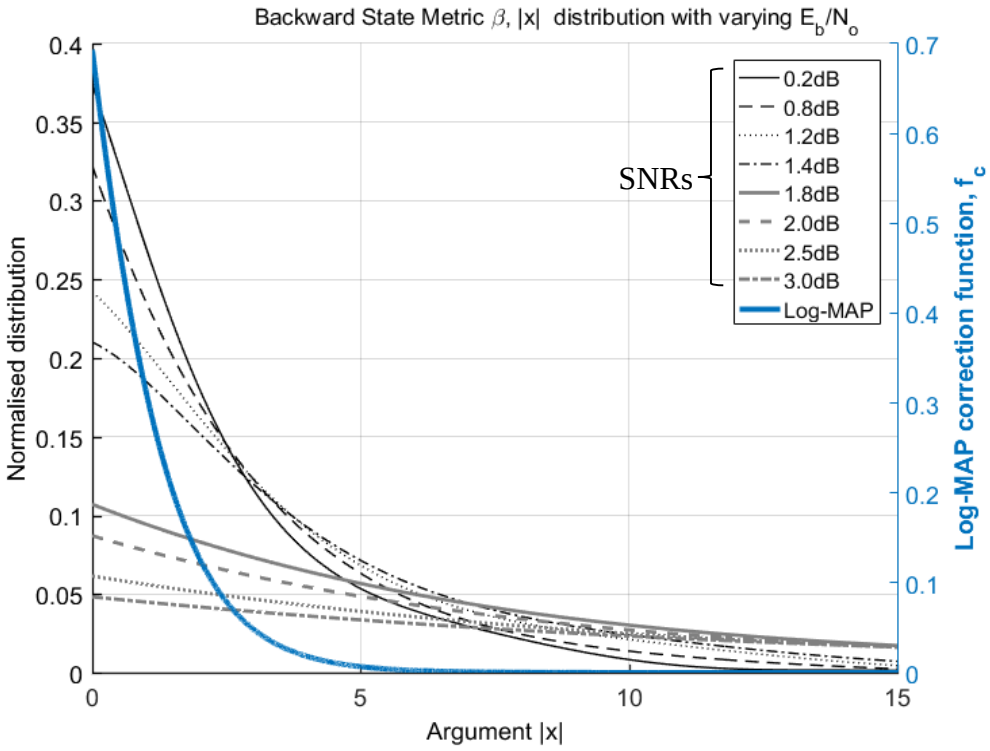


Figure 3-4 Normalised distribution of $|x|$ in backward state metric, $\beta_{k-1}(s')$ computation

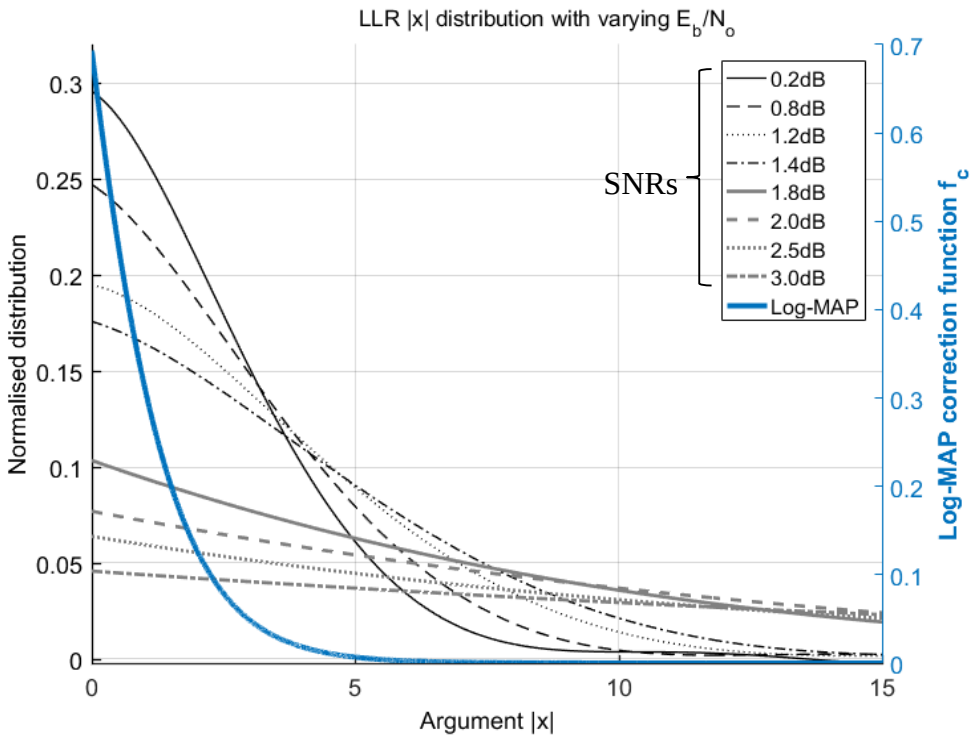


Figure 3-5 Normalised distribution of $|x|$ in LLR, $L(u_k|\underline{y})$ computation

The trend displayed by the normalised distribution of $|x|$ for various SNRs is clear and consistent for the forward, backward, and LLR values. At very low SNRs ranging from 1.4-0.2 dB, the values of $|x|$ are more likely to concentrate at the region of $|x| < 4$ where the correction function value (in blue) is most effective. Simulations above SNR of 3 dB and up to 25 dB were also performed with the trend of distribution flattening out around 0 for all values of $|x|$ at 10 dB.

This implies that the correction function plays a large role for the computation of the max^*operator at low SNRs. This is particularly important for the CCSDS Turbo decoder where the received signal from deep space is often weak and even operates at negative SNRs which often requires re-transmission when transmitted signals are severely affected by errors costing delays and power expenditure [108]. Gathered collectively, between the forward, backward, and LLR, the normalised distribution of $|x|$ for state metrics and LLR are combined in a 3D graph in Figure 3-12.

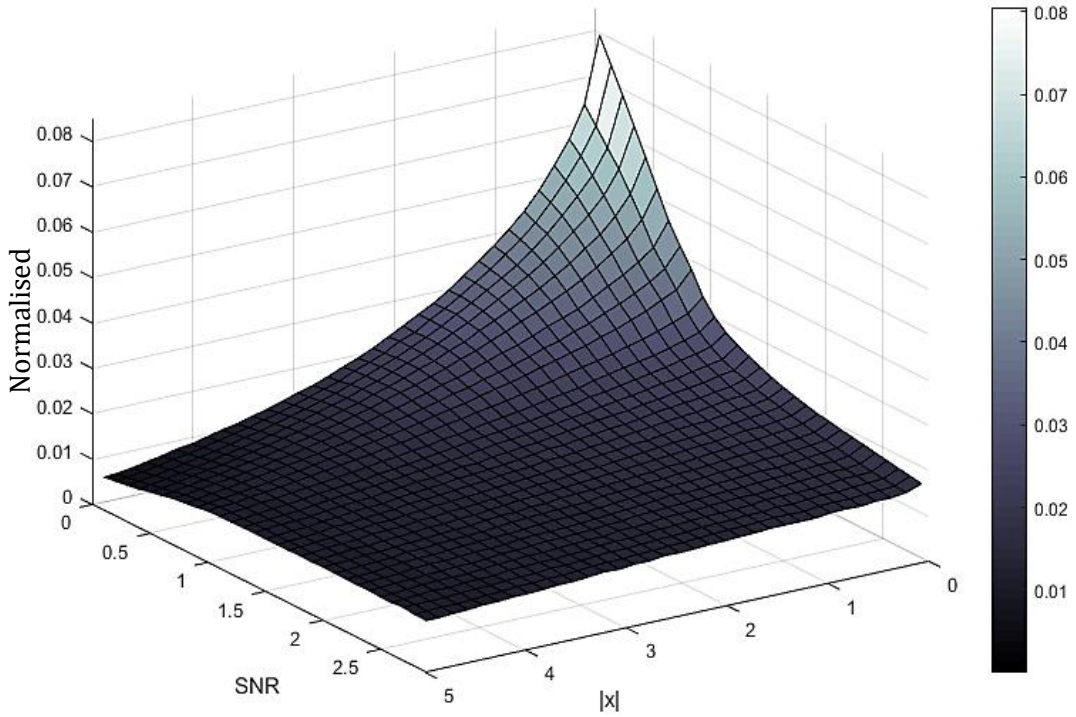


Figure 3-6 Combined normalised distribution of $|x|$ for state metrics and LLR computation

When combined, Figure 3-12 further emphasises the conclusion that at low SNRs, the $|x|$ argument within the correction function falls into the area that it is most effective with indicative values that contributes to the computation of the max^*operator. This conclusion also supports the approach to approximate values for $f_c(|x| > 4) \approx 0$ [46], [45]. Since the distribution of $|x|$ in these regions are relatively lower for any given SNR and plays a small role in contributing to the computation of the max^*. This is made more obvious for larger SNRs of more than 3dB where the distribution of $|x|$ flattens and is not likely to take smaller values where the correction function is dominant. The next section discusses the BER performance of several algorithms in the literature with a focus on their performance at low SNRs.

3.2.1 BER Performance Analysis of Approximations to the Correction Function

It is shown that at low SNRs, the correction function plays a larger role in the computation of the max^* operator. This section presents selected approximations, namely the CLM [30], [45], [64], [65], LLM [37], [48], [55], [56], Max-Log-MAP, and the exact computation for the correction function also known as the Log-MAP algorithm BER performance for discussion and relating them to the conclusion in Section 3.2.

The CLM is selected as it provides the crudest approximation to the correction function whereas the LLM adheres closely to the correction function especially at smaller values of $|x|$ due to its McLaurin series approximation as shown [48]:

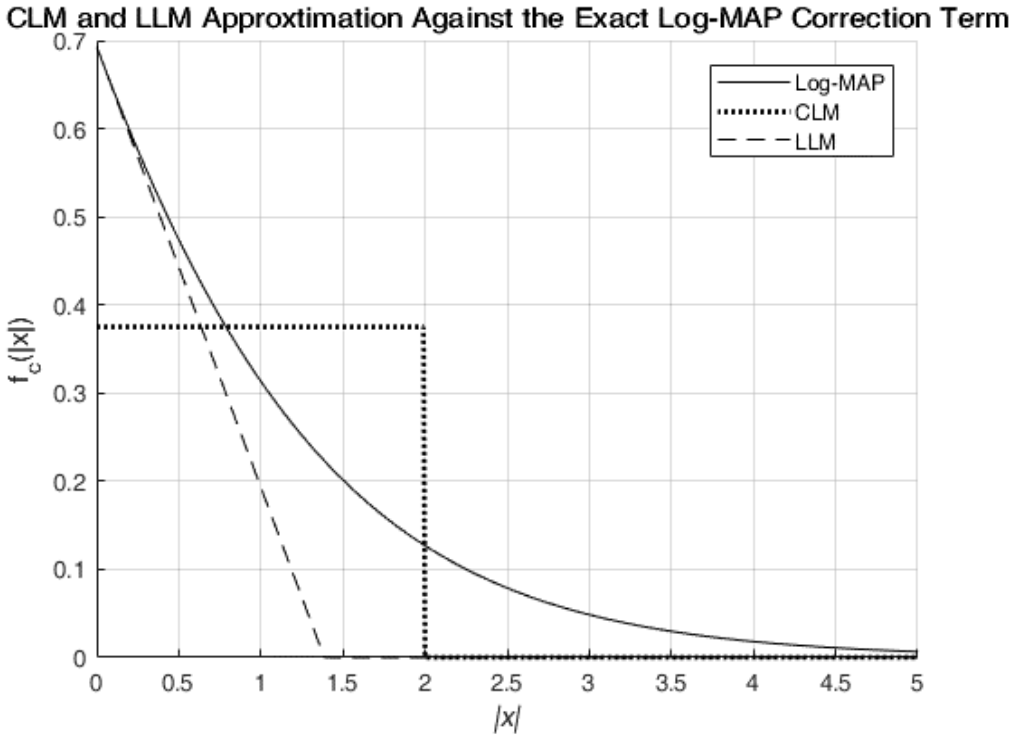


Figure 3-7 The CLM and LLM correction function

A series of simulation was conducted and the BER curves of the CLM, LLM, Log-MAP, and Max-Log-MAP are presented to analyse the effects of approximation accuracy towards performance. A similar experimental setup as shown in Table

3-3 was adopted from Section 3.2 so that the analysis remains consistent since the CCSDS Turbo decoder was selected as the platform for analysis in Section 3.2.

Table 3-3 Simulation setup for CCSDS Turbo decoder

CCSDS Turbo decoder	Parameters
Frame Size	1000
Constraint length	$K = 5$
Generator polynomial	$G = [23,33]$
Number of iteration	5
Rate	$1/2$
Channel	AWGN
Modulation scheme	BPSK
Total bits simulated	10^6 bits

Figure 3-8 presents the BER results based on bit comparison technique. Based on the BER curves, it is observed that the exact Log-MAP yielded the best performance followed by the LLM, CLM, and Max-Log-MAP which omits the correction function altogether. Based on Figure 3-7, the approximation of LLM at smaller values of $|x|$ follows the exact Log-MAP solution more closely than the CLM approximation and yields overall improved performance over the CLM.

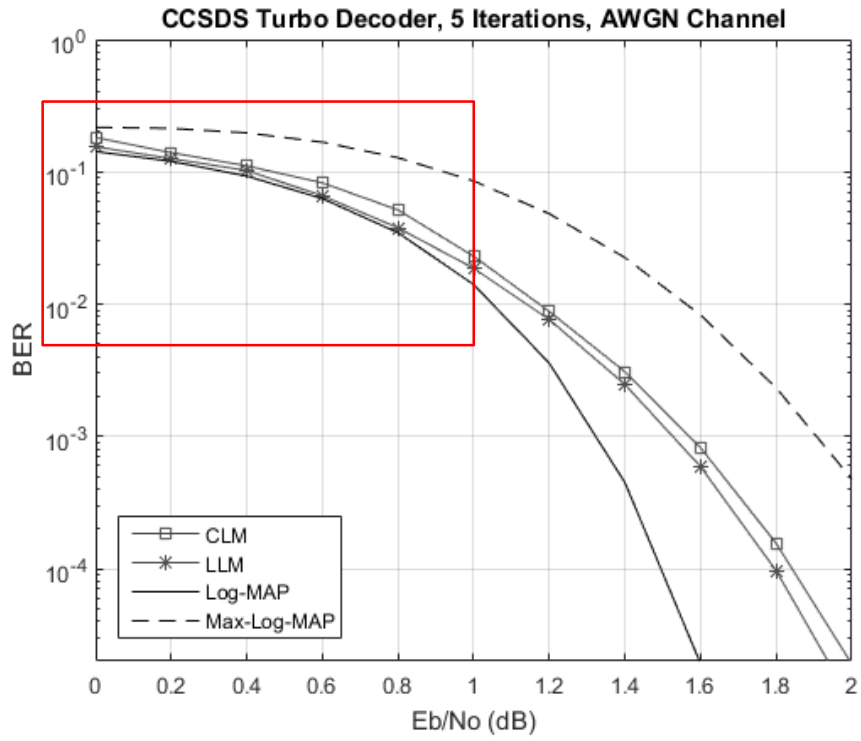


Figure 3-8 BER performance of several suboptimal Log-MAP algorithms

The highlighted portion in Figure 3-8 is enlarged as follows in **Figure 3-9**:

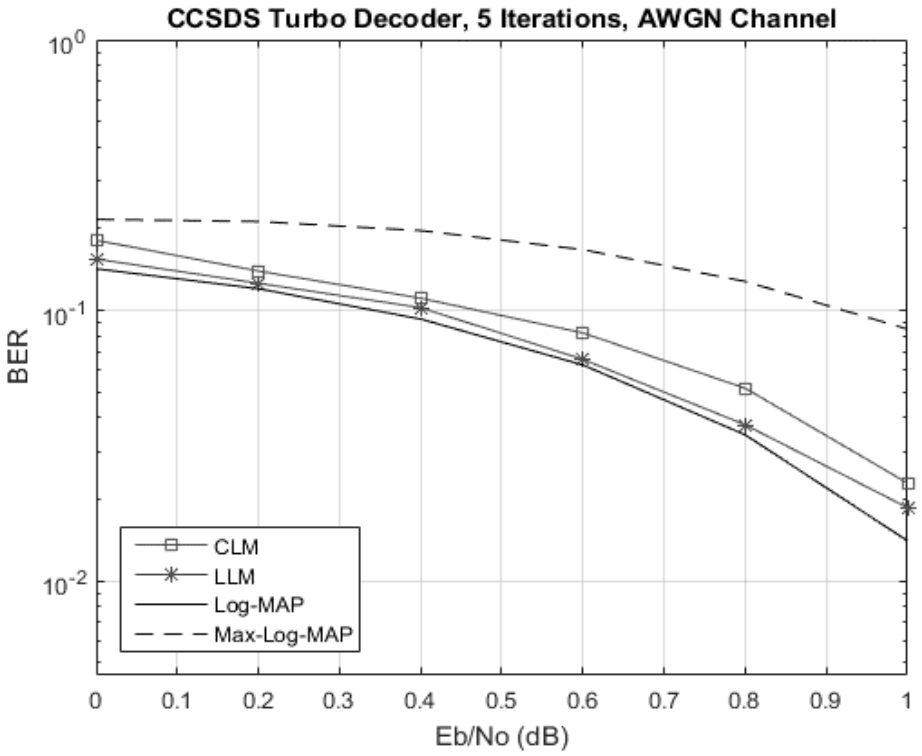


Figure 3-9 Enlarged graph for low E_b/N_0

It is interesting to note that the performance of LLM adheres more closely to the exact Log-MAP at lower E_b/N_0. Based on the conclusion from Section 3.2, the distribution of $|x|$ concentrates at smaller value of $|x|$, or more specifically, in the region of $|x| < 4$. While all simulation share the same setup, there is but one difference between the CLM and LLM results; i.e. their level of approximation accuracy. This implies that the approximation accuracy affects BER performance especially in Turbo decoders with higher constraint lengths, $K = 5$ with 16 states, such as the CCSDS Turbo decoder.

This research adopts $f_c(4) = 0.0181$ as the cut-off point for accurate approximation of the correction function, $f_c(|x|)$ [46], [45]. All values of $x > 4$ are approximated to 0 to ease approximation and hardware implementation. Section 3.3 provides a better overall insight on the effects of accuracy to BER performance for all suboptimal Log-MAP solutions in the literature.

3.3 Correction Function Approximation Accuracy Analysis

As established in the earlier section, a relationship between approximation accuracy and BER performance can be drawn. At low SNR, the approximation accuracy to the exact correction function or max^* operator is crucial for improved BER performance.

This section presents a computation method of normalised error distribution of all approximations against the exact correction function and presents graphs of these error along with a verification of these results with a section on goodness of fit (GOF) statistics of each approximation against the exact correction function. By using these results and relating them with performance, new design criteria can be formulated for better approximation.

3.3.1 Computation of Mean Error Distribution

The mean error distribution allows us to analyse the regions of argument $|x|$ in which each approximation in literature disagrees with the exact correction function $f_c(|x|)$. The exact correction function, $f_c(|x|)$ is illustrated in Figure 3-10 from $|x| = 0$, to $|x| = 5$ for analysis. Values beyond $|x| > 4$ approaches 0 and is omitted from design consideration [45], [46].

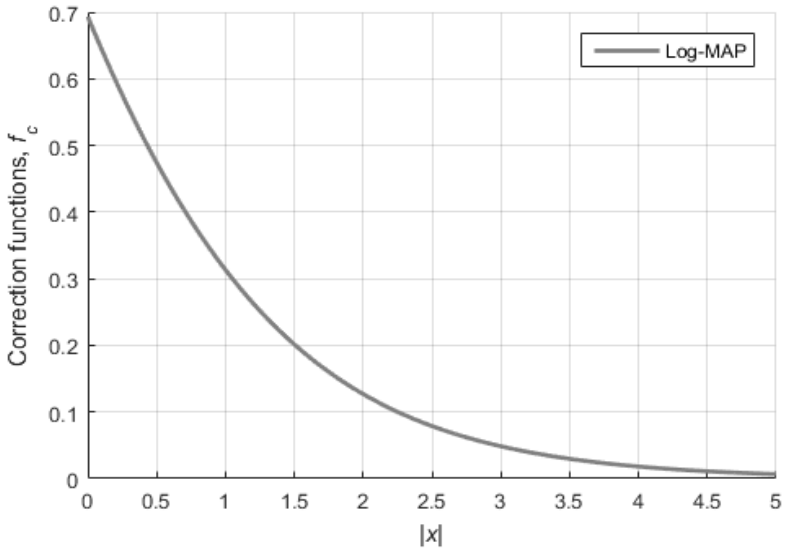


Figure 3-10 The exact correction function, $f_c(|x|)$

The method for computing the mean error distribution function for each correction function approximate, f_{approx} is summarised by Figure 3-11.

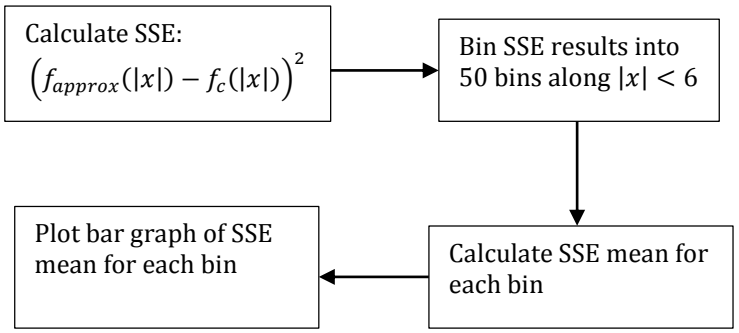


Figure 3-11 Computation of mean error distribution for an approximate correction function, $f_{approx}(|x|)$

The computation for mean error distribution is done by first computing the squared error between the approximation and the exact correction function followed by binning the squared errors into equidistance bins along the $|x| < 6$ axis for analysis. Then the mean sum of squared mean value for each bin are plotted accordingly resulting in the mean error distribution bar graph. Since the objective of these plots are to ascertain areas along the $|x|$ axis where there are discrepancies between the approximation and the exact correction function, the number of bins are set to 50 for all approximations to sufficiently illustrate these differences. The squared errors are produced at a resolution of 0.01 for $0 \leq |x| < 6$ and binned to 50 equally spaced intervals within the same interval.

The following are the respective results from this investigation for each approximation along with their respective plot against the exact correction function $f_c(|x|)$ for ease of reference followed by a discussion of results presented. Appendix C details the codes that generates these graphs.

3.3.2 Computation of Goodness of Fit (GOF) Statistics

This sections presents the methods of GOF computation for each approximation. A plot of the suboptimal solution against the exact Log-MAP will be provided for each solution together with their squared error distribution plot for analysis on the approximation's fit against the argument $|x|$.

The goodness of fit (GOF) statistics describes how well the Suboptimal Log-MAP approximation, $f_{approx}(|x|)$ fits the exact correction function, $f_c(|x|)$ solution and is calculated for each of the Suboptimal Log-MAP approximation. The GOF measures for Sum of Squared Error, Mean Squared Error, Root Mean Squared Error, and R Squared Error based on [109] are scripted in MATLAB and verified with [107] as well as with results from `fit` function within MATLAB. The equations for each GOF statistics are as follows [107]:

Sum of Squared Error (SSE)

This measure calculates the total squared deviation of $f_{approx}(|x|)$ to $f_c(|x|)$. The SSE is also known as the Residual Sum of Squares. A measure closer to 0 would indicate a more precise approximation to the exact correction function, $f_c(|x|)$ and is defined as [109], [107]:

$$SSE = \sum_x w\left(f_c(|x|) - f_{approx}(|x|)\right)^2 \quad \text{EQ. 3-3}$$

where *w* is the weight for each data point and is set at default of value 1.

Mean Squared Error (MSE)

The Mean Squared Error or Mean Squared Deviation is the average of the SSE over the length of $|x|$ points or n_x in the GOF computation. The MSE also gives the expected value of the squared error and is defined as [109], [107]:

$$MSE = \sum_x \frac{\left(f_c(|x|) - f_{approx}(|x|)\right)^2}{n_x} \quad \text{EQ. 3-4}$$

Root Mean Squared Error (RMSE)

Also known as the fit standard error and the standard error of regression, the RMSE is the estimate of the standard deviation and is defined as [109], [107]:

$$RMSE = \sqrt{MSE} \quad \text{EQ. 3-5}$$

R-Squared (R^2)

R-Squared measures how well the approximation fits the exact correction function and ranges from 0 to 1. A R^2 closer to 1 indicate that the approximation explains all variability of the response data around its mean. The R-squared is defined as [109], [107]:

$$R^2 = 1 - \frac{SSE}{\sum_x \left(f_c(|x|) - \overline{f_c(|x|)}\right)^2} \quad \text{EQ. 3-6}$$

where $\overline{f_c}$ is the mean of f_c. Although useful in determining the approximation fit, the R^2does not indicate if the approximation is biased. Hence the need for the probability of error distribution for residual analysis.

All GOF statistics are computed with a resolution of $\Delta x = 0.01$ within the range of $0 \leq |x| < 4$ [45], [46] using MATLAB at Appendix D and verified in MATLAB using the inbuilt `cftool` and GUI in MATLAB.

3.3.3 Accuracy Analysis of Approximations to the Correction Function

This section presents both the mean error distribution for each approximation against the exact correction function, $f_c(|x|)$ and GOF statistics for each approximation against $f_c(|x|)$ followed by a discussion on each results obtained. Each of these approximation was introduced in Chapter 2 and crossed referenced for ease of discussion. The exact correction function was presented in Chapter 2, Section 2.2.9 as a reference for the rest of this section.

3.3.3.1 Constant Log-MAP (CLM)

The CLM approximation was introduced in Chapter 2, Section 2.2.10.1with CLM offering the crudest approximation with a step function, albeit with two versions of step width and height, it offers the simplest approximation to the correction function. The usage of the CLM approximation with $T = 1.5$ and $C = 0.5$ or $f_{CLM}(|x|)$ is more commonly used as reported in [66], [45], [47], and [48]. However, as previously discussed, this approach has a negative effect on BER performance due to its crude approximation especially at the lower regions of $|x|$ where the exact correction function influences the Jacobian logarithm the most. The error distribution for $f_{CLM}(|x|)$ are given in Figure 3-12 as follows:

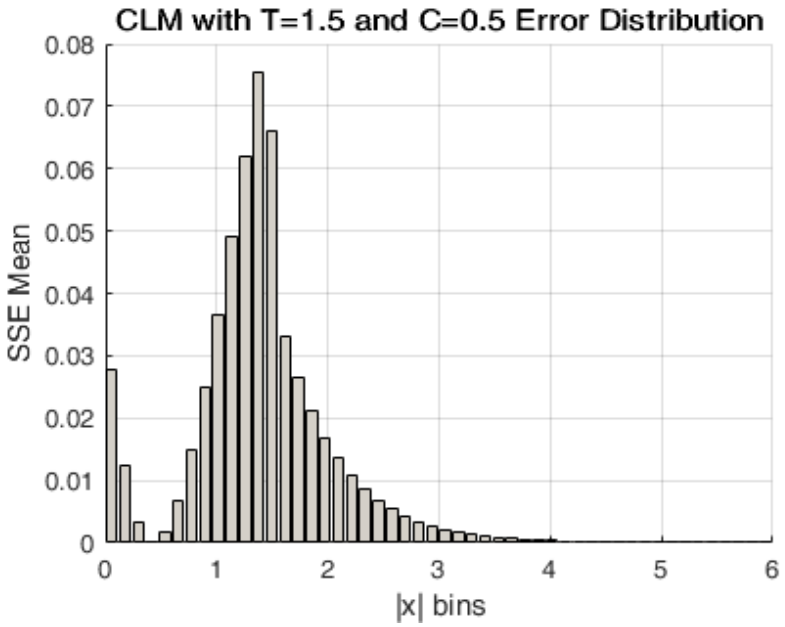


Figure 3-12 Mean SSE error distribution for the CLM with $T = 1.5$ and $C = 1/2$

The $f_{CLM}(|x|)$ shows an error peak around $|x| = 0$ in the error distribution lower regions of $|x|$ with goodness of fit statistics as detailed in Table 3-4.

Table 3-4 Goodness of fit statistics for the Constant Log-MAP with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	Measure for CLM with $T = 1.5$ and $C = 1/2$
Sum of Squared Error (SSE)	6.5098
Mean Squared Error (MSE)	0.0162
Root Mean Squared Error (RMSE)	0.1274
R-Squared (R^2)	0.5351

With a resolution of $\Delta x = 0.01$, for the region of $0 \leq x \leq 4$, the SSE measurement for $f_{CLM}(|x|)$ shows an SSE of 6.5098 which provides room for improvement, the R-Squared measure of 0.5351 is also relatively far away from the ideal 1 for accurate approximation. With high SSE measure and relatively high error distribution at lower regions of $|x|$, it is expected that the CLM approximation will not approach the exact correction function performance especially at low SNRs.

3.3.3.2 Linear Log-MAP (LLM)

The mean error distribution for the linear log-MAP approximation introduced in Chapter 2 Section 2.2.10.2 is given in Figure 3-13:

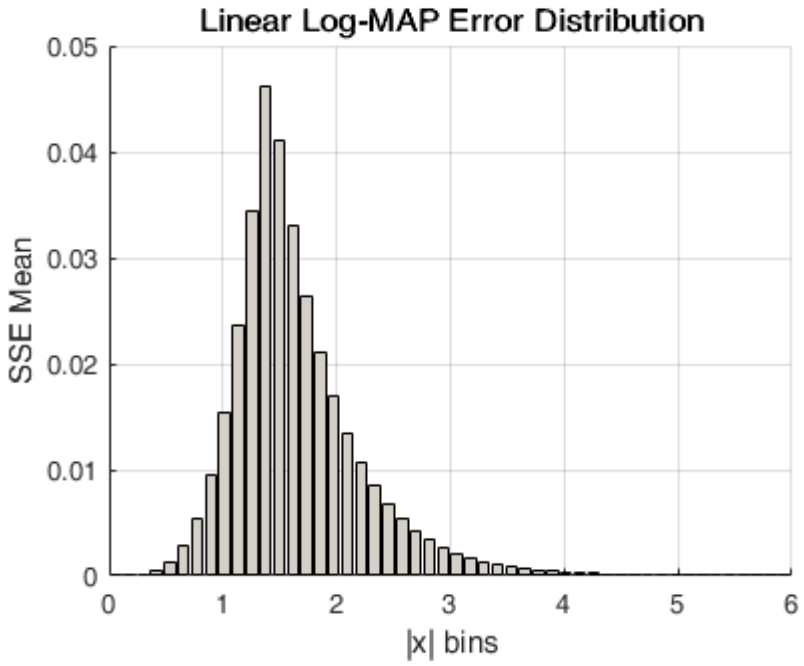


Figure 3-13 Mean error distribution for the Linear Log-MAP

As illustrated in the mean error distribution, the correction function for LLM, or $f_{LLM}(|x|)$ is particularly accurate for $|x|$ around 0 with minimal errors which is consistent from observation of Figure 2-13 and the approach taken to minimise error for small values of $|x|$ with the Maclaurin series. However, for subsequent higher values of $|x|$, the error distribution increases and the overall goodness of fit statistics shown in Table 3-5 that the LLM is a better approximation to the Lop-MAP algorithm compared to the Constant Log-MAP.

Table 3-5 Goodness of fit statistics for the Linear Log-MAP with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	Measure
Sum of Squared Error (SSE)	4.1015
Mean Squared Error (MSE)	0.0102
Root Mean Squared Error (RMSE)	0.1011
R-Squared (R^2)	0.7071

The improved fit of the LLM with SSE of 4.1015 and R-squared measure of 0.7071 over the CLM also enables better BER performance as presented in [48] and [47]. It is expected that the BER performance for LLM would be better than CLM due to $f_{LLM}(|x|)$'s more accurate approximation to the correction function.

3.3.3.3 Multistep Log-MAP (MLM)

The mean error distribution for the multistep log-MAP approximation introduced in Chapter 2 Section 2.2.10.3 is given in Figure 3-14.

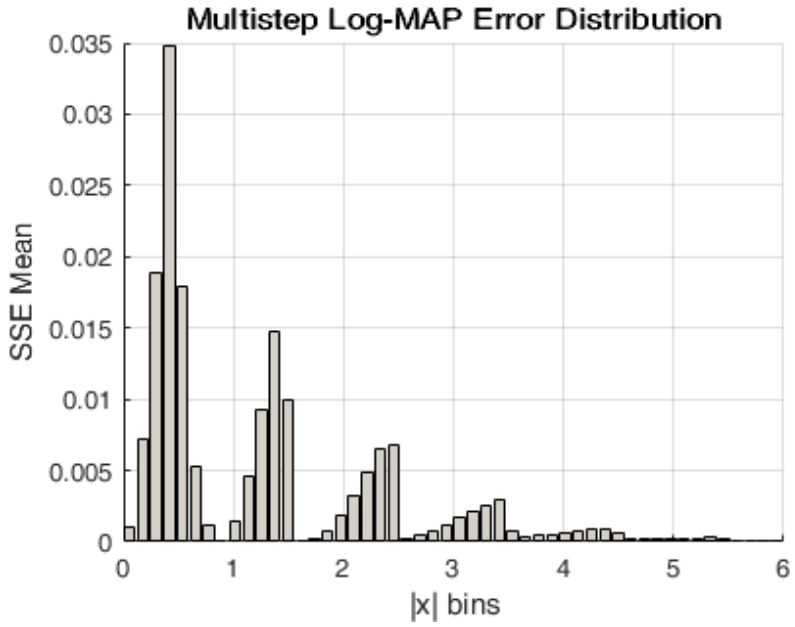


Figure 3-14 Mean error distribution for the Multistep Log-MAP

The mean error distribution for the MLM occurs at intervals of sudden bursts due to the multistep nature of the MLM. Low error distribution are found for areas of $|x|$ that agrees with the Log-MAP and the error quickly rises subsequently in bursts due to the successive step nature of MLM. The approximation of the MLM to the smaller regions of the correction function, specifically, $|x| < 1$ are still relatively high, peaking at approximately 0.035 for $|x| \approx 0.4$. Overall however, the MLM provides for a more accurate approximation to the exact Log-MAP correction function with better goodness of fit statistics compared to CLM as shown in Table 3-6.

Table 3-6 Goodness of fit statistics for the Multistep Log-MAP with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	Measure
Sum of Squared Error (SSE)	1.9782
Mean Squared Error (MSE)	0.0049
Root Mean Squared Error (RMSE)	0.0702
R-Squared (R^2)	0.8587

The GOF statistics for MLM supports the observation of the mean error distribution with a lower overall SSE of 1.9782 and better R-squared measure of 0.8587. The GOF statistics for MLM however, is not better than the LLM and it is also noted that the mean error distribution for the region of $|x| < 1$ is worse than the MLM approximation. This indicates that the MLM approximation's BER performance for low SNR to be better than the CLM but worse than the LLM.

3.3.3.4 Linear-Constant Log-MAP (LC)

The LC approximation or $f_{LC}(|x|)$ offers even closer approximation with linear operations compared to the MLM with good mean error distribution illustrated in Figure 3-15.

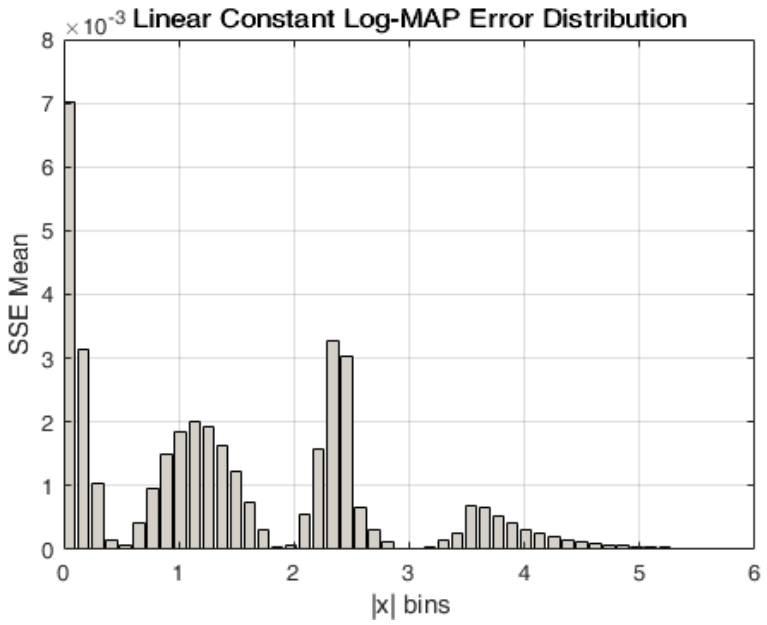


Figure 3-15 Mean error distribution for the Linear Constant Log-MAP

Although there are error peaks surrounding at the lower regions of $|x|$, the magnitude of the peak are very low with a maximum peak of 0.95×10^{-3} and this, along with the subsequent peaks are the lowest magnitudes when compared to the MLM, LLM, and CLM. The overall error distribution for the LC approximation is verified with the GOF statistics in Table 3-7.

Table 3-7 Goodness of fit statistics for the Linear Constant Log-MAP with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	Measure
Sum of Squared Error (SSE)	0.4368
Mean Squared Error (MSE)	0.0011
Root Mean Squared Error (RMSE)	0.033
R-Squared (R^2)	0.9688

Based on the accuracy for the LC approximation, the BER performance for LC should adhere closely to the exact correction function.

3.3.3.5 Second Order Approximation / Fitted Log-MAP (FLM)

The polynomial order two, fitted Log-MAP approximation, $f_{FLM}(|x|)$ mean squared error distribution are shown in Figure 3-16.

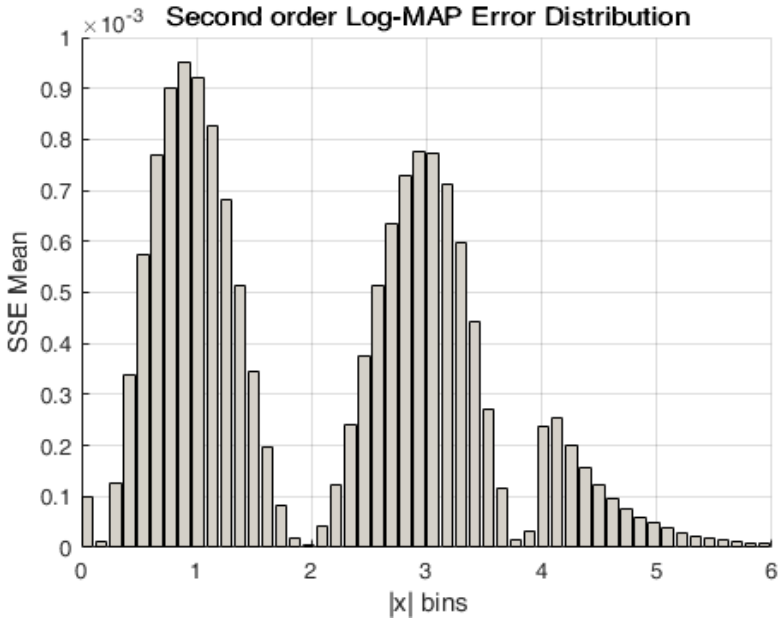


Figure 3-16 Mean error distribution for the second order approximation

The FLM offers the best overall approximation to the exact correction function with error peaks not more than approximately 0.95e-3 SSE mean even for the lower regions of $|x|$.

Table 3-8 Goodness of fit statistics for the second order approximation with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	Measure
Sum of Squared Error (SSE)	0.1658
Mean Squared Error (MSE)	4.14E-04
Root Mean Squared Error (RMSE)	0.0203
R-Squared (R^2)	0.9882

The use of a second order polynomial fit provides for a better approximation than all linear solutions reviewed in Section 2.2.10 which features a quadratic function that employs comparators, multipliers and adders.

However, in striving for a closer fit, polynomial curve fits beyond the second order is not practical as the cost on hardware for higher order function will increase. This is especially significant in Turbo decoding as the max^* operation occurs at each state metric computation and complexity of the decoder exacerbates with increased constraint length and frame rate. Although the FLM presents a good fit to the Log-MAP correction function, resources on hardware for the computation of x^2 and $-0.392x$ can be costly as a minimum of two multipliers are required. Therefore, the FLM algorithm is omitted in future analysis and implementation but included to provide for a more comprehensive literature review.

Further discussion on the results for each suboptimal algorithm will be covered in the results and discussion chapter. Results from BER simulation for each suboptimal Log-MAP approximation algorithm further enhances conclusions from this section that with improved accuracy of the correction function, the BER performance will also be improved especially at low SNR. The next section outlines the design criteria for two new approximations and details the

development of the algorithm followed by the digital circuit design implemented in FPGA.

3.4 A Fresh Approach to the Development of Suboptimal Log-MAP Algorithm

This section details the development of two suboptimal Log-MAP approximation to the correction function. The general approach taken for both approximations are close goodness of fit statistics to the ideal Log-MAP with special attention given to smaller values of $|x|$ within the correction function. The BER simulation results are presented in the following chapter.

The development of the algorithm will also focus on hardware implementation and to maximise the usage of highly optimised DSP modules in XSG for good hardware performance. Where the usage of optimised DSP modules are not possible, hardware resources will be optimised for low complexity and low latency.

3.4.1 Design Criteria for an Optimal Approximation to the Correction Function

Ultimately, a good suboptimal Log-MAP algorithm is one that achieves the closest BER performance to the exact Log-MAP Turbo decoder which has been shown to possess best overall BER performance. Specifically, this can be achieved by approximating the correction function as closely as possible to the exact Log-MAP by means of close goodness of fit (GOF) measure against the exact correction function. The accuracy of the approximation to the correction function is especially important at low SNRs as $|x|$ will most likely to take

smaller values where the correction function is most effective for these areas and influences the computation of the Jacobian logarithm.

It is emphasised from the results presented in previous sections that this is particularly important for low SNR channel conditions where x will take on lower values across all state metric computation and LLR calculations. Hence, careful attention to a close goodness of fit for the lower regions of x must be given for a good approximation.

However, a close goodness of fit alone for the suboptimal Log-MAP does not guarantee good performance in practice. The suboptimal Log-MAP algorithm will also need to cater to optimal hardware implementation. As previously discussed, the hardware implementation should be with low complexity, low latency, and low power consumption. A direct approach to this would be to use XSG blocks and High Level Synthesis tools within Xilinx to map blocks onto highly optimised DSP modules in Xilinx FPGAs.

In this section however, the key focus is to first develop key design criteria for the approximation equations with consideration to hardware implementation. The key design criteria are summarised as follows:

- Close GOF statistics against the exact correction function with the lowest SSE and R-squared measure close to 1 as compared to existing solutions in literature
- Good GOF to the exact Log-MAP in the lower regions of x in the max^* operator for close to exact Log-MAP BER performance at low SNR
- Good overall BER performance that adheres closely to the exact Log-MAP solution which has been shown to possess the best BER performance
- Hardware design with low complexity, low power, and low latency to meet the demands of future 5G solutions

With these design criteria outlined, two approximations, i.e. the Hybrid Log-MAP (HLM) and Best Fit Linear (BFL) are proposed.

3.4.2 The Hybrid Log-MAP (HLM) Approximation

The Hybrid Log-MAP algorithm is inspired by the observation that the approximation for small argument of $|x|$ is crucial for channel conditions with low SNR based on the work done throughout Section 3.2 and 3.3. Coupled with the inspiration from [30] where the MLM algorithm utilises simple right shifts to achieve division by two to achieve the multi-step approximation.

However, it is also observed that the MLM algorithm requires a relatively high number of right shifts for larger values of $|x|$, requiring a wide barrel shifter as reported in [30] which will cause latency issues especially for higher values of $|x|$ as more number of division is required to return a value to approximate the Jacobian logarithm or max^* operator. Furthermore, it was also observed that the MLM approximation for smaller values of $|x|$ does not give a desirable goodness of fit to cater to channel conditions with low SNR as compared to LC.

Figure 3-17 illustrates the design process for the Hybrid Log-MAP (HLM) approximation to the correction function.

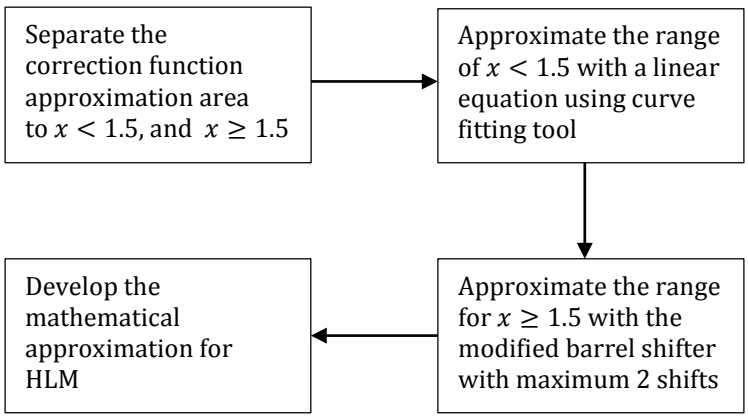


Figure 3-17 The Hybrid Log-MAP (HLM) approximation design flow

In order to approximate the Log-MAP correction function with acceptable accuracy for small regions of $|x|$, the Hybrid Log-MAP (HLM) separates the correction function to two regions, i.e. $x \leq 1.5$, and $x > 1.5$. The value 1.5 was chosen as the separation point of the piecewise function for $f_{HLM}(|x|)$ as the decimal value 1.5 can be stored as a constant on hardware with only one fractional binary bit and thus reducing the comparator circuit complexity and heighten the operation speed to separate the computation between the two regions. Furthermore, it is also observed that the value of 1.5 is suitable to isolate the lower region of $|x|$ where the correction function decays exponentially to suit a simple linear approximation. A linear approximation for $|x| < 1.5$ of the following form in EQ. 3-7 has the advantage of simple hardware implementation using only a single multiplier and add operation:

$$f_{HLM}(|x|) = c - mx, \qquad for\ x \leq 1.5 \qquad \text{EQ. 3-7}$$

Next, the region for $|x| \leq 1.5$ is approximated with a linear equation using the curve fitting tool (`fit`) in MATLAB as shown in Figure 3-18:

```
%% HLM Curve Fit Experiment

x=0:0.01:1.5;
lenx=length(x);

%% Exact correction term
y=log(1+exp(-x));

fitresult = fit(x',y','poly1')
ci95 = confint(fitresult,0.95) % verify 95% confidence bound calculated in fitresult
ci96 = confint(fitresult,0.96)
ci97 = confint(fitresult,0.97)
ci98 = confint(fitresult,0.98)
ci99 = confint(fitresult,0.99)
ci99_99 = confint(fitresult,0.9999)
```

Figure 3-18 Linear fit MATLAB script and confidence bounds computation

The fit results returned the following fit for a polynomial order 1 or linear fit setting in EQ. 3-8:

$$f_{HLM}(|x|) = 0.6512 - 0.3251x, \quad for\ x \leq 1.5 \qquad \text{EQ. 3-8}$$

The confidence bounds define the lower and upper values for the confidence interval. The confidence interval indicates the probability (often set to 95% as default or up to 99.99%) that the *true best fit* lies within the confidence interval. The confidence bound and interval for m, and c calculated using MATLAB are tabulated in Table 3-9 as follows:

Table 3-9 Confidence bounds and interval for 95% to 99.99% for c and m in EQ. 3-7

Confidence Level	**95%**	**96%**	**97%**	**98%**	**99%**	**99.99%**
Parameter	Bounds	Bounds	Bounds	Bounds	Bounds	Bounds
m = -0.3251	(-0.3319, -0.3183)	(-0.3323, -0.3180)	(-0.3327, -0.3176)	(-0.3332, -0.3170)	(-0.3341, -0.3162)	(-0.3389, -0.3114)
c = 0.6512	(0.6453, 0.6571)	(0.6451, 0.6574)	(0.6447, 0.6578)	(0.6442, 0.6583)	(0.6435, 0.6590)	(0.6393, 0.6632)

In reference to Table 3-9, for various confidence intervals, it can be concluded that the generated values for m and c in EQ. 3-7 lies within the confidence bounds with up to 99.99% confidence interval that are very narrow for m, [-0.3114 - (-0.3389) = 0.0275 interval], and c (0.0239 interval). As such the parameters for both m and c are acceptable and will be used as final values for the linear region.

The remaining area for approximation are for the regions of $|x| > 1.5$. For this region, the exact correction function is approximated by the use of a barrel shifter to complete the HLM as follows in EQ. 3-9 and illustrated in Figure 3-19:

$$f_{HLM}(|x|) = \begin{cases} 0.6512 - 0.3251x, & for\ x \leq 1.5 \\ 0.1635 \cdot 2^{-\lfloor 0.5x \rfloor}, & otherwise \end{cases} \qquad \text{EQ. 3-9}$$

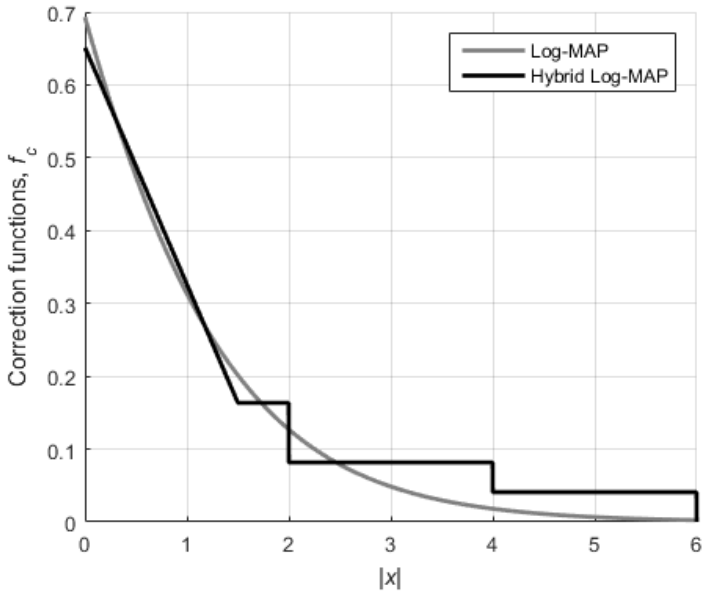


Figure 3-19 HLM approximation against the exact correction function

The regions for $|x| > 1.5$ consists of a constant i.e. 0.1635 being divided by $2^{\lfloor 0.5x \rfloor}$. The value 0.1635 is chosen to provide continuity for the linear region of $|x| \leq 1.5$ where $f_{HLM}(|x| = 1.5) = 0.1635$ based on EQ. 3-9. With the constant value beginning at the end of the linear approximation, the HLM can provide for a better fit at the separation between the two regions. Subsequently, the constant of 0.1635 is divided by two based on the value of $\lfloor 0.5x \rfloor$ where $0.5x$ is performed by a simple single right shift followed by a floor or truncation operator, $\lfloor \quad \rfloor$ which is easy, and simple to implement on hardware. The effect of dividing the constant 0.1635 with $2^{\lfloor 0.5x \rfloor}$ allows for only two maximum shifts to sufficiently approximate the correction function. By limiting the number of shifts to two, the HLM is able to return a value for the approximated correction function with a maximum of 3 division (including $\lfloor 0.5x \rfloor$) by 2 for the barrel shift register. With the exponent of $\lfloor 0.5x \rfloor$, the step size is also widened for $|x| > 1.5$. Although, the HLM loses the accuracy advantage, but it gains hardware simplicity. Furthermore, coupled by the conclusion that with higher SNR, the distribution of $|x|$ falls on relatively larger value where the correction function decays exponentially to 0, the HLM strategically approximates the exact

correction function where the effects matters the most in lower regions of $|x|$ as shown in Figure 3-19.

In order to verify the effectiveness of the HLM approximation for regions of $|x| \leq 1.5$, the GOF statistics for the range of $0 \leq |x| \leq 1.5$ is given in Table 3-10.

Table 3-10 Goodness of fit statistics for linear range of HLM, with $\Delta x = 0.01$, $0 \leq x \leq 1.5$ against other approximations

Goodness of Fit Test	HLM	CLM	LLM	MLM	LC	FLM
Sum of Squared Error (SSE)	0.0506	4.3784	1.9702	1.5133	0.2697	0.0833
Mean Squared Error (MSE)	3.3498×10^{-4}	0.029	0.013	0.01	0.0018	5.5144×10^{-4}
Root Mean Squared Error (RMSE)	0.0183	0.1703	0.1142	0.1001	0.0423	0.0235
R-Squared (R^2)	0.9836	-0.42	0.361	0.5092	0.9125	0.973

It is shown that the minimum SSE recorded for the range of $|x| \leq 1.5$ belongs to the HLM and is expected to provide better BER performance for low values of $|x| \leq 1.5$. This is also supported by the R-squared value for HLM with a measure of 0.9836 which gives an improvement of approximately 7.79% over the LC, and an improvement of 1.09% over the more complex FLM approximation in terms of accuracy for the regions of $|x| \leq 1.5$.

The overall mean error distribution of the HLM against the Log-MAP correction function is given in Figure 3-20 followed by the goodness of fit parameter for the overall HLM algorithm in Table 3-11.

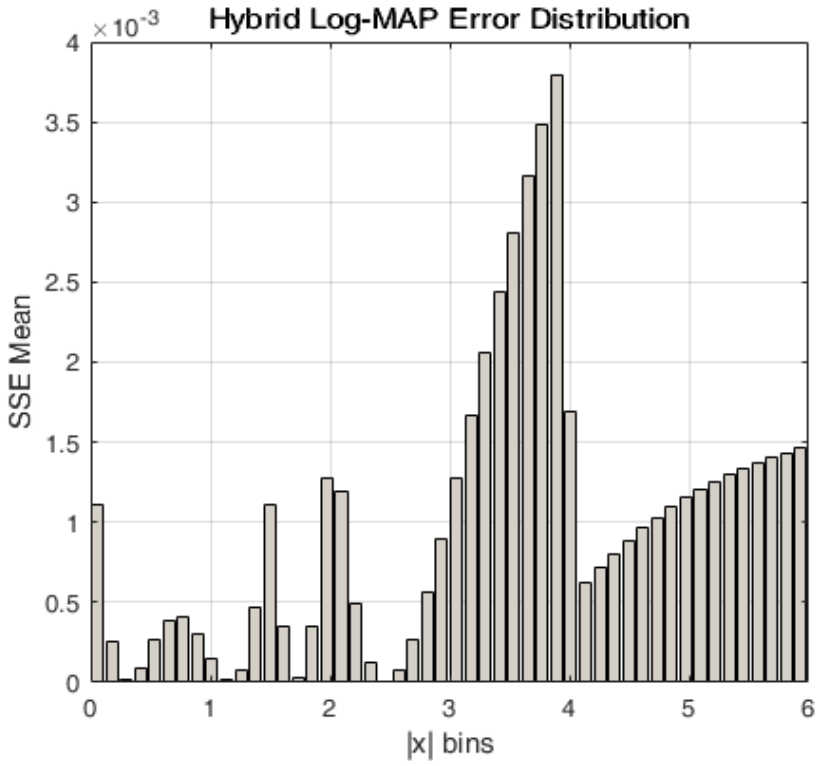


Figure 3-20 Mean error distribution for the HLM approximation

The mean error distribution for HLM indicates excellent approximation accuracy lower regions of $|x|$ that outdo the LC approximation with no mean SSE error peaks exceeding 1.5×10^{-3} for the region of $|x| < 3$. By contrast to the LC in Figure 3-15, error peaks within this range recorded a peak of up to 7×10^{-3} for regions surrounding $|x| = 0$.

For regions of $|x| > 3$ however, the HLM recorded a large discrepancy against the exact correction function. Strategically, however, as demonstrated in Section 3.2, the probability of $|x|$ falling into these regions at very low SNR is relatively low and that the region covers an area where the correction function magnitude is not as high as the lower regions. Due to this, a trade-off between accuracy and BER performance is observed for the larger regions of $|x|$ with the HLM but this is done strategically where the correction function magnitude is not as effective. Table 3-11 shows the GOF statistics for the HLM against all

other linear approximations for the region of $0 \leq |x| < 4$ for further comparison.

Table 3-11 Goodness of fit statistics for the overall range of $|x|$ with $\Delta x = 0.01$, $0 \leq x < 4$ for HLM against other approximations

Goodness of Fit Test	HLM	CLM	LLM	MLM	LC	FLM
Sum of Squared Error (SSE)	0.3882	6.5098	4.1015	1.9782	0.4368	0.1658
Mean Squared Error (MSE)	9.68E-04	0.0162	0.0102	0.0049	0.0011	4.14E-04
Root Mean Squared Error (RMSE)	0.0311	0.1274	0.1011	0.0702	0.033	0.0203
R-Squared (R^2)	0.9723	0.5351	0.7071	0.8587	0.9688	0.9882

The HLM algorithm is strategically designed to cater to accuracy for lower values of x and hardware efficiency for larger values where the correction function magnitude is not dominant. Collectively, the HLM placed second in terms of overall goodness of fit parameters for the region of $0 \leq |x| < 4$. However, comparatively the HLM approximates the correction function more accurately for $|x| \leq 1.5$ compared to other approximations. The full BER simulation results for the HLM algorithm will be presented in Chapter 4.

3.4.3 The Best Fit Linear (BFL) Approximation

The Best Fit Linear (BFL) approximation is inspired by the ability of a linear fit to approximate the correction function accurately particularly at the lower regions of $|x|$ as demonstrated in the HLM approximation.

The BFL approximation maintains a two region approximation as opposed to the work done in [45] where the PL method approximates the correction function with 5 linear regions. Although superior in terms of accuracy, the hardware latency for the PL approximation is high due to the 5 regions

implemented through a series of comparator before the corresponding multiply and addition operation takes place.

A better approach would be to exploit the observation illustrated in Section 3.2, i.e. the approximation accuracy for lower regions of $|x|$ should be as accurate as possible to the exact correction function where it is most dominant as the distribution of $|x|$ would more likely take lower values at low SNRs. A lower accuracy approximation can be afforded as a trade-off for hardware simplicity for higher values of $|x|$ making the approximation more hardware friendly.

Returning to the matter at hand, the BFL approximates the correction function with only two regions of linear fit to minimize latency in the use of comparators to identify the right region of $|x|$ in computing the max^* operator. The search for initial optimal points for the BFL approximation are illustrated in Figure 3-21.

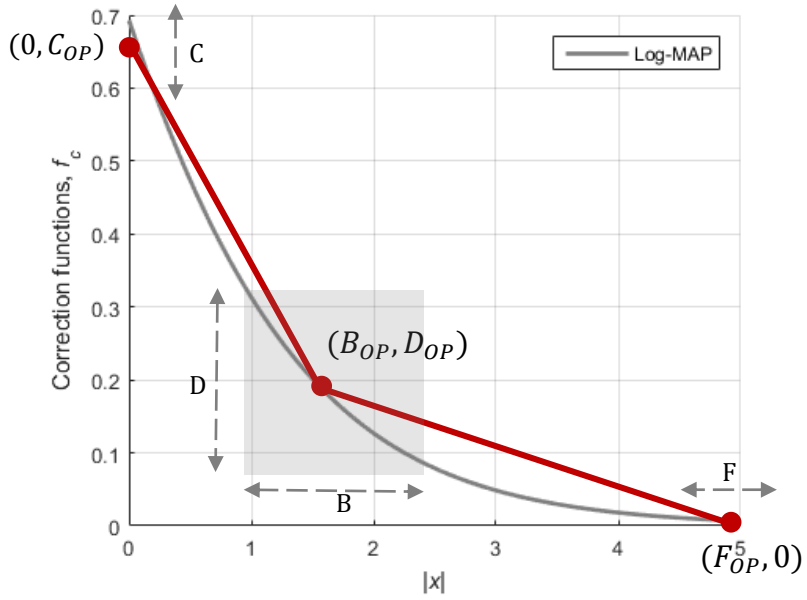


Figure 3-21 Search areas for BFL optimal values

The BFL is the result of a search of three optimal points i.e. $\{(0, C_{OP}), (B_{OP}, D_{OP}), (F_{OP}, 0)\}$ to form two linear graphs in order to approximate the Log-MAP correction function as illustrated in Figure 3-21. The

search algorithm for these points is computed within a range of values for variables C, B, D, and F as indicated in Figure 3-21.

Each variable (C, B, D, and F) range is set to approximate the rough area of where the optimal points would reside and would include a START and STOP region search point e.g. C_{START}, and C_{STOP}. Each START and STOP area would be equally spaced with a resolution of δ. Following this, the goodness of fit for all permutation of all combinations of C(START: δ :STOP), B(START: δ :STOP), D(START: δ :STOP), and F(START: δ :STOP) will be computed and the best combination of (C,B,D, and F) with the lowest SSE will be identified. Next, the resolution δ will be refined to smaller values and a smaller gap between the START and STOP values for each variable will be set to narrow down the search area. Care is taken so that the optimal value for each iteration falls between each START and STOP value. This ensures that the optimal values are not out of bounds of the search area. The process is iterated until the resolution was set to be δ =0.00001 to ensure that the optimal values found are accurate and that the exhaustive search would still be practical. The iterative computation is unfortunately necessary as there are 4 optimal values for computation, each with differing range of optimal values, along with different δ value set. The computation for optimal values will take all permutation of

$(C_{START}: \delta: C_{STOP}, B_{START}: \delta: B_{STOP}, D_{START}: \delta: D_{STOP}, F_{START}: \delta: F_{STOP})$

to perform an exhaustive search of the combination (C, B, D, and F) that gives the lowest SSE for further refinement. The value for $\delta = 0.00001$ was selected as the end of iteration as it is an accurate enough precision to represent the optimal values. Such accuracy would also allow for floating point computation should the BFL be adopted for floating point processing. Throughout the iterative process, the resolution to compute SSE is kept to $\Delta x = 0.01$ which is consistent with the SSE computation method of GOF statistics in this research.

It was also found that the value of Δx has no bearing to the final optimal coefficient output for the BFL. This iterative process is summarised in the form of a flow chart in Figure 3-22.

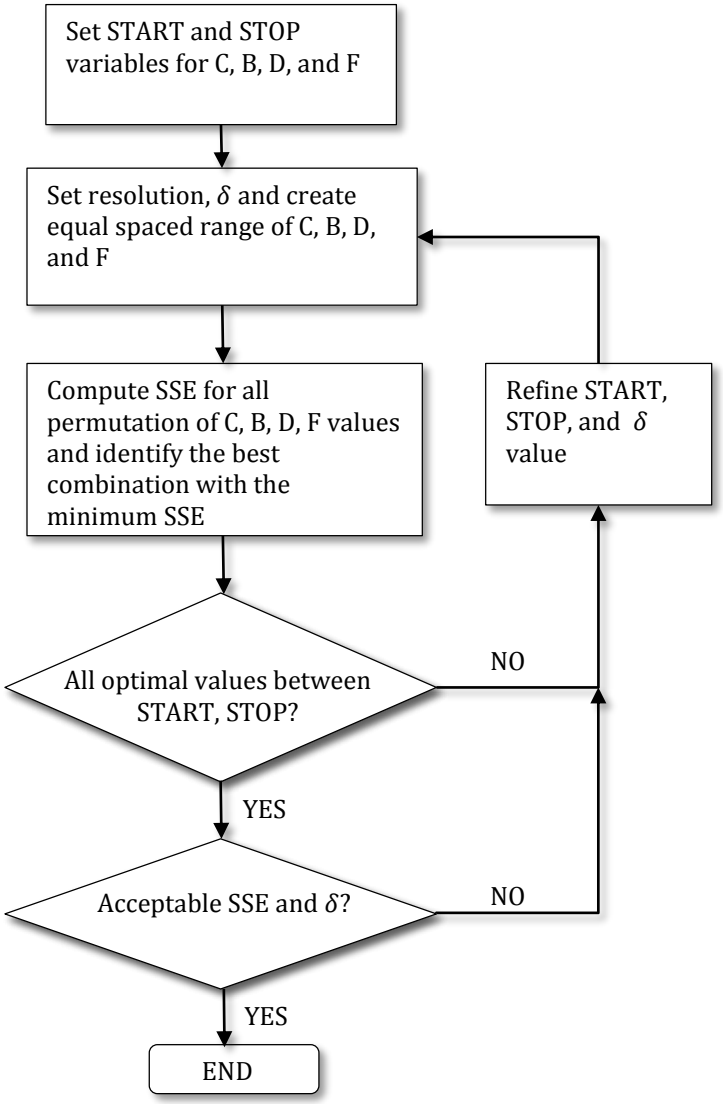


Figure 3-22 Iterative search for $(\mathbf{0}, \boldsymbol{C_{OP}})$, $(\boldsymbol{B_{OP}}, \boldsymbol{D_{OP}})$, and $(\boldsymbol{F_{OP}}, \mathbf{0})$

The iteration process is detailed step by step in Table 3-12 to reach the optimal values as illustrated in Figure 3-23

Table 3-12 Iterative search parameters to arrive to final optimal values, $\Delta x = 0.01$

Iteration	δ	$C_{OP}(C_{START}, C_{STOP})$	$B_{OP}(B_{START}, B_{STOP})$	$D_{OP}(D_{START}, D_{STOP})$	$F_{OP}(F_{START}, F_{STOP})$
1	0.1	0.7 (0.6,0.8)	1.3 (1,1.6)	0.2(0.1,0.3)	3.6(3.5,4)
2	0.05	0.65(0.6,0.8)	1.35(1,1.6)	0.2(0.1,0.3)	3.55(3.5,4)
3	0.01	0.66(0.55,0.75)	1.42(1.25,1.45)	0.18(0.1,0.3)	3.68(3.45, 3.7)
4	0.005	0.655(0.55,0.75)	1.425 (1.25,1.45)	0.18(0.1,0.3)	3.675(3.45, 3.7)
5	0.001	0.655(0.645,0.665)	1.434(1.415,1.435)	0.178(0.17,0.19)	3.687(3.665,3.7)
6	0.0005	0.6545(0.645,0.665)	1.435(1.415,1.440)	0.178(0.17,0.19)	3.686(3.665,3.7)
7	0.0001	0.6547(0.6535,0.6555)	1.4346(1.434,1.436)	0.178(0.177,0.179)	3.6864(3.685,3.688)
8	0.00005	0.65475(0.6535,0.6555)	1.4348(1.434,1.436)	0.17795(0.177,0.179)	3.6867(3.685,3.688)
9	0.00001	0.65473(0.6546,0.6548)	1.4347(1.4346,1.4349)	0.17797(0.1778,0.1781)	3.6865(3.6864,3.6867)

The $(C_{OP}, B_{OP}, D_{OP}, F_{OP})$ are the combination with the lowest SSE within the START and STOP range of each (C,B,D,F) variable with respective resolution, δ in each iteration. At the first iteration, a large $\delta = 0.1$ was set along with a large area for each variable START and STOP range to search for the optimal combination of $(C_{OP}, B_{OP}, D_{OP}, F_{OP})$ for the particular iteration. The second iteration is set at a smaller $\delta = 0.05$ with the same START and STOP search area to better identify a narrower range in the third iteration. It is not possible to directly identify a large area with a small $\delta = 0.00001$ as such a setting will produce a too large permutation matrix of all (C,B,D,F) with following length of combination L_T in EQ. 3-10:

$$L_T = {}^{L_C}C_1 \times {}^{L_B}C_1 \times {}^{L_D}C_1 \times {}^{L_F}C_1 \qquad \text{EQ. 3-10}$$

where L_C is the total length of the search area for variable C from C_{START}: δ: C_{STOP}. Similar definition applies for L_B, L_D, and L_F. To give an idea of the length of combination, the smallest $\delta = 0.00001$ is taken for all variable range in iteration 1. The total number of combination for MATLAB to compute an exhaustive search for the minimum SSE combination would be L_T =1.2002× 10^{18} which is not possible for practical computation in MATLAB. Hence, it is necessary to systematically narrow down the search area by resetting and adjusting the δ and START, STOP values for each variable. It is also for this reason that $\delta = 0.00001$ is selected as the smallest δ that is practical for the search. The value of $\delta = 0.00001$ is more than enough for the study of fixed point implementation as the binary representation for $\delta = 0.00001$ is 0.0000000000000000101 in binary representation with 19 radix point.

Where the optimal values are identified at exactly the START or STOP value, the iteration is repeated with a larger START and STOP range to ensure that the optimal values are not out of bounds. The MATLAB script written for this iterative search can be referred in Appendix E. The resulting BFL algorithm for $\delta = 0.00001$ is given in EQ. 3-11 using the final optimal values at iteration 9 and illustrated in Figure 3-23.

$$f_{BFL}(x) = \begin{cases} 0.6547 - 0.3323x, & x \leq 1.4347 \\ 0.2914 - 0.079x, & 1.4347 < x < 3.6865 \\ 0, & x \geq 3.6865 \end{cases} \qquad \text{EQ. 3-11}$$

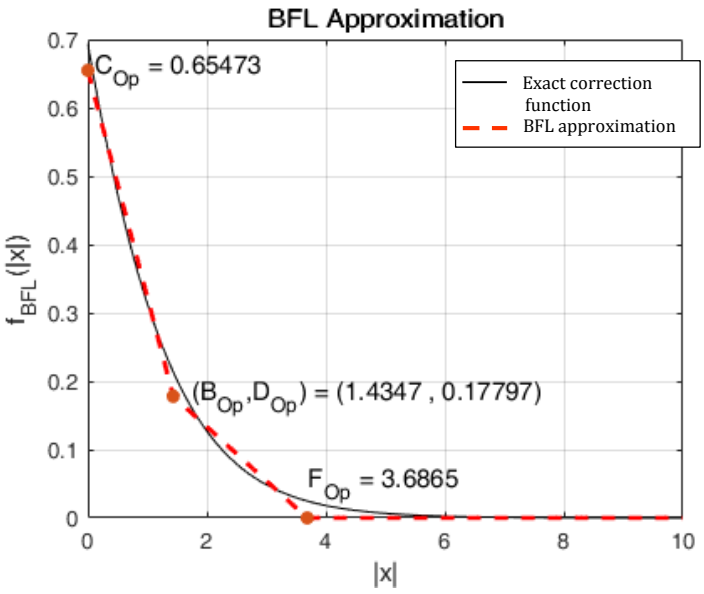


Figure 3-23 BFL approximation against the exact correction function

The main advantage with this approach is the ability to determine the optimal value for B_{OP} to separate between the two regions for approximation. Verification with the curve fitting tool in MATLAB for the two regions yields the exact same results as shown:

```
%% BFL verification Curve Fit Experiment for range x<=1.4347

x=0:0.01:1.4347;
lenx=length(x);

%% Exact correction term
y=log(1+exp(-x));

fitresult = fit(x',y','poly1')
ci95 = confint(fitresult,0.95) % verify 95% confidence bound calculated in fitresult
ci96 = confint(fitresult,0.96)
ci97 = confint(fitresult,0.97)
ci98 = confint(fitresult,0.98)
ci99 = confint(fitresult,0.99)
ci99_99 = confint(fitresult,0.9999)
```

```
fitresult =

     Linear model Poly1:
     fitresult(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =     -0.3323   (-0.339, -0.3256)
       p2 =      0.6547   (0.6492, 0.6603)
```

Figure 3-24 BFL curve fit verification for $x \leq 1.4347$

The results of the confidence bound and interval are compiled in Table 3-13.

Table 3-13 Confidence bounds and interval for 95% to 99.99% for c and m for BFL $x \leq 1.4347$

Confidence Level	95%	96%	97%	98%	99%	99.99%
Parameter	Bounds	Bounds	Bounds	Bounds	Bounds	Bounds
p_1 = -0.3323	(-0.3390, -0.3256)	(-0.3394, -0.3252)	(-0.3398, -0.3248)	(-0.3403, -0.3243)	(-0.3412, -0.3234)	(-0.3459, -0.3187)
p_2 = 0.6547	(0.6492, 0.6603)	(0.6489, 0.6606)	(0.6486, 0.6609)	(0.6481, 0.6614)	(0.6474, 0.6621)	(0.6435, 0.6660)

Based on the confidence bounds computed from 95% to 99.99% confidence level, the values of m, and c initially calculated via the exhaustive search for minimum SSE method is accepted as it consistently lies within the bounds for each confidence interval. The same process is repeated for range $1.4347 < x < 3.6865$.

```
%% BFL verification Curve Fit Experiment for range 1.4347<x<3.6865

x=1.4347:0.01:3.6865;
lenx=length(x);

%% Exact correction term
y=log(1+exp(-x));

fitresult = fit(x',y','poly1')
ci95 = confint(fitresult,0.95) % verify 95% confidence bound calculated in fitresult
ci96 = confint(fitresult,0.96)
ci97 = confint(fitresult,0.97)
ci98 = confint(fitresult,0.98)
ci99 = confint(fitresult,0.99)
ci99_99 = confint(fitresult,0.9999)
```

```
fitresult =

     Linear model Poly1:
     fitresult(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =    -0.07912  (-0.08185, -0.07639)
       p2 =      0.2917  (0.2845, 0.2989)
```

Figure 3-25 BFL curve fit verification for $1.4347 < x < 3.6865$

It is noted that the respective m and c value computed from the exhaustive search method in EQ. 3-13 differs from the verification via MATLAB for this region. However, if the initial m and c value computed from the exhaustive search is acceptable and consistently within the confidence bounds of 95% up to 99.99%. The values are still acceptable within 99.99% level of certainty.

Similarly, the results of the confidence bound and interval are compiled in Table 3-14

Table 3-14 Confidence bounds and interval for 95% to 99.99% for c and m for BFL $x \leq 1.4347$

Confidence Level	**95%**	**96%**	**97%**	**98%**	**99%**	**99.99%**
Parameter	Bounds	Bounds	Bounds	Bounds	Bounds	Bounds
p_1 = -0.07912	(-0.0818, -0.0764)	(-0.0820, -0.0763)	(-0.0821, -0.0761)	(-0.0824, -0.0759)	(-0.0827, -0.0755)	(-0.0846, -0.0736)
p_2 = 0.2917	(0.2845, 0.2989)	(0.2842, 0.2993)	(0.2837, 0.2997)	(0.2831, 0.3003)	(0.2822, 0.3012)	(0.2772, 0.3062)

Since the values from the exhaustive search ($m = -0.079$, $c = 0.2914$) lies within the confidence bounds at 99.99% level of certainty, the computed values are acceptable.

The mean error distribution against the Log-MAP correction function and goodness of fit statistics against other algorithm are given in Figure 3-26 and Table 3-15 respectively.

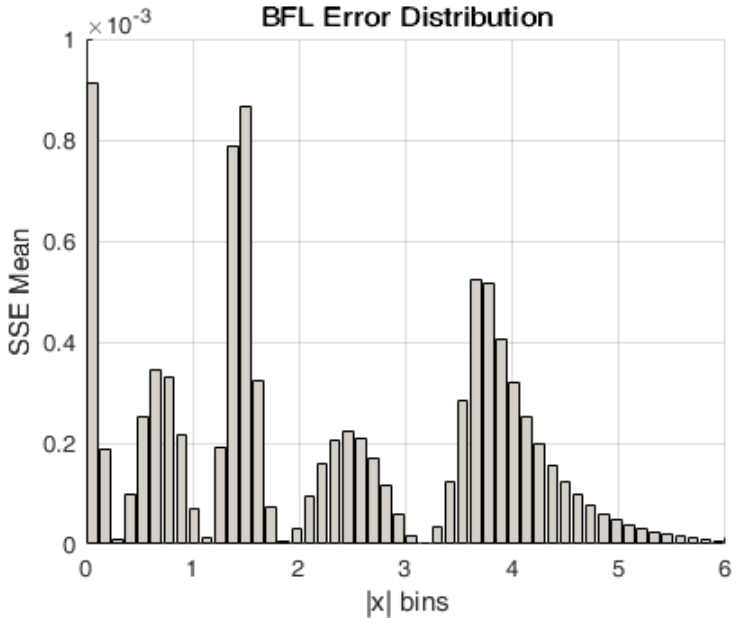


Figure 3-26 Mean SSE error distribution for the BFL approximation

The mean SSE error distribution for the BFL approximation is with one of the lowest recorded error peaks (less than 1×10^{-3}) with low discrepancy to the

exact correction function among all approximations analysed. This is confirmed with the GOF statistics computed in Table 3-15:

Table 3-15 Goodness of fit statistics for the BFL approximation with $\Delta x = 0.01$ for $0 \leq x \leq 4$ based on EQ. 3-11

Goodness of Fit Test	BFL	HLM	CLM	LLM	MLM	LC	FLM
Sum of Squared Error (SSE)	0.0958	0.3882	6.5098	4.1015	1.9782	0.4368	0.1658
Mean Squared Error (MSE)	2.39E-04	9.68E-04	0.0162	0.0102	0.0049	0.0011	4.14E-04
Root Mean Squared Error (RMSE)	0.0155	0.0311	0.1274	0.1011	0.0702	0.033	0.0203
R-Squared (R^2)	0.9932	0.9723	0.5351	0.7071	0.8587	0.9688	0.9882

It is clear that the BFL offers the lowest SSE measure at 0.0958 with R-squared value of 0.9932. The encouraging GOF statistics implies that the BFL BER performance will adhere to the exact correction function more closely than other approximations.

3.4.3.1 Family of BFL for Fixed Point Processing

For the purpose of discussion, the BFL can be generalised as follows in EQ. 3-12:

$$f_{BFL}(x) = \begin{cases} c_1 + m_1 x, & x \leq x_1 \\ c_2 + m_2 x, & x_1 < x < x_2 \\ 0, & x \geq x_2 \end{cases} \quad \text{EQ. 3-12}$$

A family of fixed point variations of the BFL based on possible fixed point representation radix values, N to be implemented on hardware are also calculated and proposed as follows:

Table 3-16 Parameter values of BFL_N for EQ. 3-12 with varying N rounded to four decimal places

Algorithm	N	Parameter values					
		c_1	m_1	c_2	m_2	x_1	x_2
f_{BFL4}	4	0.6250	-0.3125	0.2500	-0.0625	1.3750	3.6250
f_{BFL6}	6	0.6406	-0.3281	0.2813	-0.0781	1.4219	3.6719
f_{BFL8}	8	0.6523	-0.3320	0.2891	-0.0781	1.4336	3.6836

This study is done to investigate the effects of fixed point implementation on quantization of each parameter. However, for ease of discussion, the quantisation effects in Table 3-16 are rounded to the nearest four decimal places. The exact quantization effect to each parameter is discussed in Section 3.5 when the BFL approximations are designed on hardware. Each of the m_1 and m_2 values for $N = 4$ to 8 are proposed based on the results in EQ. 3-11. All m_1 and m_2 values are converted to rational values based on the radix N representation.

Take for instance the $N = 4$ radix for f_{BFL4}, the respective values for $c_1, m_1, c_2, m_2, x_1, x_2$ are calculated by converting the exact decimal values in EQ. 3-11 to the nearest representation for radix 4 fixed point. This can be easily verified using the `num2fixpt` function in MATLAB as shown in Figure 3-27.

```
%% Fixed point quantization- rounded to 4 decimal values
N=4; % Change accordingly to radix N
num2fixpt(0.6547, sfix(20), 2^-N) %c1
num2fixpt(0.3323, sfix(20), 2^-N) %m1
num2fixpt(0.2914, sfix(20), 2^-N) %c2
num2fixpt(0.0790, sfix(20), 2^-N) %m2
num2fixpt(1.4347, sfix(20), 2^-N) %x1
num2fixpt(3.6865, sfix(20), 2^-N) %x2
```

Figure 3-27 Fixed point quantization effects

The effects of quantization due to fixed point radix results in the respective values in Table 3-16 rounded to the nearest four decimal places. The exact

values for implementation on fixed point shall be presented in the next section for hardware design.

The proposed f_{BFL4} to f_{BFL8} are designed with close GOF statistics suitable of the radix value N representation as well as to exploit bit shifts operation for computation ease for BFLN as shown in Table 3-16. This approach firstly approximates the gradient coefficients, m_1 and m_2, such that the denominators are set to be 2^N. Again, this was designed to exploit the fact that a division by 2 is a simple left-bit-shift operation in hardware and N denotes the maximum bit shifts required in the respective approximations. The task of converting the N radix value into a ratio can be easily verified in MATLAB using the `rat` function.

Subsequently. the numerators for m_1 and m_2 are selected as a subset of integers, $\mathbb{Z}$ to form rational constant multiplication [110],[111]–[114] with x for each linear approximation regions of $x \leq x_1$ and $x_1 < x < x_2$. This arrangement allows for further simplification of f_{BFL4} and f_{BFL6} as shown in Table 3-17; eliminating the need for any multiplication operation.

For future discussion, only the cases for radix 4, 6, and 8 are discussed as these radix values are sufficient to illustrate the concept of application and are considered easier to integrate for binary computation. Radix of $N < 4$ are not calculated as the most common radix implementation are $N = 4$ and above [19], [115]–[117].

Table 3-17 Numerator adjustments for further optimisation to BFL_N rounded to four points decimal

Algorithm	*N*	Pre Numerator Adjustment	Post Numerator Adjustment
f_{BFL4}	4	$f_{BFL4}(x) = \begin{cases} 0.6250 - \left(\frac{2^2+1}{16}\right)x, & x < 1.3750 \\ 0.2500 - \left(\frac{1}{16}\right)x, & 1.3750 < x < 3.6250 \\ 0, & x \geq 3.6250 \end{cases}$	$f_{BFL4}(\lvert x \rvert) = \begin{cases} 0.6250 - \left(\frac{x}{2^2} + \frac{x}{2^4}\right), & x \leq 1.3750 \\ 0.2500 - \left(\frac{x}{2^4}\right), & 1.3750 < x < 3.6250 \\ 0, & x \geq 3.6250 \end{cases}$ EQ. 3-13
f_{BFL6}	6	$f_{BFL6}(x) = \begin{cases} 0.6406 - \left(\frac{2^4+2^2+1}{64}\right)x, & x < 1.4219 \\ 0.2813 - \left(\frac{2^2+1}{64}\right)x, & 1.4219 < x < 3.6719 \\ 0, & x \geq 3.6719 \end{cases}$	$f_{BFL6}(x) = \begin{cases} 0.6406 - \left(\frac{x}{2^2} + \frac{x}{2^4} + \frac{x}{2^6}\right), & x \leq 1.4219 \\ 0.2813 - \left(\frac{x}{2^4} + \frac{x}{2^6}\right), & 1.4219 < x < 3.6719 \\ 0, & x \geq 3.6719 \end{cases}$ EQ. 3-14
f_{BFL8}	8	$f_{BFL8}(x) = \begin{cases} 0.6523 - \left(\frac{2^6+2^4+2^2+1}{256}\right)x, & x < 1.4336 \\ 0.2891 - \left(\frac{2^4+2^2}{256}\right)x, & 1.4336 < x < 3.6836 \\ 0, & x \geq 3.6836 \end{cases}$	$f_{BFL8}(x) = \begin{cases} 0.6523 - \left(\frac{x}{2^2} + \frac{x}{2^4} + \frac{x}{2^6} + \frac{x}{2^8}\right), & x \leq 1.4336 \\ 0.2891 - \left(\frac{x}{2^4} + \frac{x}{2^6}\right), & 1.4336 < x < 3.6836 \\ 0, & x \geq 3.6836 \end{cases}$ EQ. 3-15

With further simplification done to the BFL_N algorithms, the computations are reduced to bit shifting and addition/subtraction operations only. Thereby, further optimising the implementation of BFL_N on hardware. It is also interesting to note that the addition for regions of $x_1 < x < x_2$ are sub addition results of $x \leq x_1$. This can be capitalised on hardware to reduce the number of addition operations.

A plot of the BFL_N functions against the exact Log-MAP is given in Figure 3-28, Figure 3-29, and Figure 3-30 followed by the goodness of fit statistics for the proposed radix point approximation for fixed point processing in Table 3-18.

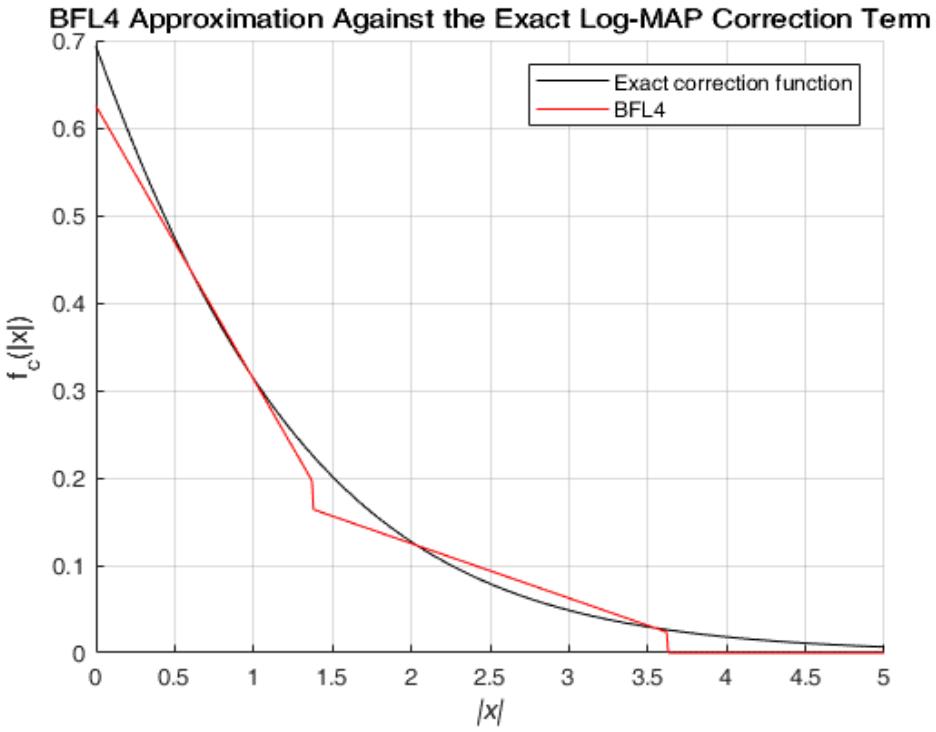


Figure 3-28 Plot of BFL_4 correction functions

Due to fixed point quantization effects, the $c_1, m_1, c_2, m_2, x_1, x_2$ values for BFL4 and BFL6 has shifted and caused inaccuracies to the approximation such as the BFL4 as illustrated in Figure 3-28. The linear fit region for smaller values of $|x|$ shows a larger discrepancy and will affect the BER performance at low SNRs.

The approximation accuracy for BFL6 in Figure 3-29 however, is better approximated to the exact correction function with minor inaccuracies from the naked eye.

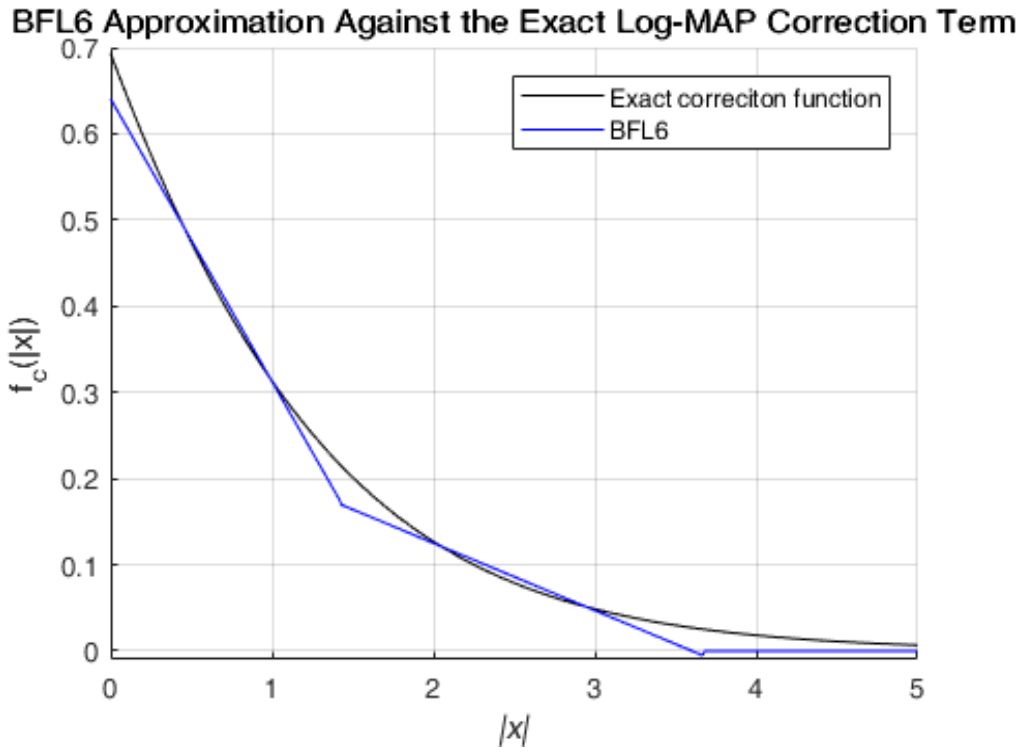


Figure 3-29 Plot of BFL_6 correction functions

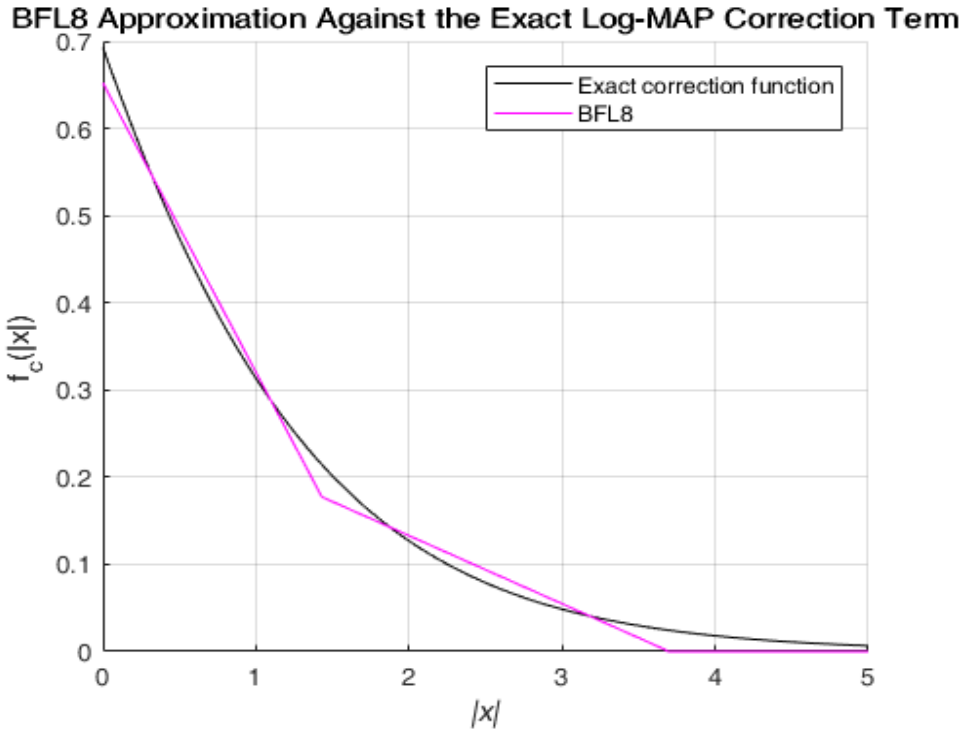


Figure 3-30 Plot of BFL_8 correction functions

The goodness of fit statistics for BFL8 offers the best performance in terms of GOF accuracy to the correction function for fixed point implementation as shown in Table 3-18.

Table 3-18 Goodness of fit statistics for BFL_N with $\Delta x = 0.01$ for $0 \leq x \leq 4$

Goodness of Fit Test	f_{BFL4}	f_{BFL6}	f_{BFL8}	f_{BFL}
Sum of Squared Error (SSE)	0.1834	0.1272	0.0966	0.0958
Mean Squared Error (MSE)	4.5739e-04	3.1726e-04	2.4096e-04	2.3898 e-04
Root Mean Squared Error (RMSE)	0.0214	0.0178	0.0155	0.0155
R-Squared (R^2)	0.9869	0.9909	0.9931	0.9932

It is expected that next to the BFL floating point implementation, the BFL8 should offer the next closest BER performance to the exact correction function. It is also interesting to note that there is only a 0.0001 R-squared difference between BFL and BFL8 and that the BER performance between them should be very close to the exact correction function.

3.5 Design of FPGA Architecture for Proposed Approximations

Based on the review of Xilinx FPGA in Chapter 2, Section 2.3, it is clear that the Virtex Ultrascale+ offers better resource availability. The choice for FPGA implementation for the HLM and BFL design shall be confined to the Virtex Ultrascale+. The following sub-sections details the FPGA architecture for HLM and BFL for floating point, BFL4, BFL6, and BFL8 implementation. The design criteria for FPGA implementation are oriented for low complexity, low latency, and low power consumption.

Efforts were made to ensure the latest available design tools and hardware platforms are employed to exploit the latest high level synthesis and

optimisation tools for Xilinx FPGA implementation. Details and parameters of the chosen Virtex Ultrascale+ device are given in Table 3-19 [96].

Table 3-19 Xilinx FPGA product specifications

Feature	Specification
Product Family	Virtex UltraScale+
Technology	16nm FinFET+
Device Name	XCVU9P
System Logic Cells (K)	2,586
CLB Flip-Flops (K)	2,364
CLB LUTs (K)	1,182
Block RAM (Mb)	75.9
DSP Slices	6,840
Speed Grade	-3 (fastest), 891 MHz
Operating Voltage (V)	0.9

The device, XCVU9P is also readily integrated with Simulink and VIVADO HLS making it an optimal choice as hardware platform for design [96], [118].

3.5.1 HLM FPGA Architecture Design

The HLM approximation is realised in Xilinx System Generator (XSG) as shown in Figure 3-31.

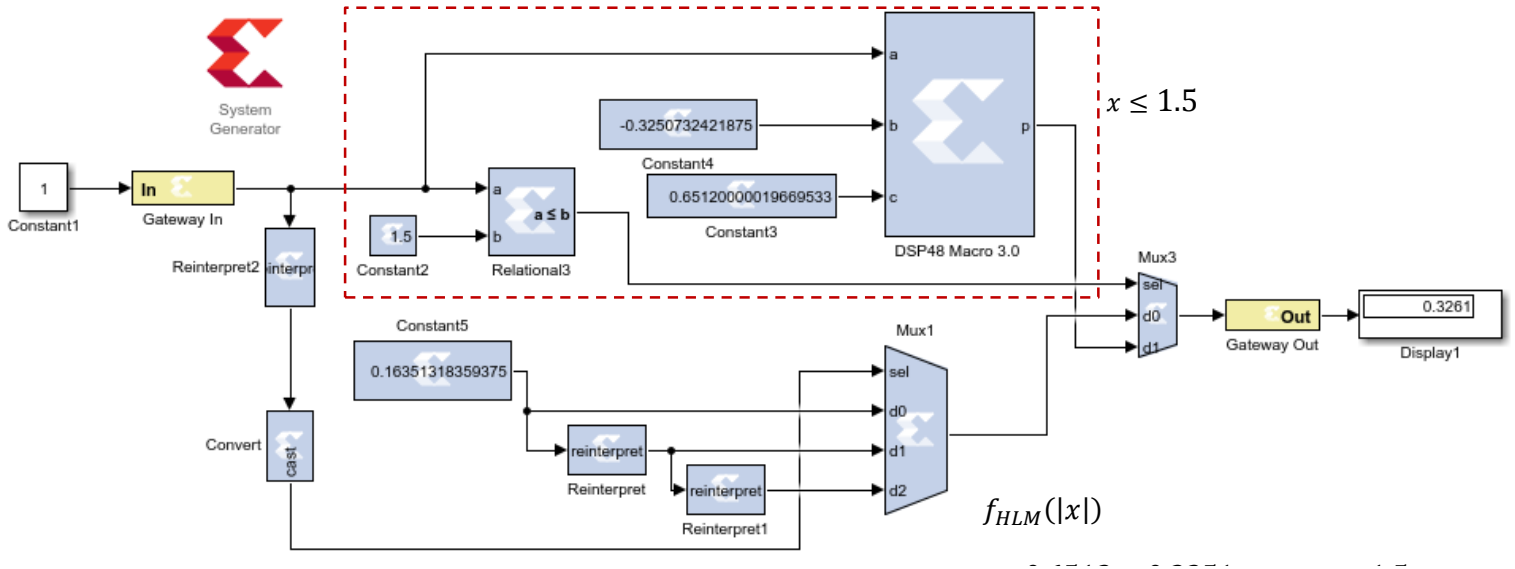


Figure 3-31 HLM approximation as implemented on XSG

.

The design of the HLM approximation block diagram is separated to two regions, $x \leq 1.5$ or otherwise according to the HLM approximation in EQ. 3-9. The highlighted region indicates the computational blocks for the region of $|x| \leq 1.5$. The list of resources used and a brief description for each block are summarised in Table 3-20 [119]

Table 3-20 Resources used for the HLM design in XSG

Label	Parameters	Function Description
System Generator	Virtex Ultrascale+ XCVU9P	FPGA device selection and compiler that generates the HDL and Vivado project file
Constant1	1	Constant source representing value of $\|x\|$ for computation by $f_{HLM}(\|x\|)$
Gateway In	Default	Converts Constant1 into a Xilinx signal
Constant2	1.5	Constant stored in Look up Table (LUT)
Relational3	Compares $x \leq 1.5$	Comparator
Constant3	0.6512	Constant stored in Look up Table (LUT)
Constant4	-0.3251	Constant stored in Look up Table (LUT)
DSP Macro 3.0	$b * a + c$	DSP48E2 slice for multiply and add operation
Reinterpret2	$0.5x$	Reinterprets fixed point value for right bit shift. No cost to hardware
Convert	$\lfloor 0.5x \rfloor$	Converts output from Reinterpret2 to sutiable input format for Mux1 selector. No cost to hardware when quantization is set to truncate
Constant5	0.1635	Constant stored in Look up Table (LUT)
Reinterpret	$0.1635/2$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret1	$0.1635/4$	Reinterprets fixed point value for right bit shift. No cost to hardware
Mux1	Select regions of x based on $\lfloor 0.5x \rfloor$	Multiplexer
Mux3	Select regions between $\|x\| \leq 1.5$ or otherwise	Multiplexer
Gateway Out	Default	Converts Xilinx signal into constant for Display1
Display1	Default	Constant viewer

The HLM approximation uses the maximum allowable fixed point radix with limits to only a single usage of DSP slice for computation to minimise quantization effects.

The HLM for $|x| > 1.5$ region implements a maximum 2 right bit shifts which is dependent on the value of $\lfloor 0.5x \rfloor$ after the Convert block. This is ensured by setting the maximum number of bits for convert by truncation to allow an input

of selection $d0$, $d1$, or $d2$ to the multiplexer that limits the choice to only the three regions of the multistep function for the HLM.

After verification of the computational function of HLM in XSG, the VHDL codes are generated and the top level schematics for the HLM are given in Figure 3-32.

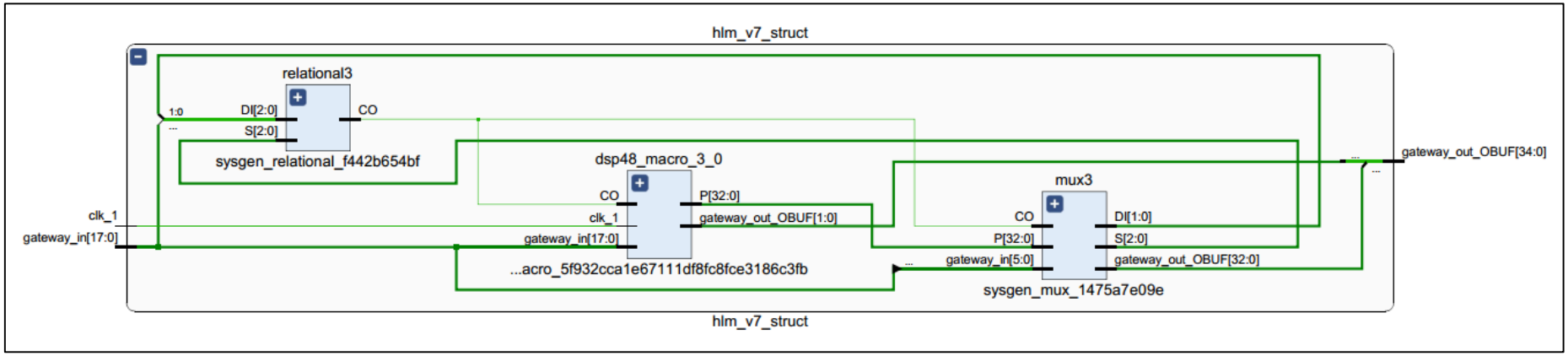


Figure 3-32 HLM top level schematic in Vivado

The top level schematics verifies the hardware implementation of the HLM in XSG as shown in Figure 3-31. The resulting logic count for the HLM only involves the relational operator, DSP48E2 slice, and multiplexer. The use of reinterpret and convert blocks (in truncation mode) does not cost any logic count to the final architecture. This differs from the proposed solution for the MLM approximation used in [30], [45] where the division by 2 is implemented using wide shift registers that divides in a sequential manner with each clock cycle. The reinterpret block offers a combinational interconnect for dedicated right bit shift in the HLM and does not incur additional logic for bit shifts. The logic block level schematics for the HLM is expanded from Figure 3-32 and shown in Figure 3-33 with fixed sets details of interconnects and logic blocks for the DSP48E2 DSP slice [93].

The only operation that is clock cycle dependent for HLM is the multiply and add operation using the DSP48E2 slice that is an existing dedicated optimised, DSP slice combining low power (0.001W), high speed (825MHz) and small size as detailed in Section 0 [93]. Further discussions on complexity, latency and power shall be discussed in results, Chapter 4.

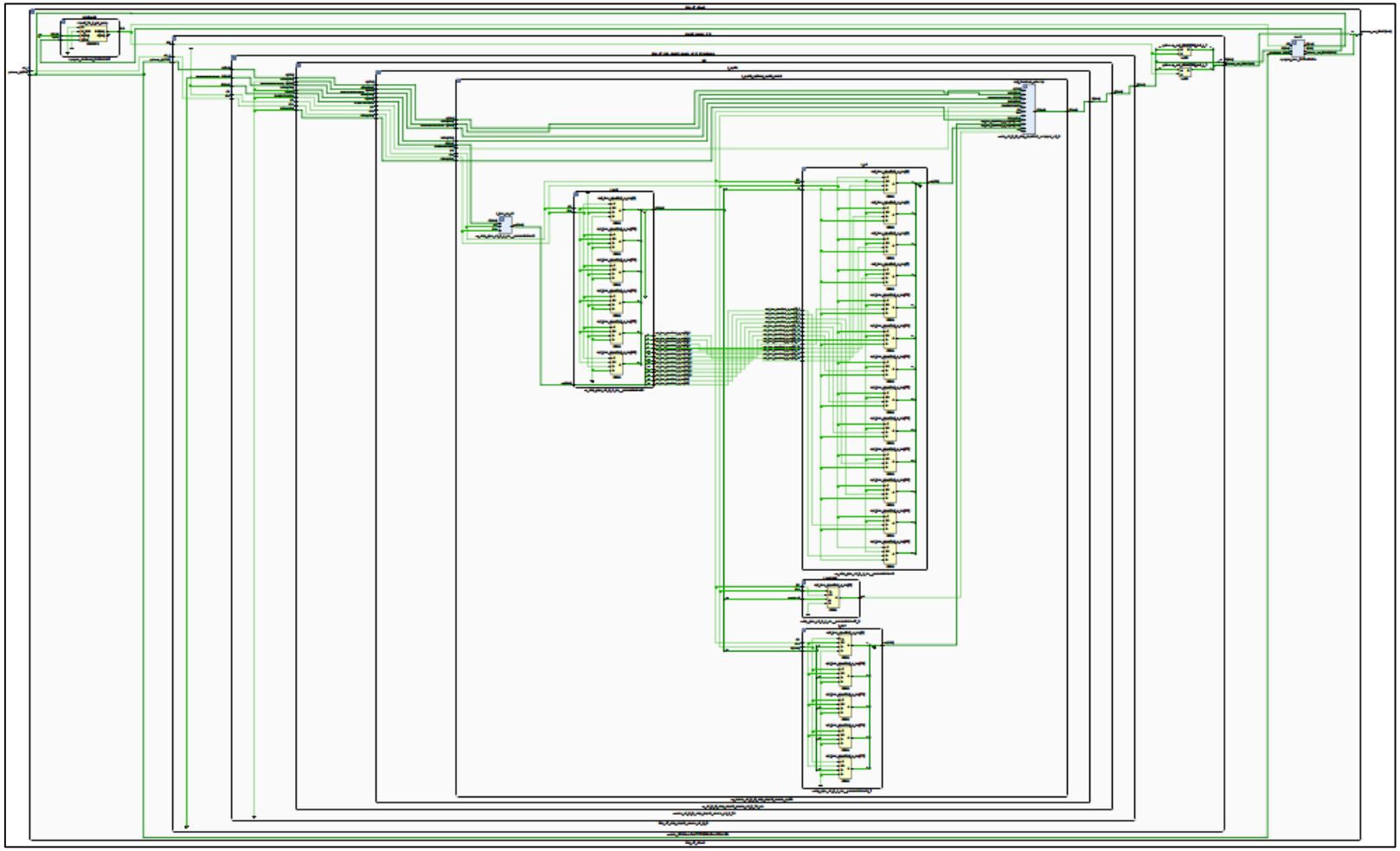


Figure 3-33 HLM schematic logic block array level view

3.5.2 BFL FPGA Architecture Design

The design for the family of BFL FPGA architectures are focused directly on the computed fixed point application as per Table 3-17. In this section, the respective BFL4, 6, and 8 are implemented in XSG and Vivado. The motivation of implementation for fixed point processing is key to reduction of complexity, latency, and power [120], [19]. More specifically, it allows fewer usage of DSP48E2 utilisation, LUTs, and flip flop needed. This indirectly, reduces resources used, latency, and power for optimised applications.

3.5.2.1 BFL4 Architecture

Similarly to the HLM, the BFL4 FPGA architecture is designed in XSG as illustrated in Figure 3-34.

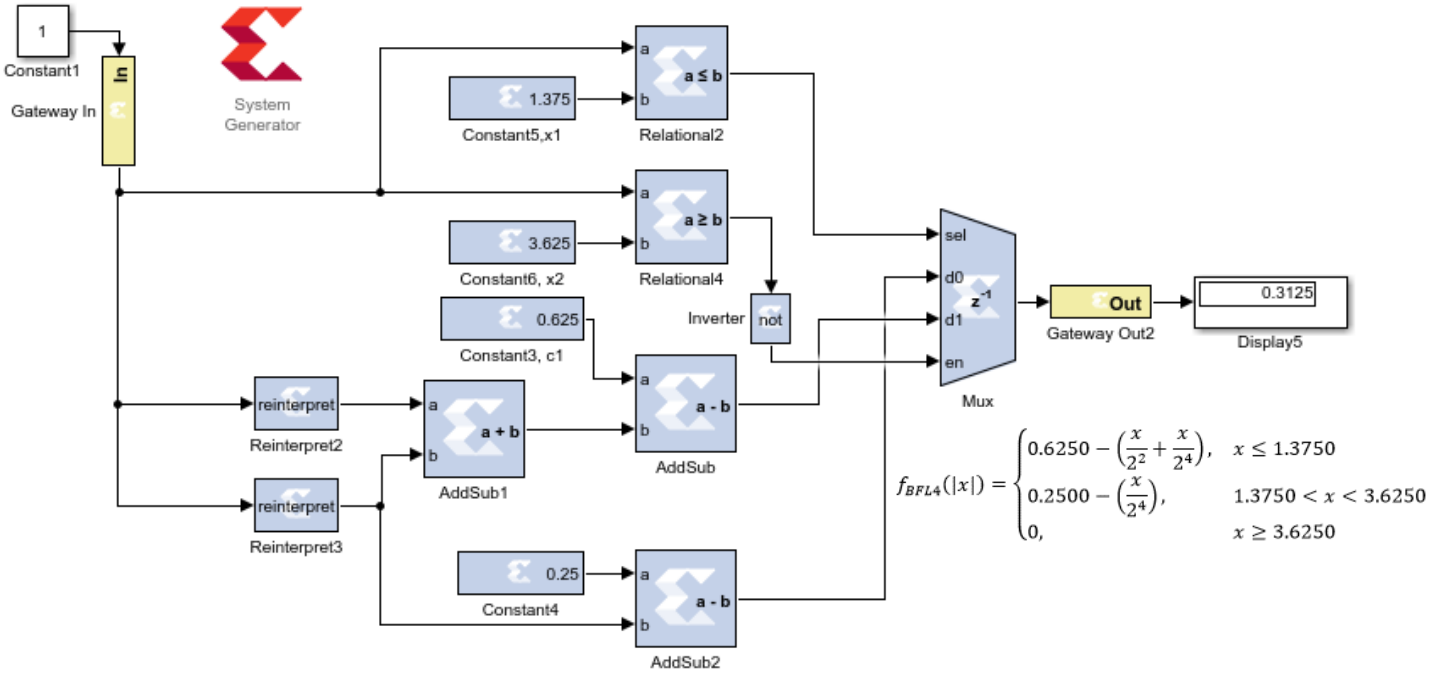


Figure 3-34 BFL4 approximation as implemented on XSG

Table 3-21 details the resources used for BFL4 along with a brief description for each block function.

Table 3-21 Resources used for the BFL4 design in XSG

Label	Parameters	Description
System Generator	Virtex Ultrascale+ XCVU9P	FPGA device selection and compiler that generates the HDL and Vivado project file
Constant1	1	Constant source representing value of $\|x\|$ for computation by $f_{HLM}(\|x\|)$
Gateway In	Default	Converts Constant1 into a Xilinx signal
Relational2	Compares $x \leq 1.375$	Comparator to control Mux selector
Constant5, x1	1.375	Constant stored in Look up Table (LUT)
Mux	Select regions of x based on $x \leq 1.375$ or otherwise	Multiplexer. Output is set to 6 bits length, 4 radix point.
Constant6,x2	3.625	Constant stored in Look up Table (LUT)
Relational4	Compares $x \geq 3.625$	Output will enable or disable Mux. If disabled, the output shall be driven to 0 to implement computation for range $x \geq 3.625$
not	Inverter	Disable or enable Mux (active high)
Reinterpret2	$x/2^2$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret3	$x/2^4$	Reinterprets fixed point value for right bit shift. No cost to hardware
AddSub1	$\frac{x}{2^2}+\frac{x}{2^4}$	Adds output from Reinterpret 2 and 3. Output is set to 6 bits length, 4 radix point.
Constant3, c1	0.625	Constant stored in Look up Table (LUT)
AddSub	0.625-$\left(\frac{x}{2^2}+\frac{x}{2^4}\right)$	Subtract. For Mux input $d1$. Output is set to 6 bits length, 4 radix point.
Constant4	0.25	Constant stored in Look up Table (LUT)
AddSub2	0.25-$x/2^4$	Subtract. For Mux input $d0$. Output is set to 6 bits length, 4 radix point.
Gateway Out	Default	Converts Xilinx signal into constant for Display1
Display1	Default	Constant viewer

The BFL4 is separated to three regions, with one of the first test to determine if $|x| \geq 3.6250$ using a comparator (Relational4) that directly controls the multiplexer (Mux). Should the value of $|x|$ falls within this region, the multiplexer will be disabled an all output will be tied to 0, following $f_{BFL4}(|x|)$. Otherwise, the multiplexer output will be enabled and shall take the value from its input, $d0$ or $d1$ depending on the input to the selector from (Relational2).

The Relational2 block determines whether $|x| \leq 1.375$, and if indeed so, it takes a value of logic '1' that selects the input $d1$ to the output of Mux and eventually

out through the gateway. The Reinterpret2, and 3 each performs a dedicated, combinational right bit shift to for $\frac{x}{2^2}$ and $\frac{x}{2^4}$ respectively. These values are then added with AddSub1 and subtracted from Constant3, c1 using AddSub to form the equation 0.625-$\left(\frac{x}{2^2}+\frac{x}{2^4}\right)$ for range $|x| \leq 1.375$.

Similarly, range $1.375 < x < 3.625$ are fed into input $d0$ of the multiplexer which is selected whenever the test for $|x| \leq 1.375$ and $|x| \geq 3.6250$ is not true. The input to is $d0 = 0.25 - \frac{x}{2^4}$. Care has been taken to exploit the use of pre-calculated bit shifts in order to minimise overall complexity of the logic circuit design.

Similar to the design flow for the HLM, the XSG model for BFL4 is compiled and exported to Vivado. The VHDL code, project file, and netlist are then migrated to Vivado for synthesis and post route and place. The results from post implementation or post route and place are taken for analysis and is an accurate tool for actual timing analysis, power consumption, and resource optimisation on the FPGA IC.

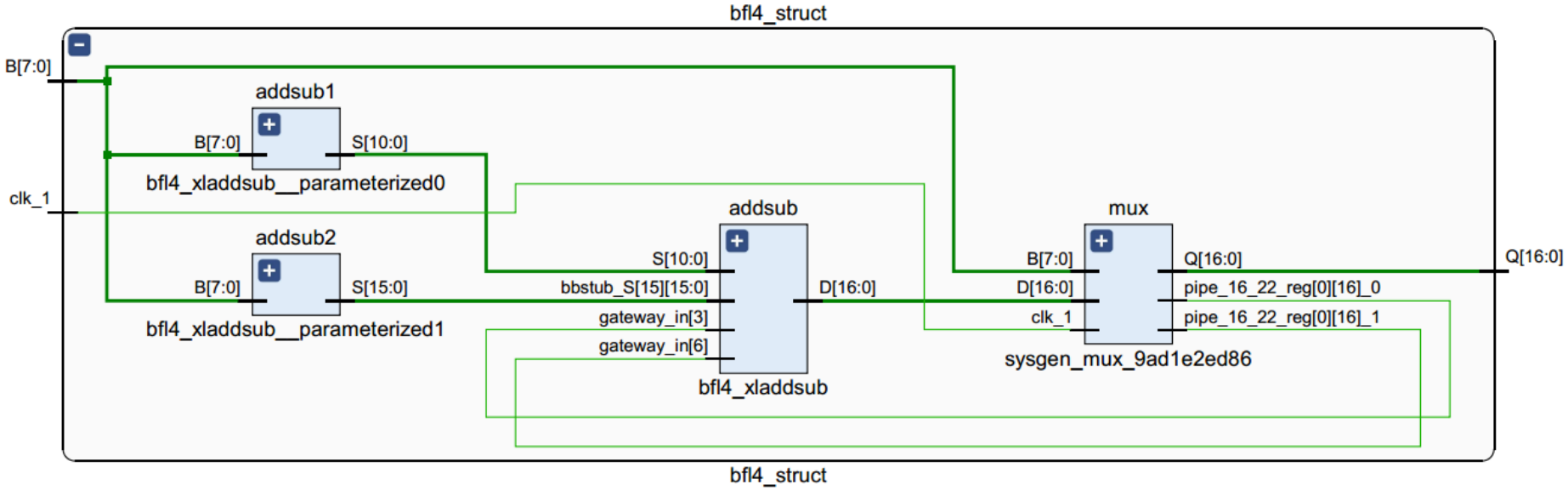


Figure 3-35 BFL4 top level schematic in Vivado

The top level schematics shown in Figure 3-35 verifies the XSG implementation with the relational, reinterpret function (without resources count) embedded within the addsub blocks. When expanded, the second level logic blocks are shown in Figure 3-36.

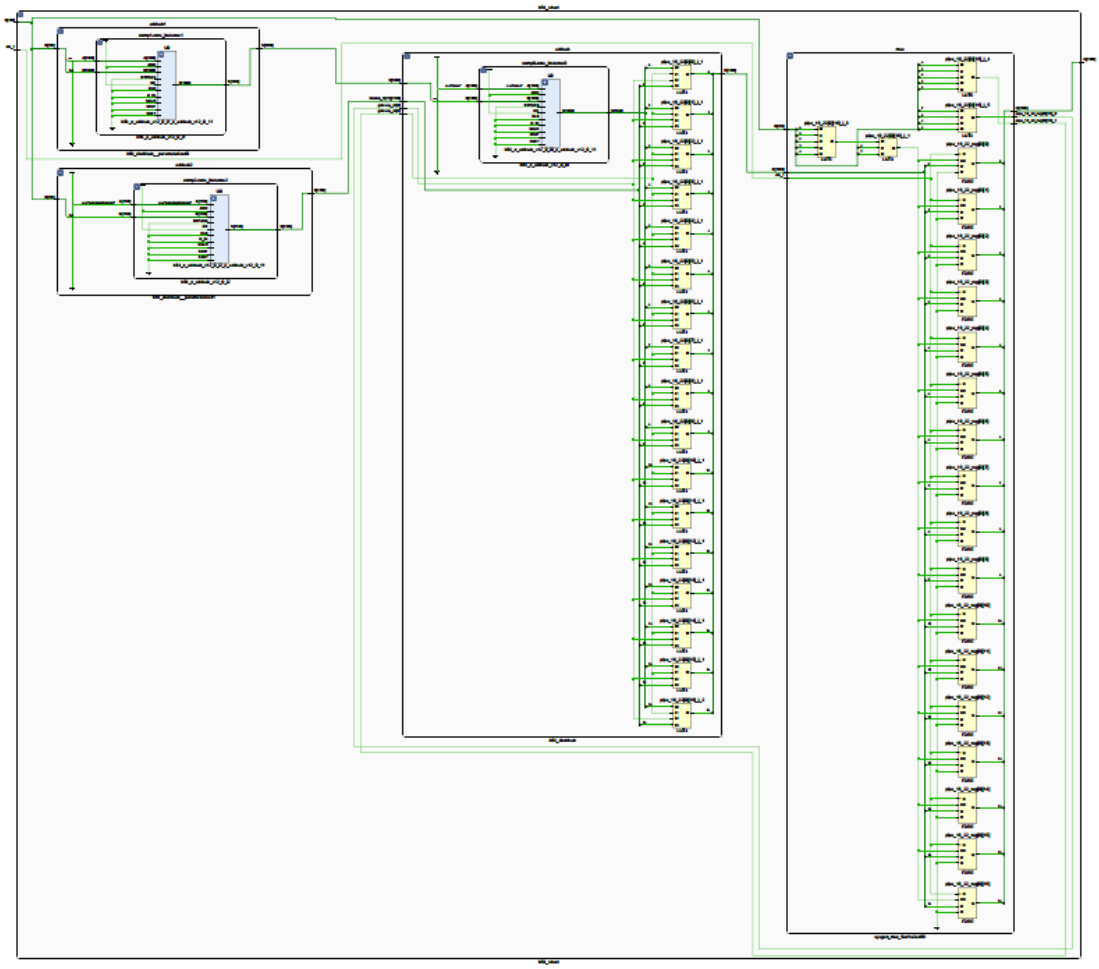
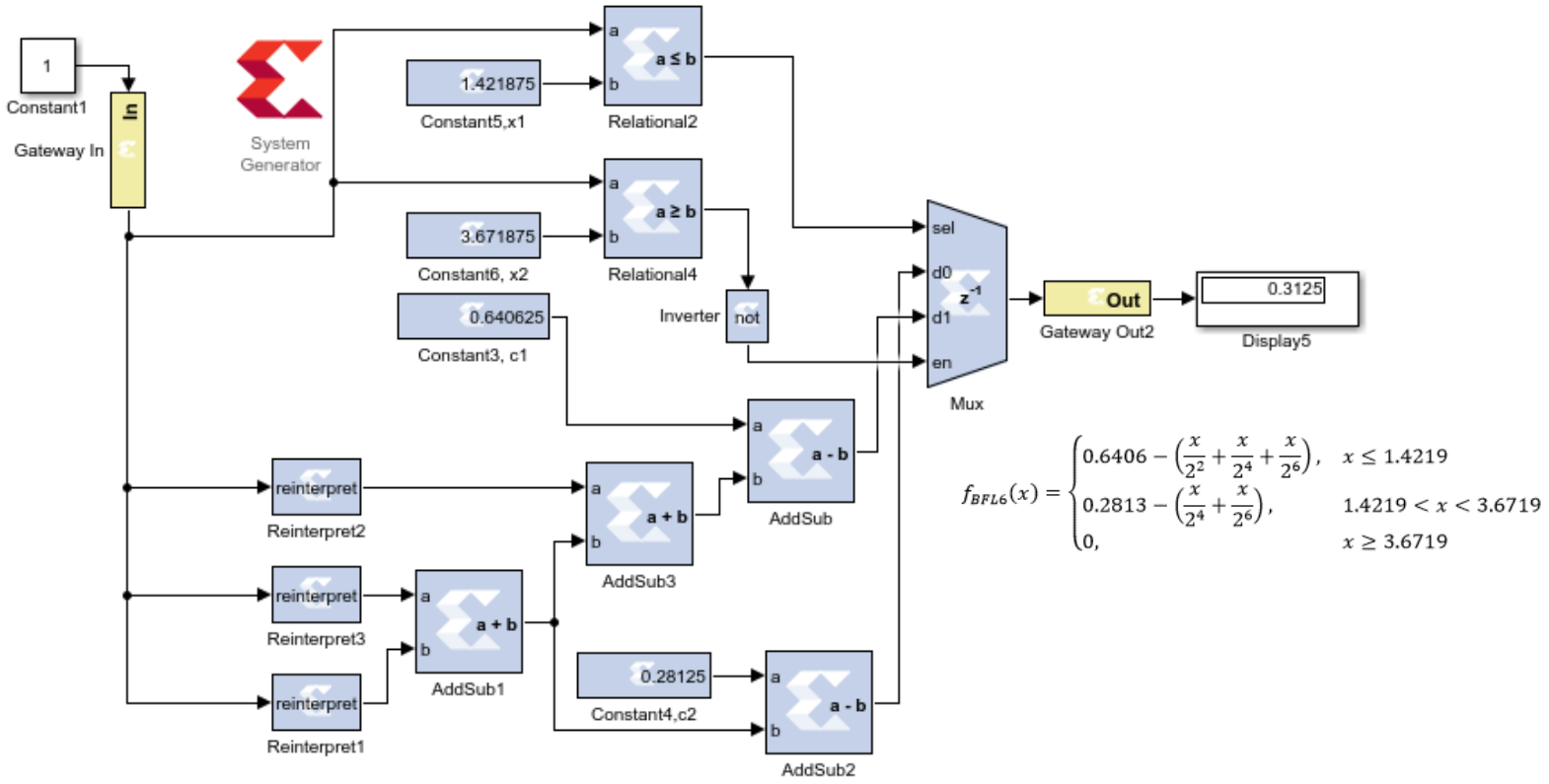


Figure 3-36 BFL4 schematic logic block array level view

The BFL6 and 8 shares a similar architecture to the BFL4 but with quantization effects when the radix is limited to $N = 6$ or 8. The BFL4 suffered no further quantization effects as shown in Figure 3-34 from what was presented in Table 3-16.

3.5.2.2 BFL6 Architecture

The BFL6 XSG implementation is illustrated in Figure 3-37.



$$f_{BFL6}(x) = \begin{cases} 0.6406 - \left(\frac{x}{2^2} + \frac{x}{2^4} + \frac{x}{2^6}\right), & x \le 1.4219 \\ 0.2813 - \left(\frac{x}{2^4} + \frac{x}{2^6}\right), & 1.4219 < x < 3.6719 \\ 0, & x \ge 3.6719 \end{cases}$$

Figure 3-37 BFL6 approximation as implemented on XSG

The implementation for BFL6 is very similar to the BFL4 as detailed in Table 3-22.

Table 3-22 Resources used for the BFL6 design in XSG

Label	Parameters	Description
System Generator	Virtex Ultrascale+ XCVU9P	FPGA device selection and compiler that generates the HDL and Vivado project file
Constant1	1	Constant source representing value of $\|x\|$ for computation by $f_{HLM}(\|x\|)$
Gateway In	Default	Converts Constant1 into a Xilinx signal
Relational2	Compares $x \leq 1.4219$	Comparator to control Mux selector
Constant5, x1	1.4219	Constant stored in Look up Table (LUT)
Mux	Select regions of x based on $x \leq 1.4219$ or otherwise	Multiplexer. Output is set to 8 bits length, 6 radix point.
Constant6,x2	3.6719	Constant stored in Look up Table (LUT)
Relational4	Compares $x \geq 3.6719$	Output will enable or disable Mux. If disabled, the output shall be driven to 0 to implement computation for range $x \geq 3.625$
not	Inverter	Disable or enable Mux (active high)
Reinterpret2	$x/2^2$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret3	$x/2^4$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret1	$x/2^6$	Reinterprets fixed point value for right bit shift. No cost to hardware
AddSub1	$\frac{x}{2^4}+\frac{x}{2^6}$	Adds output from Reinterpret 1 and 3. Output is set to 8 bits length, 6 radix point.
AddSub3	$\frac{x}{2^2}+\frac{x}{2^4}+\frac{x}{2^6}$	Adds output from Reinterpret 2 and AddSub1. Output is set to 8 bits length, 6 radix point.
Constant3, c1	0.6406	Constant stored in Look up Table (LUT)
AddSub	0.6406-$\left(\frac{x}{2^2}+\frac{x}{2^4}+\frac{x}{2^6}\right)$	Subtract. For Mux input $d1$. Output is set to 8 bits length, 6 radix point.
Constant4	0.2813	Constant stored in Look up Table (LUT)
AddSub2	0.2813-$\left(\frac{x}{2^4}+\frac{x}{2^6}\right)$	Subtract. For Mux input $d0$. Output is set to 8 bits length, 6 radix point.
Gateway Out	Default	Converts Xilinx signal into constant for Display1
Display1	Default	Constant viewer

There are two key differences between BFL4 and BFL6 i.e. quantization effects and additional logic blocks and adder for computation in the range of $x \leq 1.4219$. With the constants fixed at radix 6, the exact quantization effects to each constants in $f_{BFL6}(|x|)$ are as follows:

Table 3-23 Quantization effects to BFL6 constants with radix $N = 6$

Algorithm	N	Parameter values					
		c_1	m_1	c_2	m_2	x_1	x_2
f_{BFL6}	6	0.640625	NA	0.28125	NA	1.421875	3.671875

While Table 3-16 shows the quantisation effects rounded to the nearest four decimals for ease of discussion, Table 3-23 shows the exact quantization values when the radix is set at $N = 6$. The quantization effect for m_1 and m_2 are not applicable as the exact values form multiplications are computed using bit shifts without depending on the storage of a constant. With radix $N = 6$, the smallest value that is representable is 2^{-6} or 0.015625.

Another key difference from the BFL6 to the BFL4 is the additional bit shift term for $\frac{1}{2^6}$ along with an additional adder to complete the computation for range $x \leq 1.4219$. It is noted again that the usage of reinterpret block for bit shifts does not cost any hardware resources thereby reducing hardware complexity and provide for a combinational solution to allow low delays.

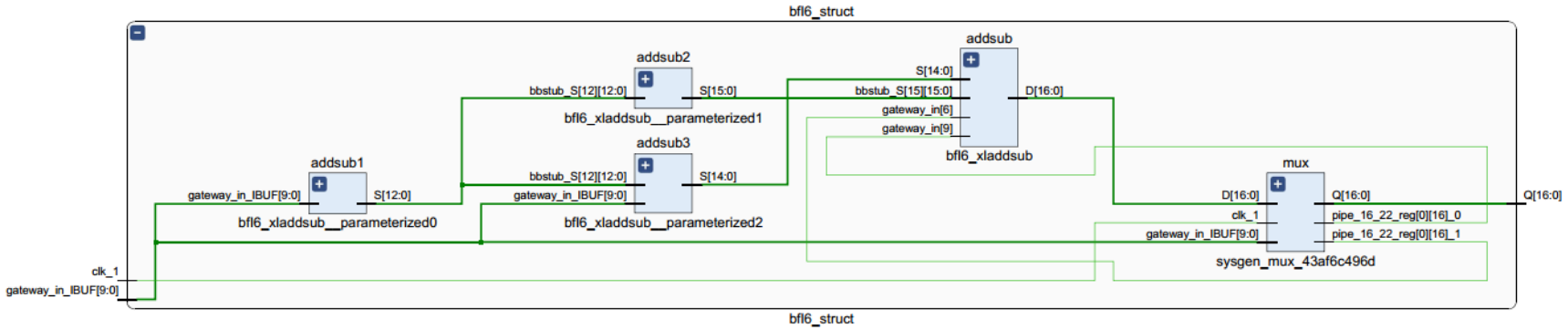


Figure 3-38 BFL6 top level schematic in Vivado

Figure 3-38 shows the top level schematic of the BFL6 that primarily consists of adders and multiplexers for the approximation. Relational operators are embedded within the adders and shown in Figure 3-39.

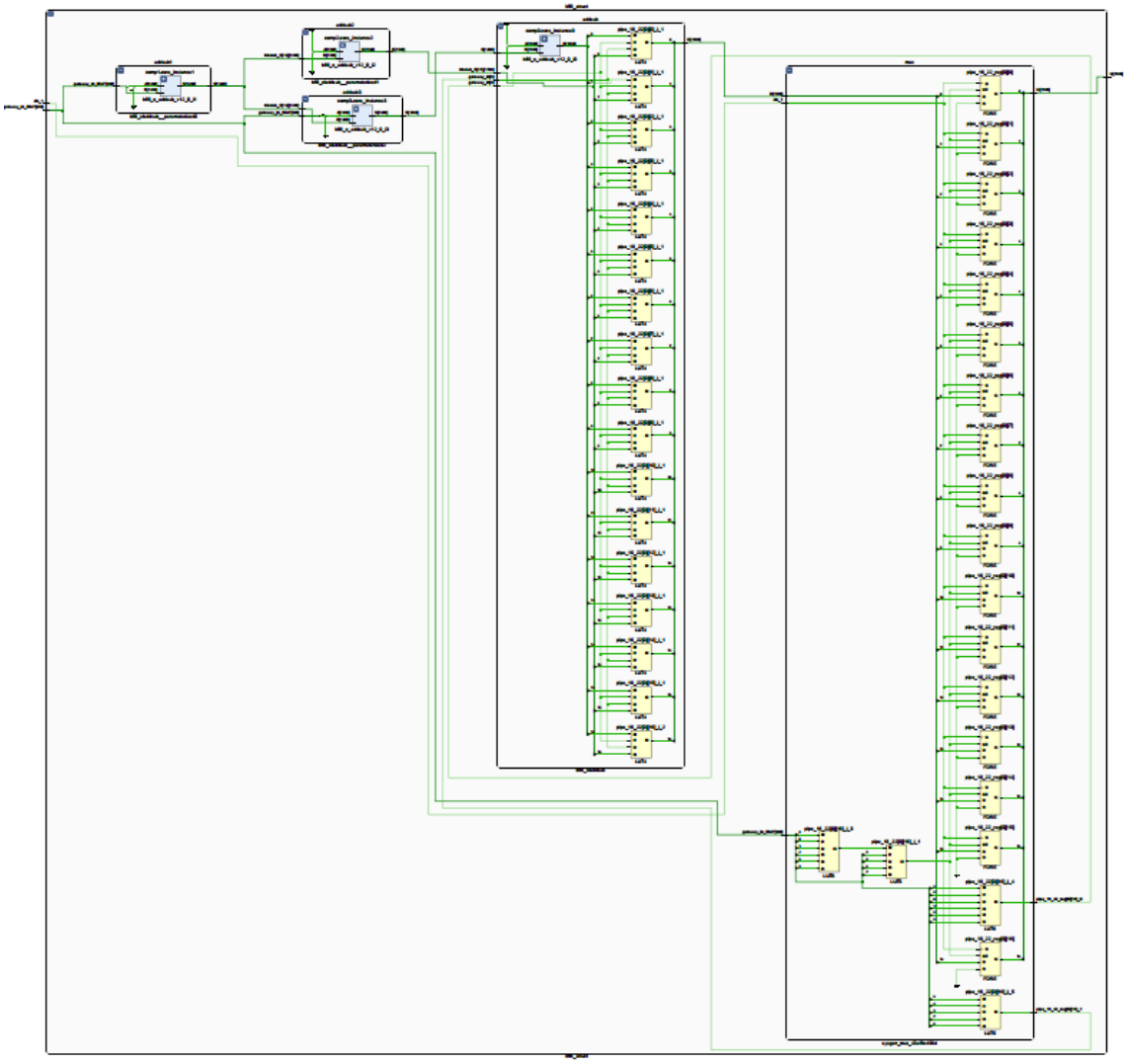


Figure 3-39 BFL6 schematic logic block array level view

3.5.2.3 BFL8 Architecture

The BFL8 approximation in XSG is illustrated in Figure 3-40:

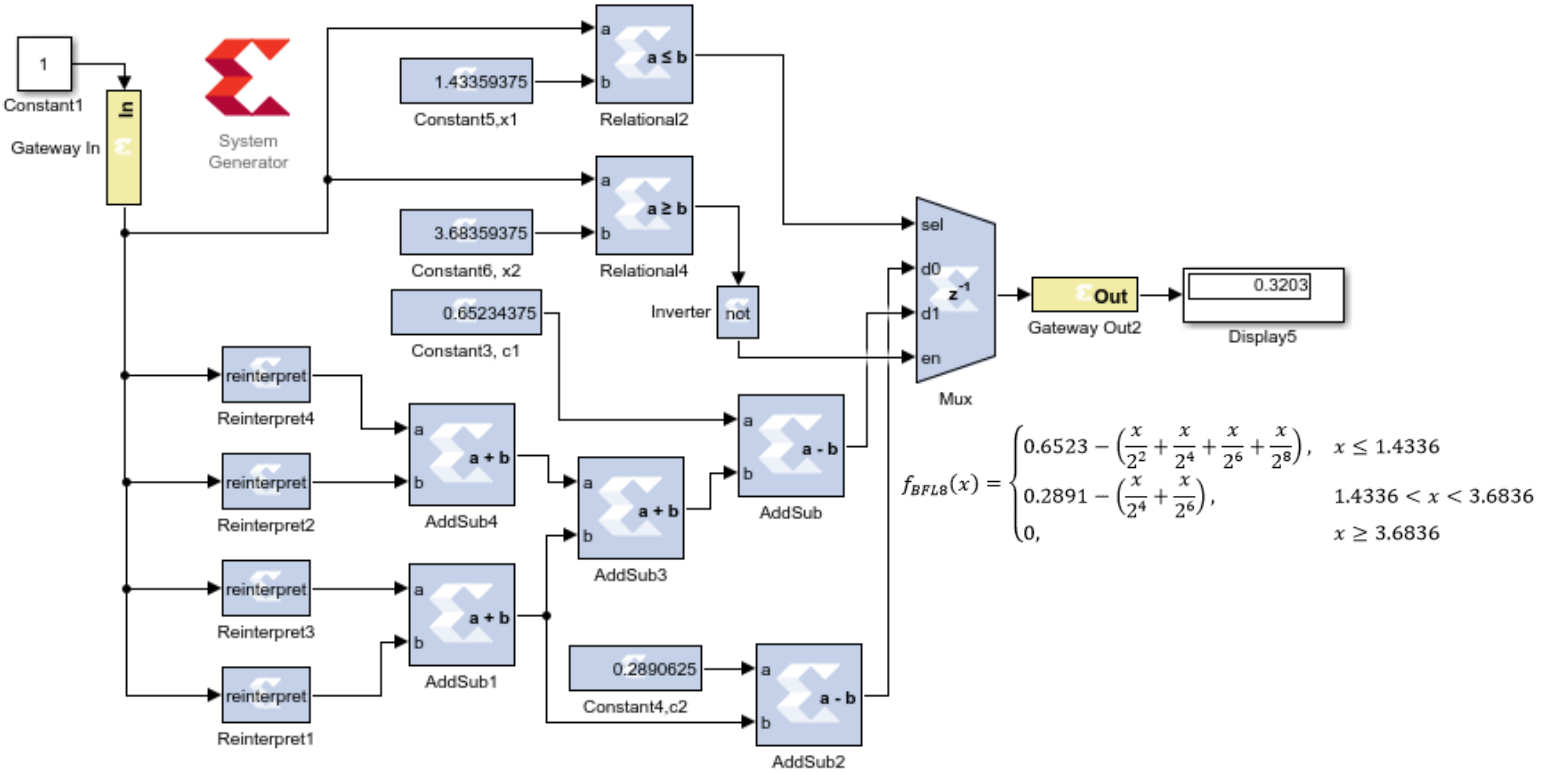


Figure 3-40 BFL8 approximation as implemented on XSG

Similar to the implementation of BFL4 and BFL6, the list of component blocks along with a brief description of their functions are listed in Table 3-24.

Table 3-24 Resources used for the BFL8 design in XSG

Label	Parameters	Description
System Generator	Virtex Ultrascale+ XCVU9P	FPGA device selection and compiler that generates the HDL and Vivado project file
Constant1	1	Constant source representing value of $\|x\|$ for computation by $f_{HLM}(\|x\|)$
Gateway In	Default	Converts Constant1 into a Xilinx signal
Relational2	Compares $x \leq 1.4336$	Comparator to control Mux selector
Constant5, x1	1.4336	Constant stored in Look up Table (LUT)
Mux	Select regions of x based on $x \leq 1.4336$ or otherwise	Multiplexer. Output is set to 10 bits length,8 radix point.
Constant6,x2	3.6836	Constant stored in Look up Table (LUT)
Relational4	Compares $x \geq 3.6836$	Output will enable or disable Mux. If disabled, the output shall be driven to 0 to implement computation for range $x \geq 3.6836$
not	Inverter	Disable or enable Mux (active high)
Reinterpret2	$x/2^2$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret3	$x/2^4$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret1	$x/2^6$	Reinterprets fixed point value for right bit shift. No cost to hardware
Reinterpret4	$x/2^8$	Reinterprets fixed point value for right bit shift. No cost to hardware
AddSub1	$\frac{x}{2^4}+\frac{x}{2^6}$	Adds output from Reinterpret 1 and 3. Output is set to 10 bits length,8 radix point.
AddSub4	$\frac{x}{2^2}+\frac{x}{2^8}$	Adds output from Reinterpret 4 and 2. Output is set to 10 bits length,8 radix point.
AddSub3	$\frac{x}{2^2}+\frac{x}{2^4}+\frac{x}{2^6}+\frac{x}{2^8}$	Adds output from AddSub1 and 4. Output is set to 10 bits length,8 radix point.
Constant3, c1	0.6406	Constant stored in Look up Table (LUT)
AddSub	0.6406-$\left(\frac{x}{2^2}+\frac{x}{2^4}+\frac{x}{2^6}\right)$	Subtract. For Mux input $d1$. Output is set to 10 bits length,8 radix point.
Constant4	0.2891	Constant stored in Look up Table (LUT)
AddSub2	0.2891-$\left(\frac{x}{2^4}+\frac{x}{2^6}\right)$	Subtract. For Mux input $d0$. Output is set to 10 bits length,8 radix point.
Gateway Out	Default	Converts Xilinx signal into constant for Display1
Display1	Default	Constant viewer

With the radix for implementation set to $N = 8$, the quantization effects to $f_{BFL8}(|x|)$ constants are listed in Table 3-25. The smallest value that is represented with radix 8 is 0.00390625.

Table 3-25 Quantization effects to BFL8 constants with radix $N = 8$

Algorithm	N	Parameter values					
		c_1	m_1	c_2	m_2	x_1	x_2
f_{BFL8}	8	0.65234375	NA	0.2890625	NA	1.43359375	3.68359375

Compared to BFL6, the BFL8 includes an additional bit shift term for computation in the region of $x \leq 1.4336$ which is implemented using the reinterpret block and an additional adder. The bit shift terms added in the region of $1.4336 < x < 3.6836$ i.e. $\frac{x}{2^4} + \frac{x}{2^6}$ is reused in the computation for the $|x| \leq 1.4336$ region to reduce complexity.

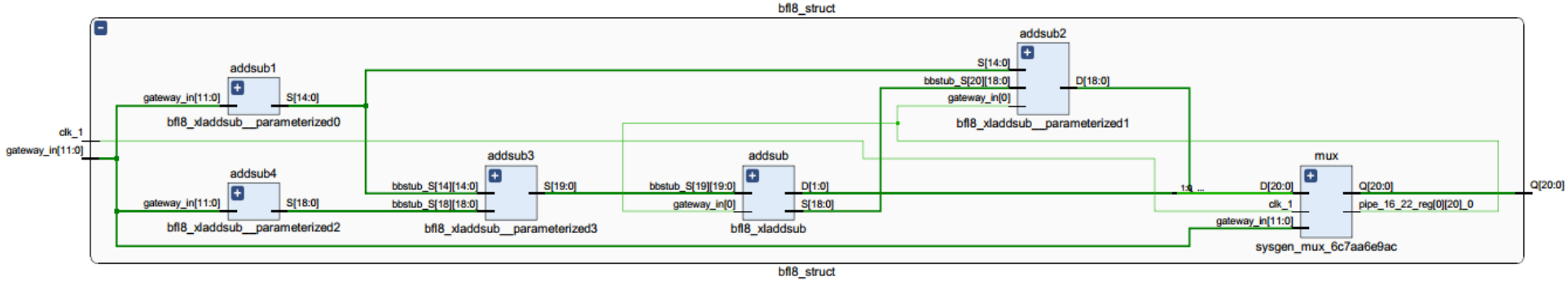


Figure 3-41 BFL8 top level schematic in Vivado

The BFL8 top level schematic in Figure 3-41 verifies the implementation in XSG with an additional addsub block to cater to the additional bit shift value, $\frac{x}{2^8}$ for region $|x| \leq 1.4219$. The next level schematic view of BFL8 is illustrated in Figure 3-42.

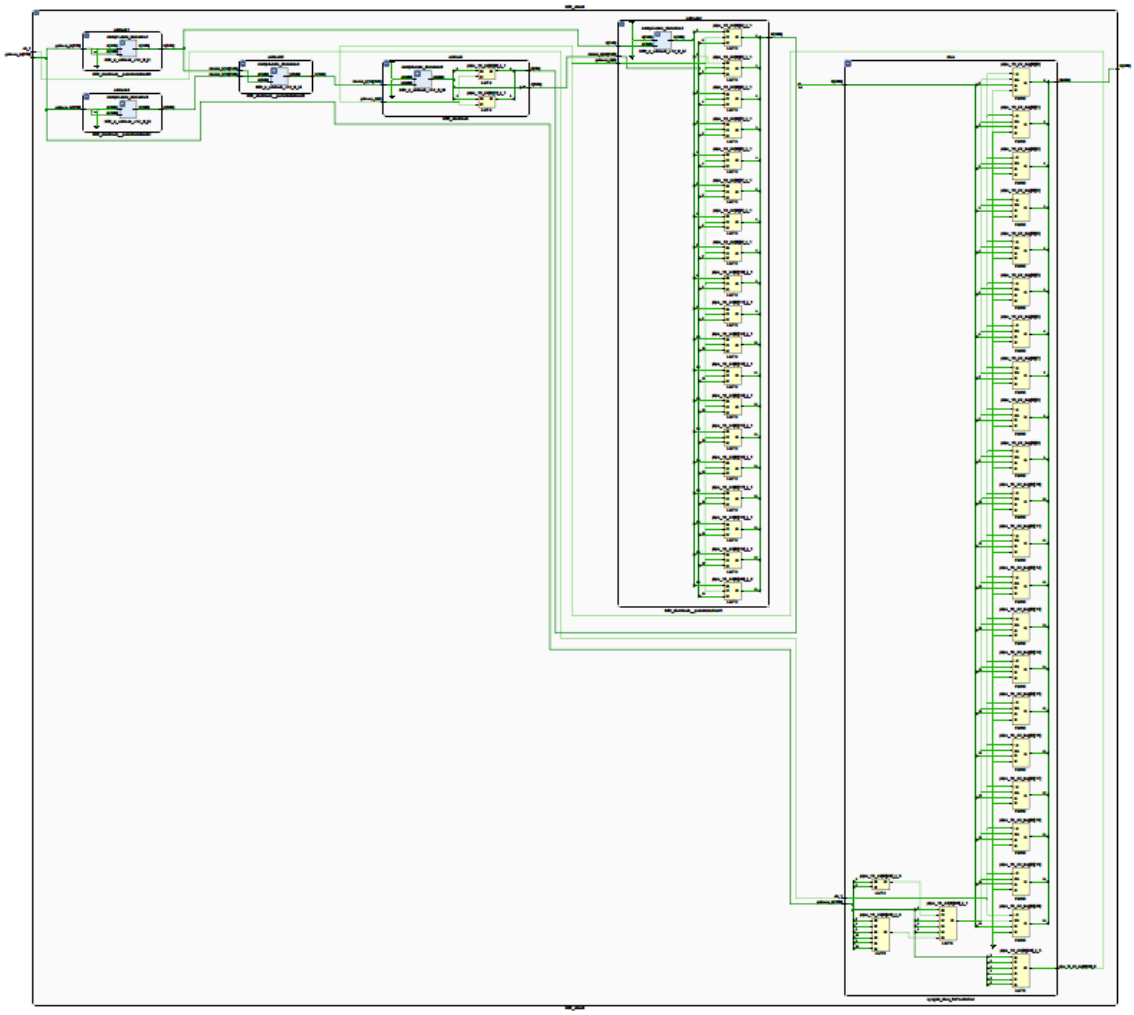


Figure 3-42 BFL8 schematic logic block array level view

The timing, power, and resources results for all BFL implementation shall be presented in the next chapter. The results presented are based on post place and route implementation and results from the simulation. Results from the post implementation stage or post place and route are considered accurate and is referred to over the post synthesis results which are merely estimates of the implementation on FPGA.

4 RESULTS AND DISCUSSION

4.1.1 Introduction

This chapter is divided into five main sections as summarised in Figure 4-1.

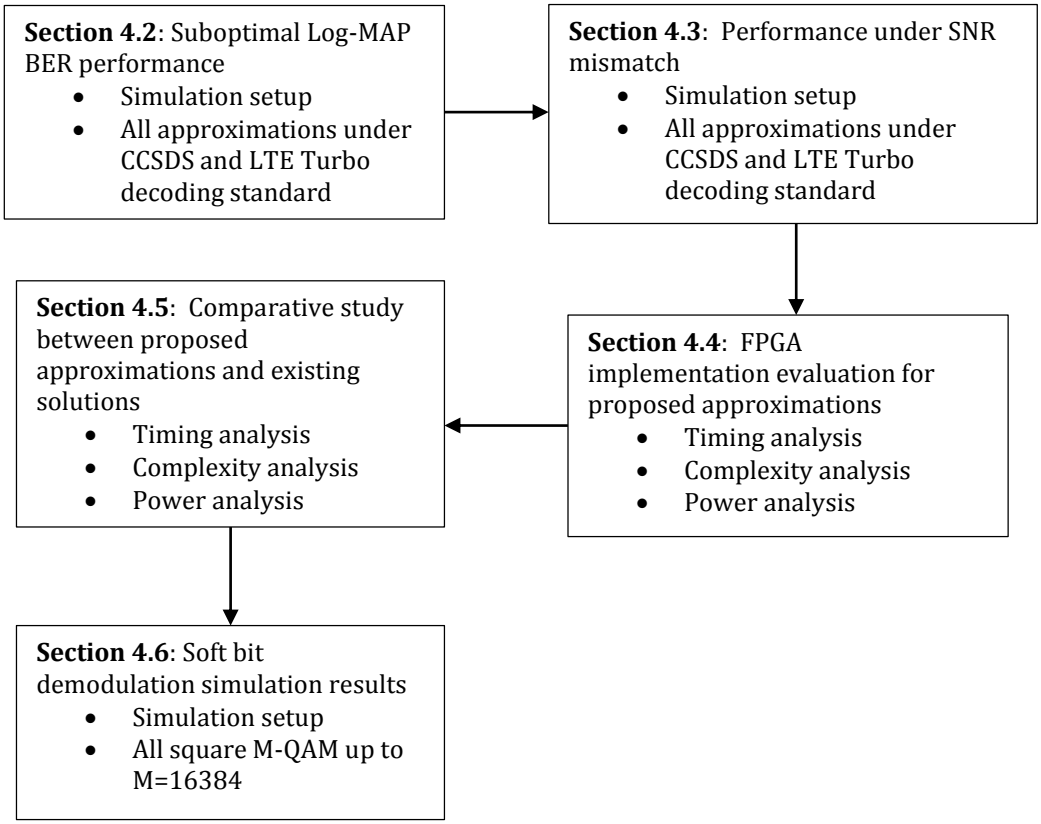


Figure 4-1 Summary and flow of results presentation

The chapter begins with the presentation of BER performance simulation results for selected approximation based on the review provided in Section 2.2.10.9, Chapter 2 (CLM, MLM, LLM, LC approximations) for hardware friendly focused implementation along with the proposed HLM and BFL approximations for wireless mobile and deep space application. This is followed by the BER performance under SNR mismatch conditions for all approximations, and followed by the detailed results from hardware implementation of the HLM and BFL. The following section then compares the results from HLM and BFL with

other approximations. The last section presents the soft bit demodulation results for M-QAM in wireless communications.

4.2 Suboptimal Log-MAP BER Performance

This section presents the BER performance of various suboptimal Log-MAP algorithms under the 4G-LTE Advanced and CCSDS Turbo decoding standard respectively. The simulation environment is simulated as described in Section 2.2.11 and summarised before each results are presented.

4.2.1 Performance of Various Suboptimal Log-MAP under the Eight States 4G-LTE Advanced Turbo Decoder

The 4G-LTE Advanced Turbo encoder schematic was previously discussed in Section 2.2.3.1. A 4G-LTE Advanced Turbo decoding system was programmed in MATLAB and simulated with all suboptimal algorithm discussed in this research. The simulations in this section assumes perfect knowledge of channel coefficients i.e. $\sigma_m^2 = 0$ with the only varying factor of the correction function approximation algorithm within each standard's channel conditions. The codes for simulation can be referred to in Appendix F.

4.2.1.1 Simulation Setup

A summary of simulation parameters are specified in Table 4-1 in accordance to the 4G-LTE Advanced standard [12], [52].

Table 4-1 Summary of 4G-LTE Advanced Turbo decoder simulation parameters

Parameter	Measure
Turbo code polynomial Generator	g[13,15]
Turbo encoder constraint length	4
Number of Turbo decoder iterations	10
Iteration stop criteria	Magic genie
Code rate	1/2
Channel type	Rayleigh, AWGN
Modulation	BPSK
Interleaver type (Size)	S-random (640)
Number of Frames	50,000
Total bits simulated	32,000,000

Consistent with the 4G-LTE Advanced standard, the system was simulated over a Rayleigh faded, AWGN channel using S-random interleaver with 640 bits block length [12], [52]. A total of 10 iterations were prescribed for the decoder with Magic Genie iteration stopping rule as explained in Chapter 2, Section 2.2.11.5, to save simulation time especially for high SNR scenarios requiring lesser iteration to correctly decode the received bits. With the use of the Magic Genie stopping rule, the simulation time can be reduced significantly especially for higher SNRs and the total simulation time taken is seven to nine days depending on the suboptimal approximation used. Suboptimal approximation with the highest complexity such as the exact Log-MAP recorded the longest simulation time. This is consistent throughout all simulations done in this research. A Monte Carlo simulation of 32×10^6 bits were simulated for all suboptimal Log-MAP approximations with the BER vs E_b/N_0 is shown in Figure 4-2:

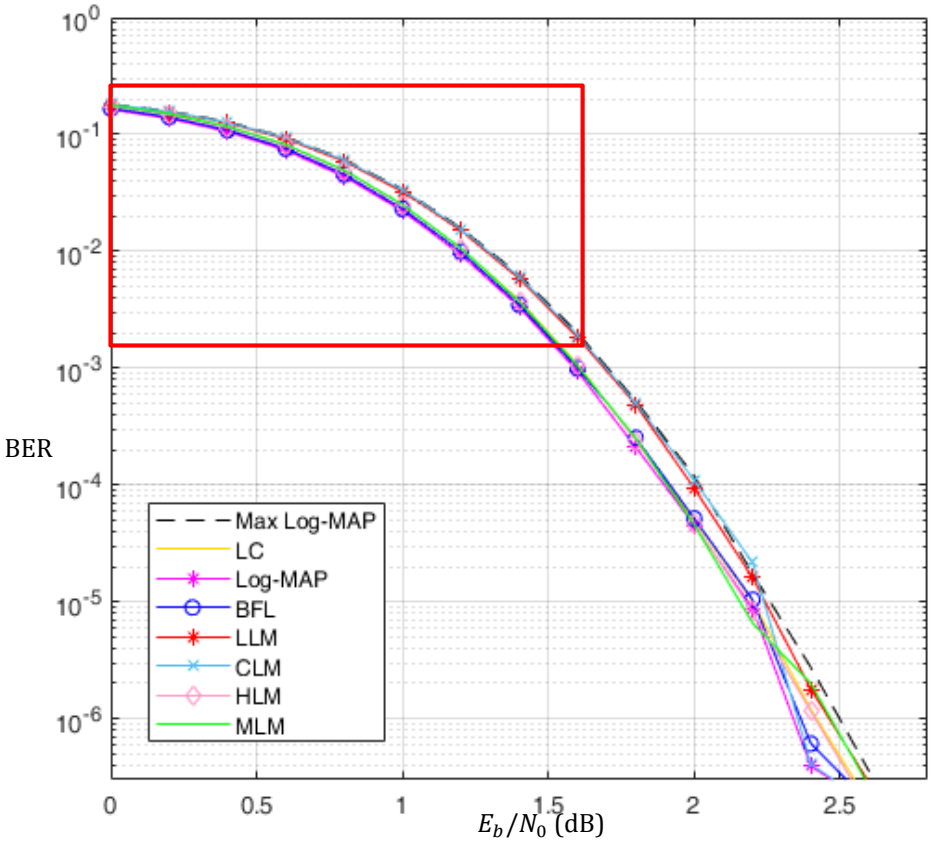


Figure 4-2 4G-LTE Advance suboptimal Log-MAP BER performance

The BER performance of each suboptimal Log-MAP algorithm simulated according to the 4G standard is shown in Figure 4-2. It is observed that the performance gain for the suboptimal algorithms performs closely to one another. A closer look into the performance of suboptimal Log-MAP approximations at lower SNR in the highlighted region is shown in Figure 4-3:

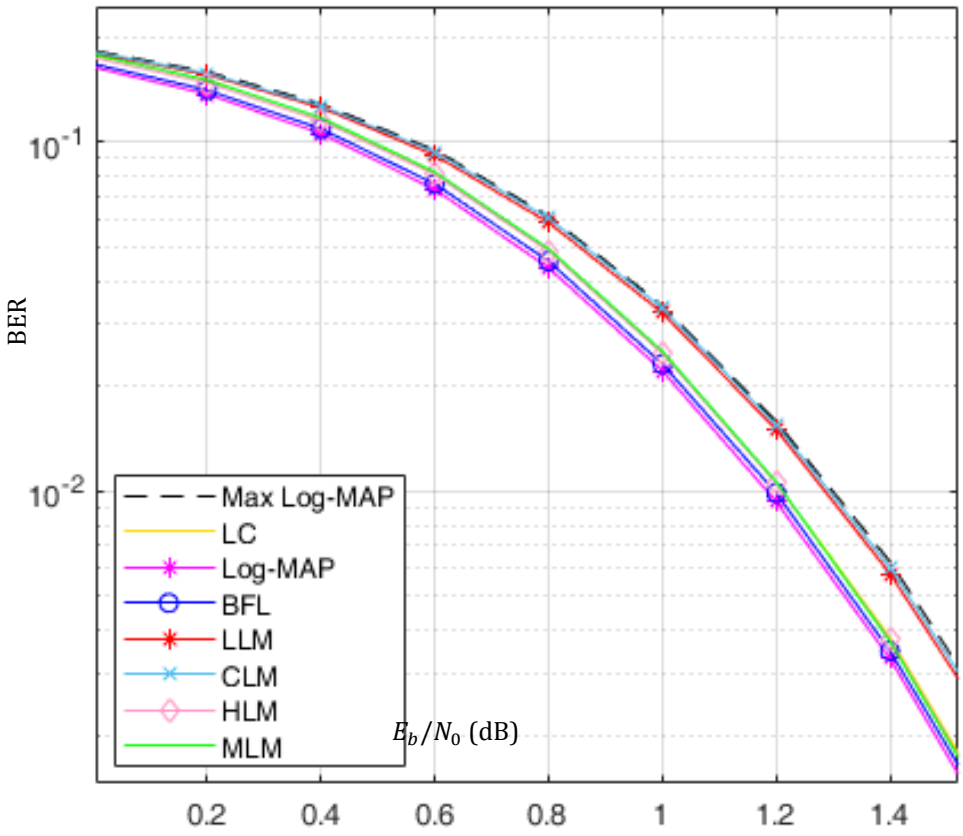


Figure 4-3 Enlarged highlighted region of Figure 4-2

It is observed that the best performance, is still that of the computationally complex exact Jacobian logarithm with exponential correction function. Most suboptimal algorithm performance fall between the performance bounds of the Max-Log-MAP and the Log-MAP with the exception of MLM algorithm that records weaker performance for low SNR, but stronger BER performance than Log-MAP for intermediate SNR values, and then again weakening BER performance intermittently. This result for the MLM is consistent with findings from other works reported in [30] due to the nature of the MLM algorithm approximating the exact correction function with repetitive positive and negative residual distribution with $|x|$. This is consistent even in the CCSDS decoder and indicated in the authors' paper [30]. Despite this slight advantage, the MLM performs relatively weaker in channels with SNR mismatch conditions as will be shown later and in lower SNR regions.

Based on the results, it is encouraging to note that the BFL gives the closest performance to the exact Log-MAP within 0.015 dB BER difference using simple hardware implementation using reinterpret blocks with no hardware cost, comparators, and additions, which are combinational in nature and independent of clock speed that allows for higher clock speed implementation limited only to the FPGA system's maximum clock speed. Next is a very close performance between the HLM, MLM, and LC approximation, followed by the LLM and CLM before finishing with the Max-Log-MAP as the performance bound for suboptimal Log-MAP algorithms. Relating these results with each approximation accuracy in Table 3-15, it is shown that higher approximation accuracy to the correction function, yields closer performance to the exact Log-MAP. It is also expected that the BFL8 will perform closely to the exact BFL with 8 radix point implementation. It is also important to highlight that other approximations are not fixed point friendly and is expected to suffer losses if radix point quantization is in effect. It is also noted however, that the results for HLM, MLM, and LC approximation is not as apparent which brings us to the simulation of the suboptimal Log-MAP algorithms in the 16 states, CCSDS Turbo decoder. The HLM recorded differences within 0.017 dB against the exact Log-MAP.

The suboptimal Turbo decoder behaves more sensitively with increased constrain length due to increased computation frequency of the Jacobian logarithm computation for higher states. It is for this reason that the research also took into account, higher Turbo decoder states standards such as the CCSDS that will be presented in the next section to better analyse the BER performance of each suboptimal Log-MAP approximation.

4.2.2 Performance of Various Suboptimal Log-MAP under the CCSDS Decoder

The CCSDS standard as well as encoder schematic was previously introduced in Section 2.2.3.2. A main factor in employing the CCSDS Turbo decoder standard is due to the longer constraint length used; demanding 16 states of metric computation. A CCSDS Turbo decoder was custom coded in this research in Appendix G and the BER result of each individual suboptimal Log-MAP algorithm are presented in Figure 4-4.

4.2.2.1 Simulation Setup

The simulation parameters [54] are summarised in Table 4-2.

Table 4-2 Summary of CCSDS simulation parameters

Parameter	Measure
Turbo code polynomial generator	g[23,33]
Turbo encoder constraint length	*5*
Number of Iterations	5
Iteration stop criteria	Magic genie
Code rate	1/2
Channel type	AWGN
Modulation	BPSK
Interleaver (Size)	Pseudo random (1000)
Number of Frames	10,000
Total bits simulated	10,000,000

The total bits simulated are limited to 10^6 due to much longer simulation time needed to complete each suboptimal Log-MAP approximation even with the genie stopping rule. The number of iterations are also reduced as Turbo decoders with longer constraint length has better code correction ability and can afford to take lesser number of iteration for faster convergence although iterations could typically be from 4 up to a maximum of10 [121], [55]. For this research the iteration number is kept to 5 to save simulation time and allow the decoder to converge with reliable results [122].

The BFL followed by the HLM and LC recorded the closest performance to the Log-MAP algorithm for low SNR values as shown in Figure 4-4.

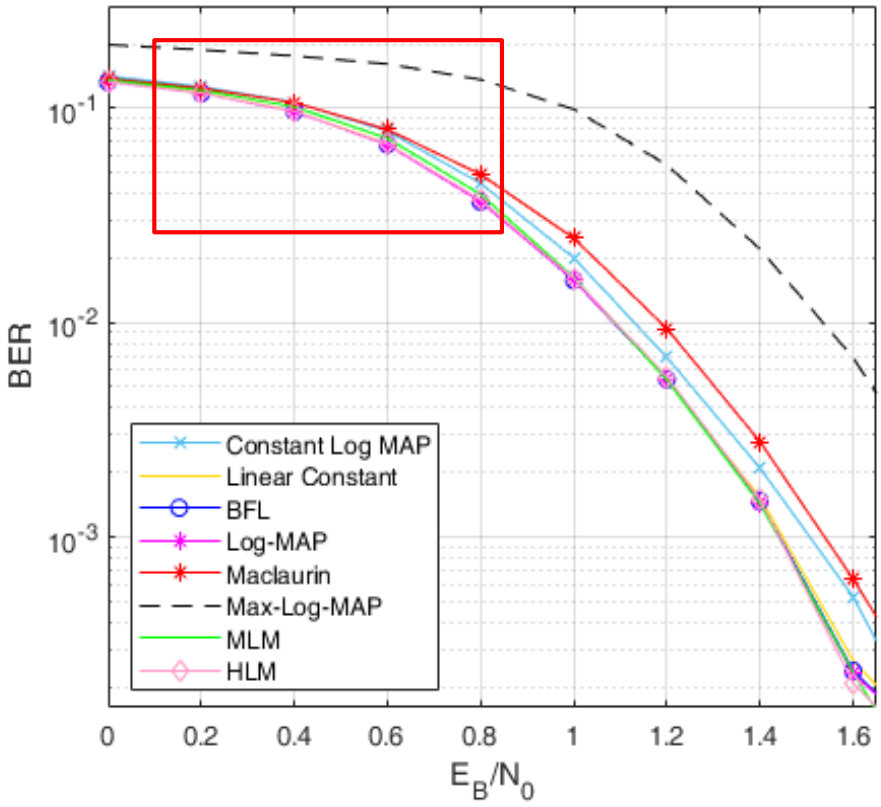


Figure 4-4 CCSDS Suboptimal Log-MAP BER Performance

Generally, it is observed that the importance of the suboptimal Log-MAP method in higher constraint length Turbo decoder. The performance gap between the Max-Log-MAP and the exact Log-MAP is wider compared to the mobile wireless decoder. Figure 4-4 has been enlarged for lower values of SNR and fewer algorithms has been plotted against each other for a closer analysis for the same graph in Figure 4-5.

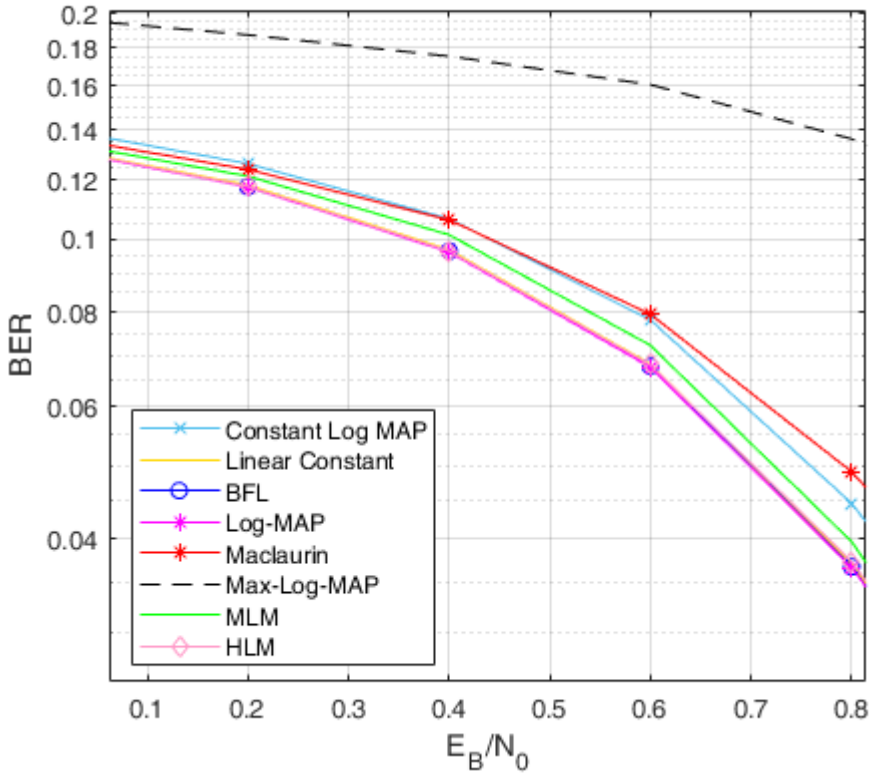


Figure 4-5 Enlarged area highlighted in Figure 4-4

The BFL and HLM algorithms has recorded the closest performance to the exact Log-MAP followed closely by LC for low SNR. We continue to observe the MLM algorithm behaviour that intermittently performs weaker or stronger against the Log-MAP. Nevertheless, in terms of overall performance gain of various algorithms recorded, the BFL, followed by the HLM, continues to adhere to the performance of the exact Log-MAP. It is also noted that the performance gain correlates with individual algorithm GOF measure, in which, better GOF gives improved performance.

4.2.2.2 BFLN Performance under CCSDS Turbo Decoder

This section evaluates the performance of the BFL4, 6, and 8 against the exact Log-MAP and the Max-Log-MAP as illustrated in Figure 4-6. The CCSDS Turbo decoder was chosen to evaluate the BFLN (where, $N = 4, 6, 8$) approximations

as the CCSDS possesses higher constraint length for its decoder and is therefore more sensitive to the correction function accuracy.

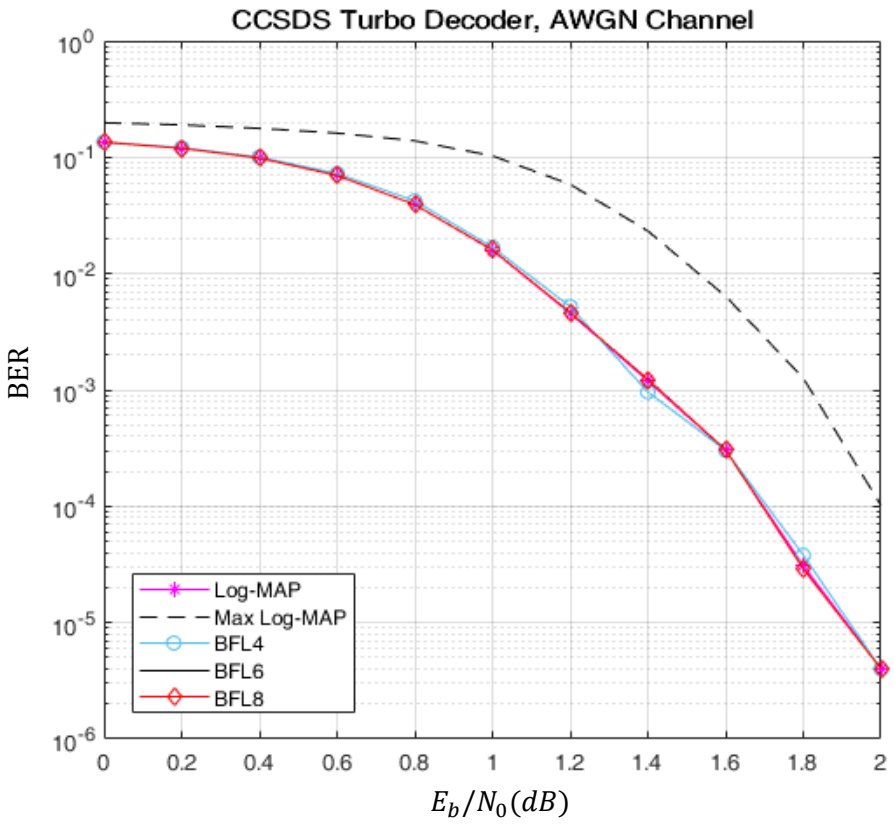


Figure 4-6 CCSDS Suboptimal Log-MAP BER Performance for BFL_N

The BFL4, BFL6 and BFL8 shows very little differences from the original BFL algorithm and shows close adherence to the exact Log-MAP solution. Although the difference are small, upon closer inspection, the BFL gave the best performance followed by BFL8, BFL6 and BFL4. This result gives an indication that the BFL4 although less accurate than its counterparts, would be sufficient to give good BER performance with reduced complexity in hardware design. Again it should be highlighted that with the implementation of BFLN, the full effect of BER performance after quantization of limited radix points are presented as opposed to other approximations which are not custom catered to radix point implementation. The BFL and HLM recorded a performance

difference within 0.0003 dB and 0.0005 dB respectively against the exact Log-MAP solution.

4.3 Performance under SNR Mismatch

This section presents the results for SNR mismatch for the 4G-LTE Advanced and CCSDS Turbo decoder.

4.3.1 SNR Mismatch Simulation under the 4G-LTE Advanced decoder

The 4G-LTE Advanced simulation sees two parameters that may be estimated in error at the receiver end. One being the SNR and another, the channel coefficients $\widehat{h_k}$. The erroneous SNR is simply simulated as an offset of -6 to 6 dB from the true SNR and the estimated channel coefficients are simulated as $\widehat{h_k} = h_k + m_k$ where h_k represents the true channel coefficient values and m_k represents the offset values of variance σ_m^2 denoting the degree of erroneous channel coefficient linked variance . Figure 4-7 and Figure 4-8 shows the results for SNR mismatch simulation with $\sigma_m^2 = 0.1$ for HLM, BFL, CLM; and LLM, MLM, LC respectively against the Log-MAP and Max-Log-MAP performance [89].

4.3.1.1 Simulation Setup

The simulation setup is the same as described in Table 4-1 with exception to the received channel coefficient that are incorrectly received and influenced by σ_m^2 due to misestimation errors. The SNR is also misestimated with offset of -6 to 6dB and the MATLAB codes are described in Appendix H.

For an organised presentation of results due to relatively high number of approximation results to present, the SNR offsets (representing SNR mismatch) graphs are composed as follows:

Table 4-3 Organisation of SNR mismatch results for the 4G-LTE Advanced Turbo decoder

Sub sections	SNR Mismatch Figures	True SNR values (dB)	Approximation Algorithm	σ_m^2 value
4.3.1.1.1	Figure 4-7	2.4, 2.8	HLM, BFL, CLM, Log-MAP, Max-Log-MAP	0.1
4.3.1.1.1	Figure 4-8	2.4, 2.8	LLM, MLM, LC, Log-MAP, Max-Log-MAP	0.1
4.3.1.1.2	Figure 4-11	6.4, 7, 7.7	HLM, BFL, CLM, Log-MAP, Max-Log-MAP	0.4
4.3.1.1.2	Figure 4-12	6.4, 7, 7.7	LLM, MLM, LC, Log-MAP, Max-Log-MAP	0.4

In addition, for each of the True SNR values, a BER difference bar graph is provided to ease analysis and presented after respective figures according to the respective true SNR and σ_m^2 values.

The true SNR values are chosen to specifically cater for a set of results within each decade in the BER logarithmic scale in the range of 10^{-2} (2.4 dB), $10^{-3}(2.8dB)$ for $\sigma_m^2 = 0.1$; and 10^{-2} (6.4 dB), $10^{-3}(7dB)$, $10^{-4}(7.7dB)$ for $\sigma_m^2 = 0.4$. It is chosen as such so that the presentation for each results are better organised.

4.3.1.1.1 Results with channel coefficient offset variance $\sigma_m^2 = 0.1$

The SNR mismatch results with misestimated channel coefficient of variance $\sigma_m^2 = 0.1$ for the HLM, BFL and CLM are presented in Figure 4-7 and the remaining approximation, LLM, MLM, and LC in Figure 4-8.

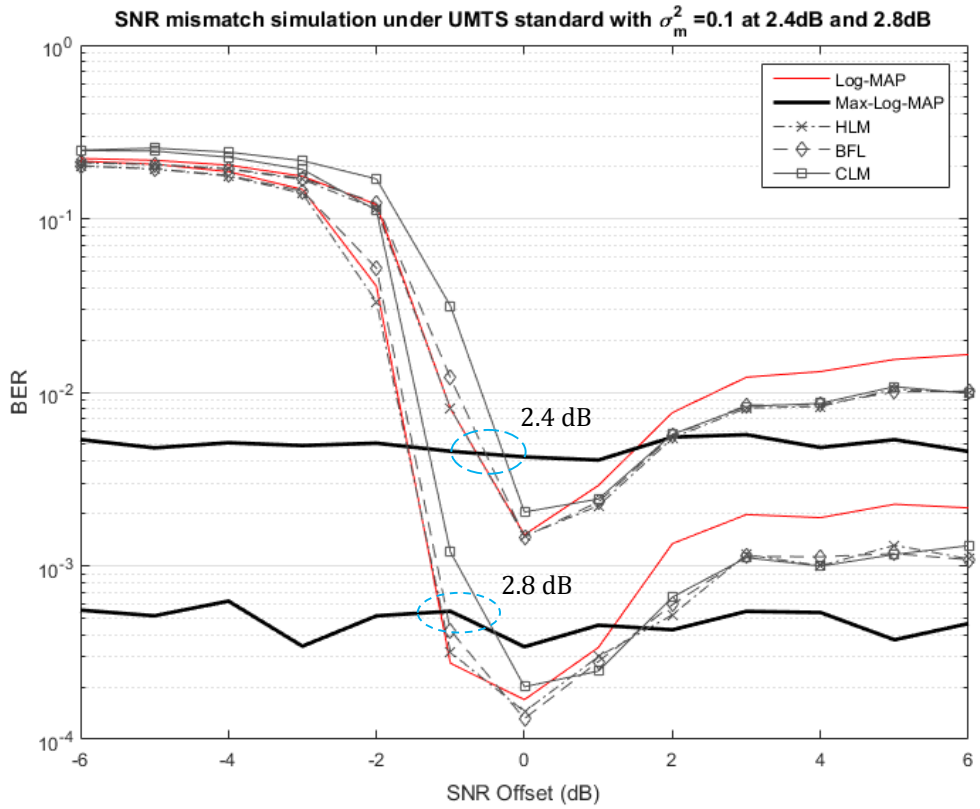


Figure 4-7 HLM, BFL, and CLM SNR mismatch simulation with $\sigma_m^2 = 0.1$ at true SNR of 2.4 dB and 2.8 dB

Figure 4-7 and Figure 4-8 are separated based on the groupings of the HLM, BFL, and CLM algorithm as well as LLM, MLM, and LC respectively against the Max-Log-MAP and Log-MAP solution to provide an clear visual presentation of simulation results. As shown in Figure 4-7, the performance of the BFL and HLM adheres closely if not, the closest, to the ideal Log-MAP performance when compared with the performance of the other approximations in Figure 4-8. As such, Figure 4-7 and Figure 4-8 should be viewed together for complete analysis. The remaining approximation algorithms are as shown in Figure 4-8 and the full analysis for both figures are provided following Figure 4-8.

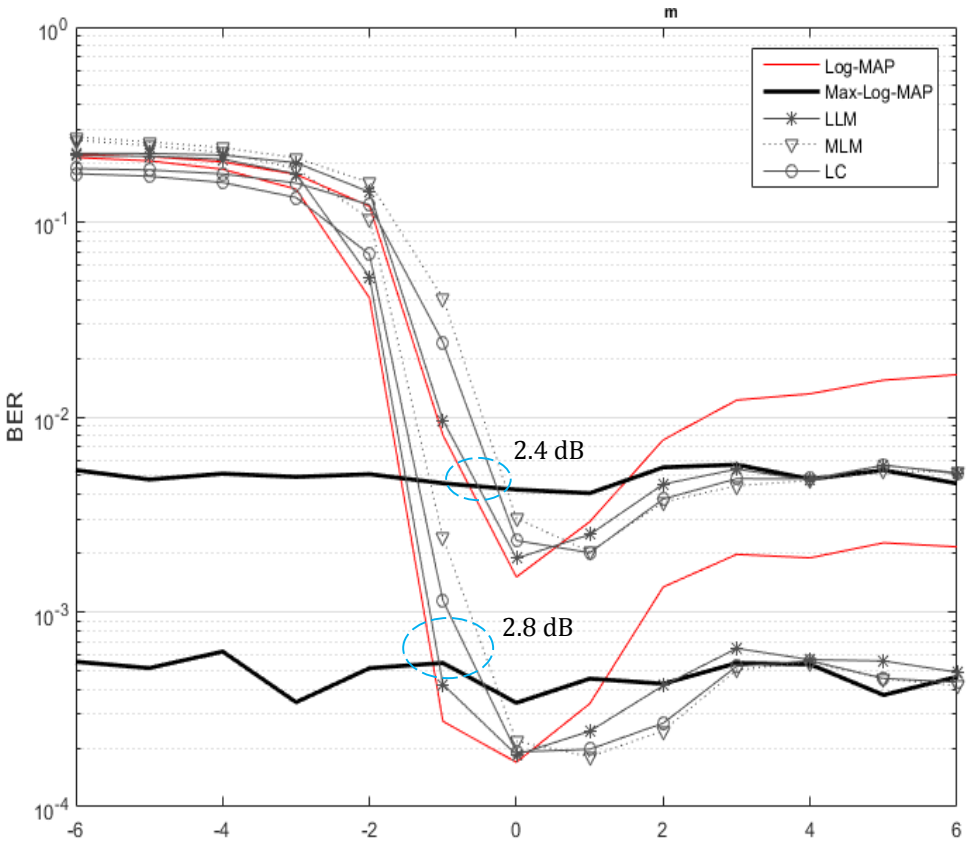


Figure 4-8 LLM, MLM, and LC SNR mismatch simulation with $\sigma_m^2 = 0.1$ at true SNR of 2.4 dB and 2.8 dB

Figure 4-7 and Figure 4-8 shows the BER performance of each approximation against the exact Log-MAP and Max Log-MAP solution. The Max Log-MAP solution is less sensitive to SNR offsets with steady BER measurement across all SNR offsets. This is consistent with reports found in literatures [86], [87], [89], [90].

The BER is at the lowest, forming a dip on the charts for each true SNR when the offset or mismatch is zero. Generally, positive SNR offsets are more tolerable than negative offsets for all results. This behaviour is also consistent with works reported in [86], [87], [89], [90]. The effects of negative offsets are more damaging as the decoder perceives the SNR to be lesser than the actual and causes slower convergence for the LLR values[123]. Large negative offsets of -6 dB causes the decoder to diverge and unable to correctly decode the received signals [123].

It is encouraging to note that the BFL and HLM algorithm follows most closely to the performance of the ideal Log-MAP solution and outperforms the majority of other suboptimal algorithms in the negative SNR offset regions. For positive SNR offsets, it is observed that all suboptimal algorithm performs better than the exact Log-MAP and converges to the performance of the Max Log-MAP consistent with the findings in [123]. This is because, for positive offset, the $|x|$ value in the correction function tends to larger values as with large positive SNR offsets, the channel reliability is higher than actual causing the difference between state metric to widen and $|x|$ to be large where the Jacobian logarithm approaches the Max Log-MAP solution with only max operation [123].

Based on the results presented, it can be concluded that overall, the BFL and HLM approximation adheres most closely to the ideal Log-MAP solution under channel and SNR misestimating conditions especially in negative SNR offset conditions.

A consolidated, and closer analysis of the performance of these algorithm are provided in Figure 4-9 and Figure 4-10 which shows all six algorithms (arranged consistently according to order in each graph legend; i.e. from left to right, in a group(HLM, BFL, CLM, LLM, MLM, and LC) BER differences towards the exact Log-MAP BER performance, plotted in group of six bar charts for each SNR offset according to each true SNR of 2.4 dB and 2.8 dB shown in Figure 4-7 and Figure 4-8.

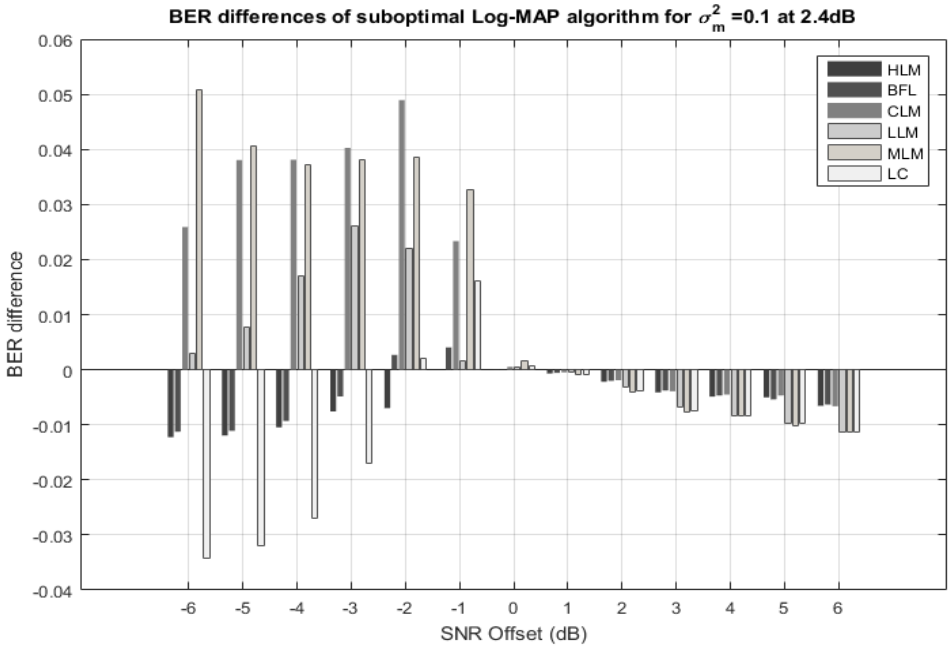


Figure 4-9 BER differences of suboptimal Log-MAP algorithms for $\sigma_m^2 = 0.1$ at 2.4 dB

The BER differences are computed by taking the difference in BER performance for each algorithm against the ideal Log-MAP performance in reference to Figure 4-7 and Figure 4-8 for true SNR at 2.4 dB and 2.8 dB in Figure 4-9 and Figure 4-10 respectively. Positive differences indicate values of SNR offsets where the suboptimal approximation records weaker performance than the ideal Log-MAP. A negative difference also indicates that the suboptimal Log-MAP algorithm concerned outperforms the exact Log-MAP decoder as reflected in the BER differences of Figure 4-9 and Figure 4-10. As previously mentioned, positive SNR offsets are generally tolerated better with small negative differences at 2.4 dB shown in Figure 4-9. These differences however, becomes less pronounced as the true SNR is increased to 2.8 dB as shown in Figure 4-10.

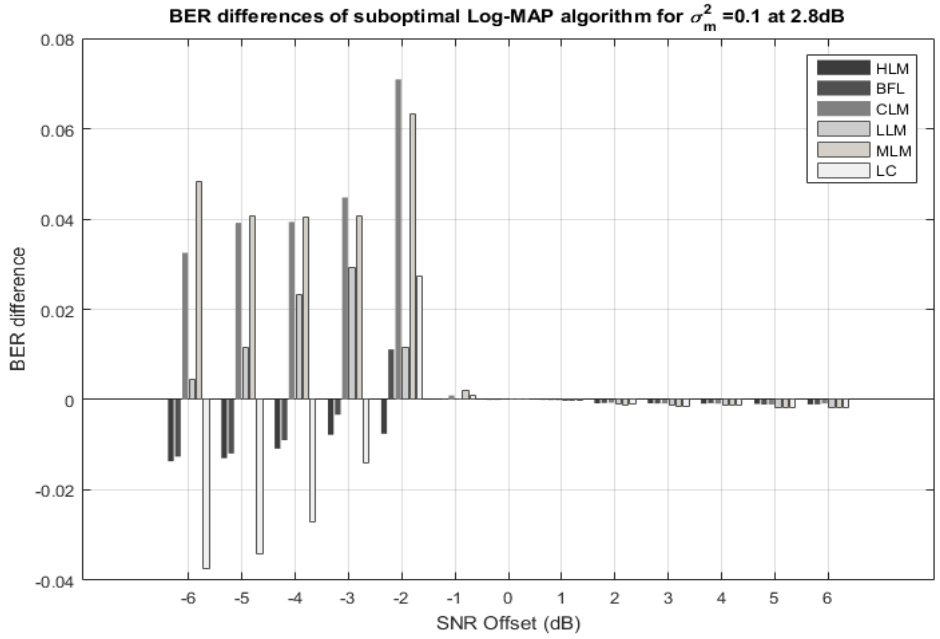


Figure 4-10 BER differences of suboptimal Log-MAP algorithms for $\sigma_m^2 = 0.1$ at 2.8 dB

Again, it can be observed that the BFL and HLM approximation follows the performance of the exact Log-MAP closely; and at the extremes of negative SNR offset, slightly outperforms the exact Log-MAP. Another suboptimal solution with encouraging results is the LC solution, slightly outperforming the exact Log-MAP decoder at large negative SNR offsets from -6 to -3 dB but quickly worsen with smaller negative offsets from -3 to 0 dB. However, this trend is less pronounced as the true SNR increased from 2.4 dB to 2.8 dB for negative SNR offsets.

With the analysis provided here, it is shown that the BFL and HLM adheres closely with the exact log-MAP solution in general and is able to deliver robust performance under SNR mismatch and channel coefficient inaccuracies compared with other suboptimal solutions.

4.3.1.1.2 Results with channel coefficient offset variance $\sigma_m^2 = 0.4$

Figure 4-11 and Figure 4-12 presents the results for when $\sigma_m^2 = 0.4$ with more severe inaccuracies to the estimated channel coefficients, $\widehat{h_k}$ at true SNRs of 6.4, 7, and 7.7 dB.

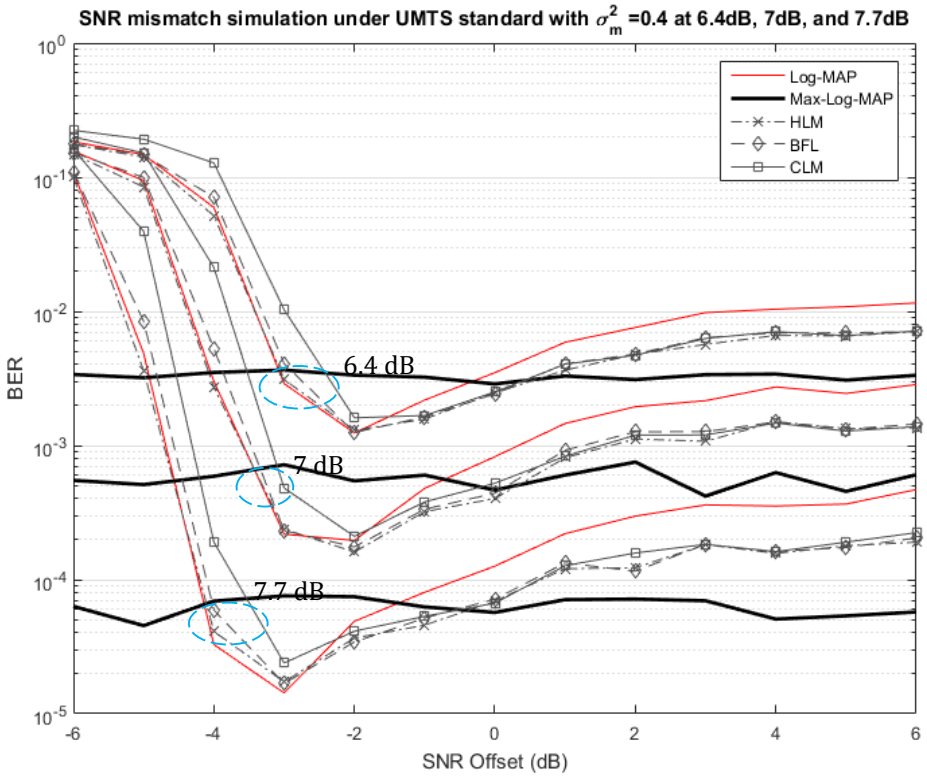


Figure 4-11 HLM, BFL, and CLM SNR mismatch simulation with $\sigma_m^2 = 0.4$ at true SNR of 6.4 dB, 7 dB, and 7.7 dB

Similar to the previous Section, Figure 4-11 and Figure 4-12 are separated based on the same groupings for a clear visual presentation of results. Consistent with the previous Section's analysis, the performance of the BFL and HLM continues to maintain close performance to the ideal Log-MAP BER performance. The performance for the corresponding LLM, MLM, and LC algorithm are presented in Figure 4-11. Likewise, the subsequent full analysis of BER performance difference under SNR mismatch for $\sigma_m^2 = 0.4$ at true SNR of 6.4 dB, 7 dB, and 7.7 dB is done in reference to both Figure 4-11 and Figure 4-12.

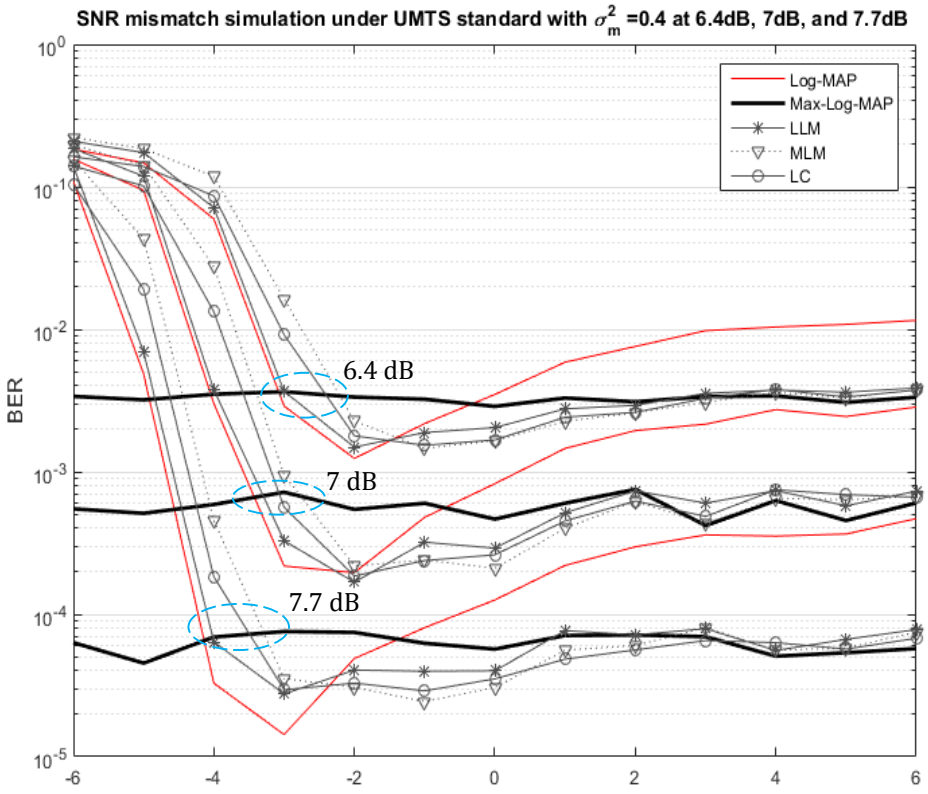


Figure 4-12 LLM, MLM, and LC SNR mismatch simulation with $\sigma_m^2 = 0.4$ at true SNR of 6.4 dB, 7 dB, and 7.7 dB

It is observed from both Figure 4-11 and Figure 4-12 that the SNR mismatch curve deviates towards the left of the chart forming dips in the (higher) negative SNR offset regions. The deviation is more pronounced with increasing true SNR. As will be shown in the results of the CCSDS decoder, the left shift of BER curve dip is more pronounced at higher true SNRs value and is not dependent on the channel coefficient estimates.

The BFL and HLM algorithm however, continues to closely follow the performance of the exact Log-MAP to give improved performance compared to other approximation algorithms. The trend of having smaller amplitude differences with increase of true SNR continues to prevail even at higher σ_m^2. As shown in Figure 4-13 , Figure 4-14, and Figure 4-15.

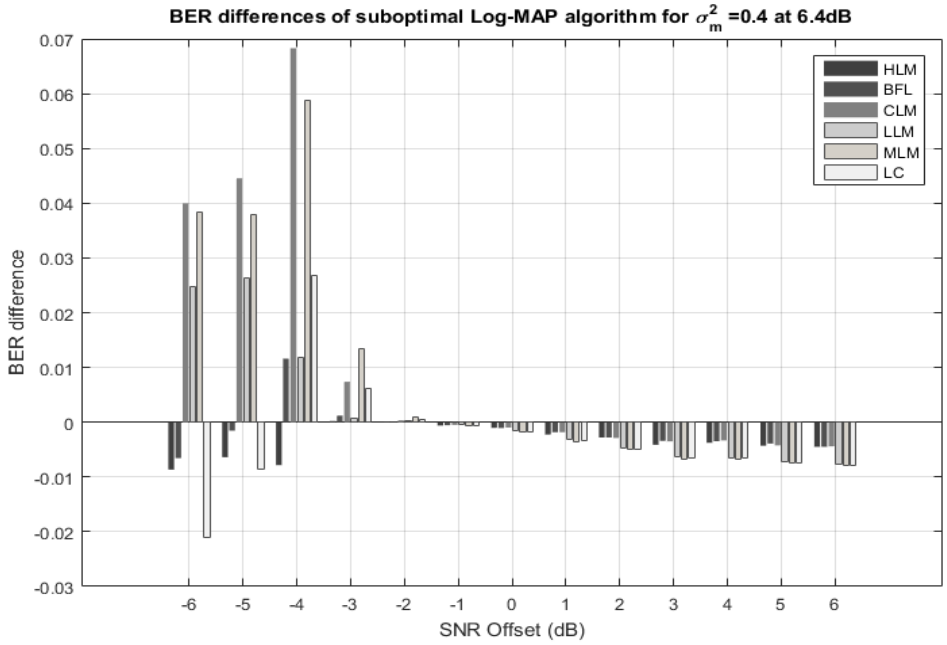


Figure 4-13 BER differences of suboptimal Log-MAP algorithms for $\sigma_m^2 = 0.4$ at 6.4 dB

Again, similarly with Section 4.3.1.1.1 the BER performance for each approximation are further detailed in the BER difference bar charts in Figure 4-13, Figure 4-14, and Figure 4-15 for $\sigma_m^2 = 0.4$ at 6.4 dB, 7 dB, and 7.7 dB respectively. Consistent with the observation made in the previous section, the BER performance at negative offset regions are less tolerable than positive offsets.

The BER curve dip has shifted further to the negative regions of SNR offsets which are less than -3 dB. This is illustrated in Figure 4-13 where the BER curve drops dramatically as the negative SNR offset values increases with the best BER performance recorded around SNR offset of -2 dB. The BER performance for positive offsets are consistent with the trend shown in the previous section as the decoder performs more resiliently in positive SNR mismatch.

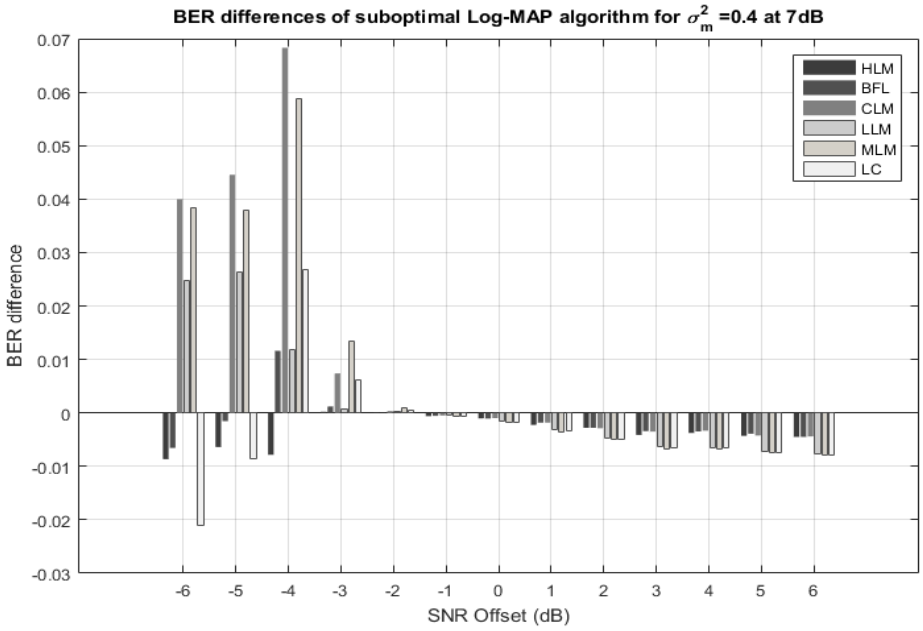


Figure 4-14 BER differences of suboptimal Log-MAP algorithms for $\sigma_m^2 = 0.4$ at 7 dB

Figure 4-14 illustrates the BER differences for each approximation algorithm for $\sigma_m^2 = 0.4$ at higher true SNR value of 7 dB. A similar BER curve dip is observed at SNR offset of -2 dB and the left shift of BER curve dip is also consistent with Figure 4-13. It is however noted that the true SNR difference between Figure 4-14 and Figure 4-13 is relatively small at 0.6 dB. Hence an additional experiment conducted at true SNR of 7.7 dB is presented in Figure 4-15.

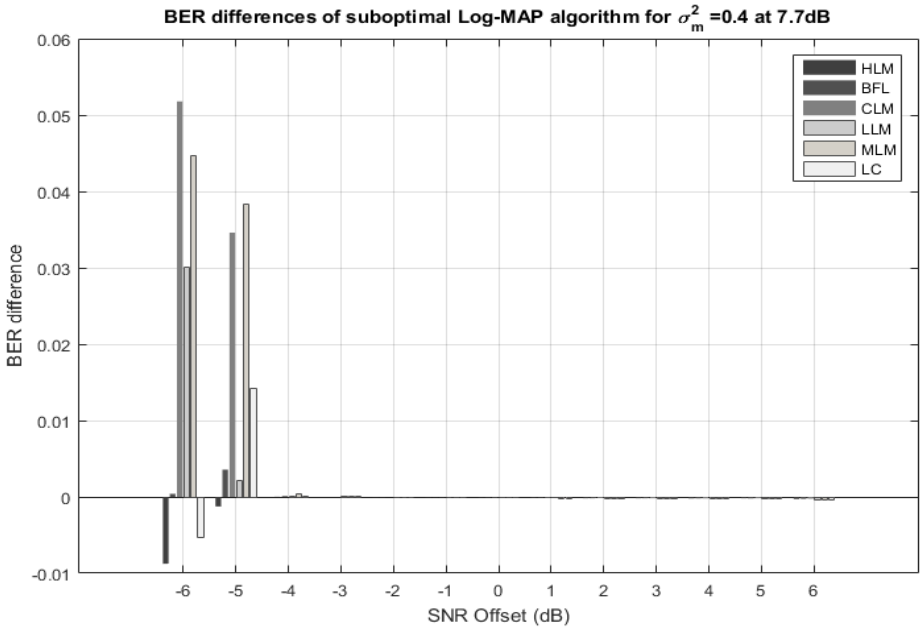


Figure 4-15 BER differences of suboptimal Log-MAP algorithms for $\sigma_m^2 = 0.4$ at 7.7 dB

Interestingly, at higher true SNR value of 7.7 dB, and at higher error in channel estimates, the left shift of BER curve dip is more pronounced, moving ever more into the negative SNR offset regions. This implies that a large negative offset is less serious at higher true SNR values as it would require a larger negative offset to result in worsened performances. In contrast, with lower true SNR values of 2.4 dB and 2.8 dB presented in Section 4.3.1.1.1, the BER curve dip is maintained consistently at 0 dB without the left shift as observed in the results of this section. Further experiments conducted in the next section with the CCSDS (16 state) Turbo decoder shall also indicate similar patterns of BER performances.

Based on the results presented for $\sigma_m^2 = 0.1$ and $\sigma_m^2 = 0.4$, it is observed that the BFL and HLM algorithms with good GOF statistics performs more resiliently in negative SNR offsets compared to their peers. The proposed suboptimal algorithms have recorded robust and encouraging results consistently with varying parameters presented.

4.3.2 SNR Mismatch Simulation under the CCSDS decoder

Consistent with other literatures implementing the CCSDS decoder [56] for deep space application, the communication channel is modelled with only AWGN to the channel without channel coefficients. A major difference between the 4G-LTE Advanced and the CCSDS Turbo decoder is the longer constraint length in the CCSDS Turbo decoder with 32 decoding states.

Similarly, for an organised presentation of results due to relatively high number of approximation results to present, the SNR offsets (representing SNR mismatch) graphs are as follows:

Table 4-4 Organisation of SNR mismatch results for the CCSDS Turbo decoder

SNR Mismatch Figures	True SNR values (dB)	Approximation Algorithm
Figure 4-16	0.3, 0.8, 1.0	HLM, BFL, CLM, Log-MAP, Max-Log-MAP
Figure 4-17	0.3, 0.8, 1.0	LLM, MLM, LC, Log-MAP, Max-Log-MAP
Figure 4-21	1.3, 1.5, 1.8	HLM, BFL, CLM, Log-MAP, Max-Log-MAP
Figure 4-22	1.3, 1.5, 1.8	LLM, MLM, LC, Log-MAP, Max-Log-MAP

4.3.2.1 Simulation Setup

The simulation setup for CCSDS SNR mismatch is the same as described in Table 4-2 with exception of SNR offsets implemented to simulate SNR mismatch. Figure 4-16 and Figure 4-17 presents the simulation results for the decoder with SNR mismatch at true SNRs of 0.3, 0.8, and 1.0 dB; as well as 1.3, 1.5, and 1.8 dB. The true SNR are selected via trial and error to adequately present distinct sets of BER results for ease of presentation. The MATLAB simulation codes for SNR mismatch using the CCSDS system simulation environment is provided in Appendix I.

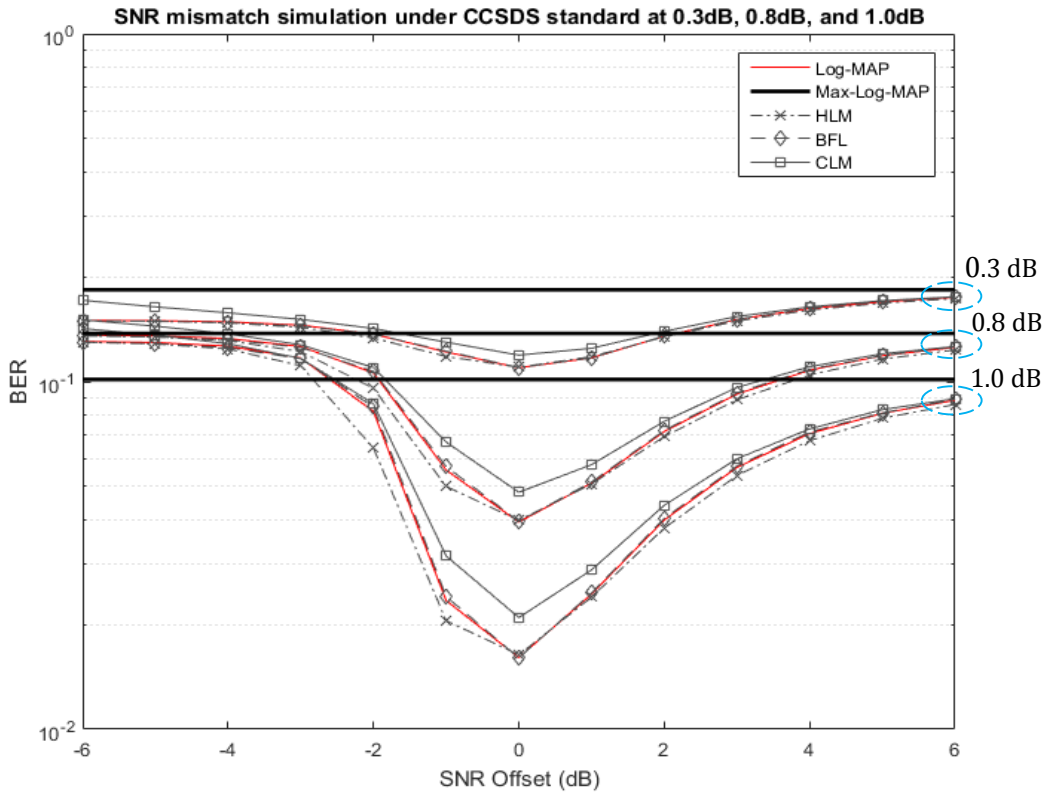


Figure 4-16 HLM, BFL, and CLM SNR mismatch simulation at true SNR of 0. 3 dB, 0.8 dB, and 1.0 dB

Figure 4-16 and Figure 4-17 are organised in the same manner as simulations under Section 4.3.1. The simulation results for all suboptimal approximations are presented under true SNR of 0.3, 0.8, and 1.0 dB. Generally, the received signals in deep space applications are frequently weak at low SNRs. Hence, the simulation presented for true SNRs in this section are focused on low values as previously mentioned. A difference in simulation done for deep space application is that there is no errors in channel estimates, $\sigma_m^2 = 0$ which implies a pure AWGN channel [56].

Based on Figure 4-16 and consistent with findings in the previous section, the BFL and HLM continues to display close adherence to the performance of the ideal Log-MAP. A consistent BER curve dip is observed at 0 dB SNR offset for all true SNR values simulated. The remaining approximation BER performance under SNR offset are presented in Figure 4-17.

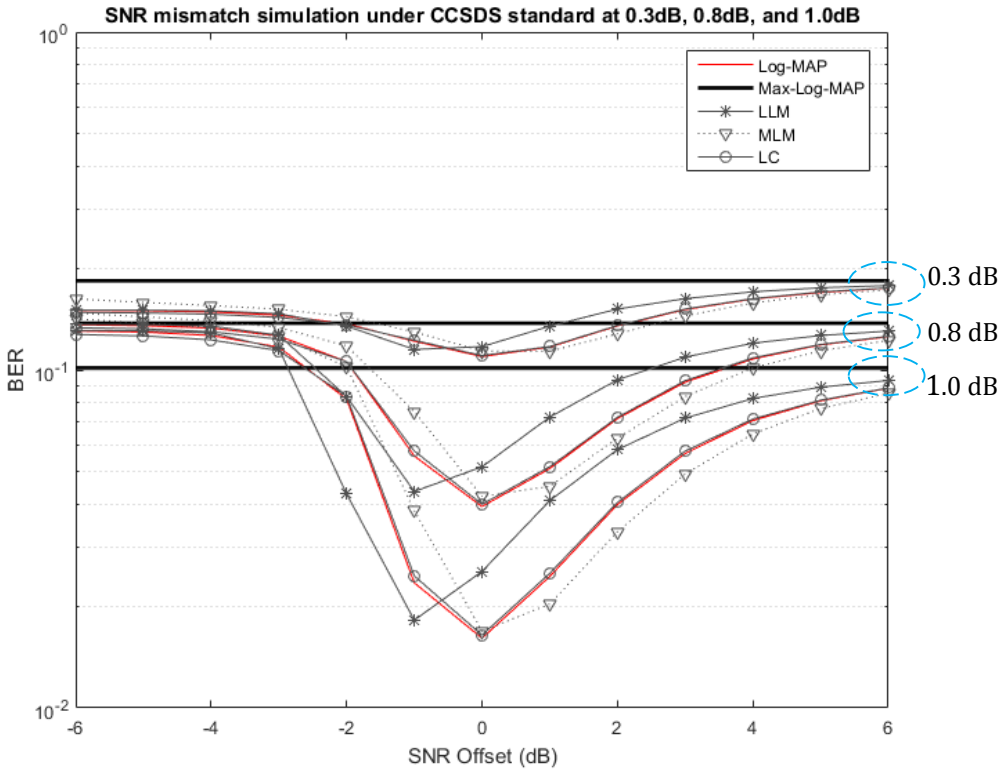


Figure 4-17 LLM, MLM, and LC SNR mismatch simulation at true SNR of 0. 3 dB, 0.8 dB, and 1.0 dB

Figure 4-16 and Figure 4-17 simulated at relatively low true SNR values shows a consistent BER curve dip at 0 SNR offset which is consistent with the findings in Section 4.3.1. It is observed in Figure 4-17 that the LLM approximation shows better resilience under negative SNR offset but comparatively poorer results in positive SNR offsets.

A closer analysis on the performance difference is provided in Figure 4-18, Figure 4-19, and Figure 4-20 for the corresponding true SNR of 0.3, 0.8, and 1.0 dB respectively provides further insights to this observation.

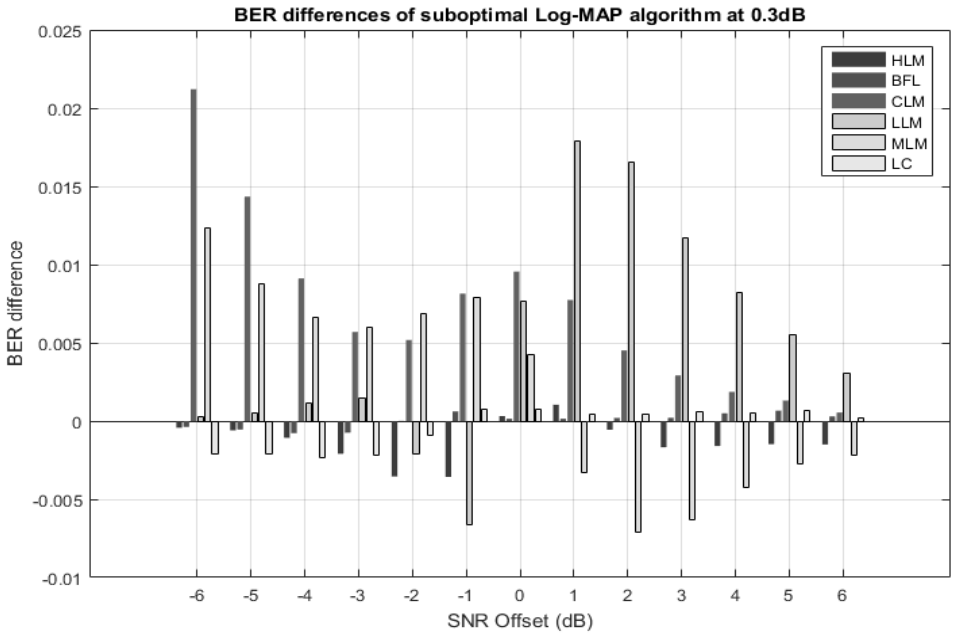


Figure 4-18 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 0.3 dB

At low SNRs, it is important to recall that the distribution of argument $|x|$ applied to each approximation will most likely fall on the lower values of the correction function where it is more prominent compared to the higher regions of $|x|$. As a result, the approximation algorithms that closely fits the ideal correction function will give the closest performance to the ideal Log-MAP solution. In Figure 4-18, with BER differences simulated at 0.3 dB true SNR, approximations that possesses close fit to the ideal correction function i.e. BFL, HLM, LLM shows the closest performance to the Log-MAP solution. However, for LLM in particular, the opposite is observed for positive SNR offset where it consistently performs the weakest as compared to other approximations. Under low true SNR, and compounded by negative SNR offset, the distribution of argument $|x|$ shifts further to the extreme smallest regions of x where the correction function plays the highest importance for computation accuracy. This results in observation made on the performance of LLM.

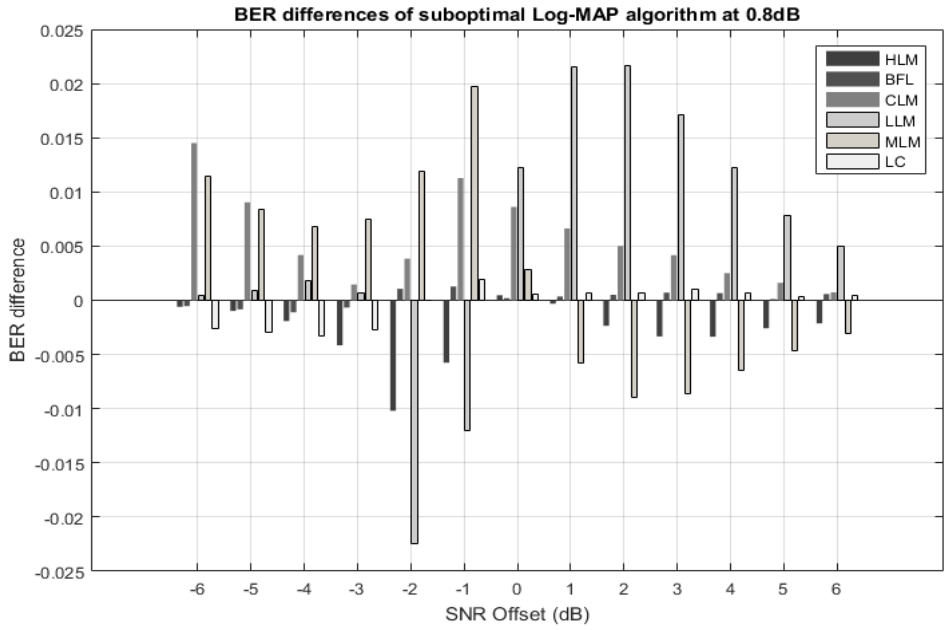


Figure 4-19 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 0.8 dB

The LLM, approximated using the MacLaurin series [48] gives great emphasis to the approximation for regions of $|x| \approx 0$, [67] and returns one of the closest fit to the correction function but suffers at higher regions of $|x|$ as it quickly approximates to zero which is more dominant at positive SNR offsets. Based on Figure 4-19, the trend observed for LLM is more prominent and is further amplified in Figure 4-20. This trend, to a certain degree, holds true for the HLM approximation too since the nature of HLM is similar to that of the LLM with closer approximation to the correction function in the smaller regions of $|x|$ but using crude, step approximation for larger values of $|x|$.

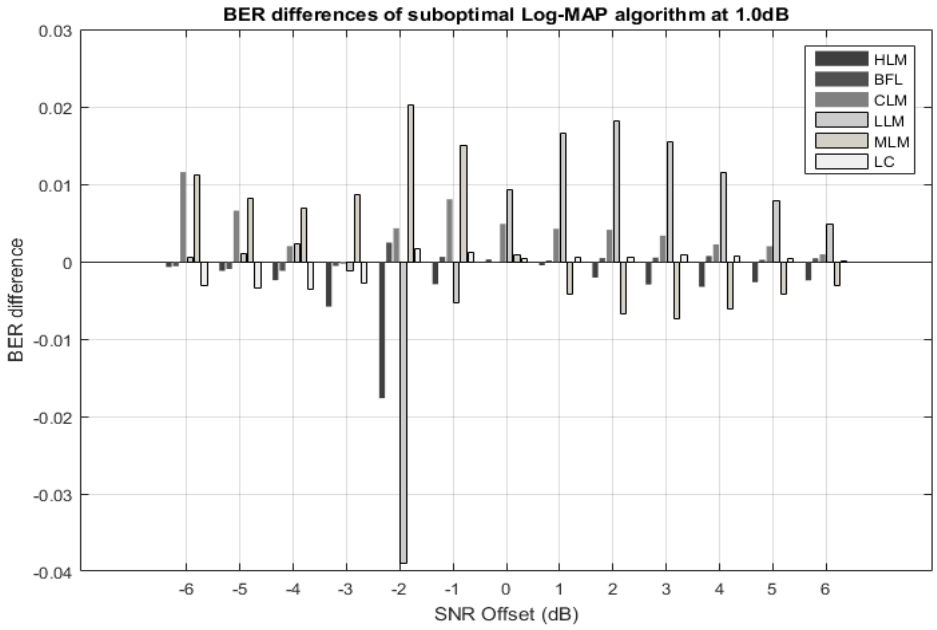


Figure 4-20 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 1.0 dB

Additional simulation results for true SNR of 1.3, 1.5, and 1.8 dB are also presented in Figure 4-21 and Figure 4-22 followed by their respective BER difference bar chart in Figure 4-23, Figure 4-24, and Figure 4-25.

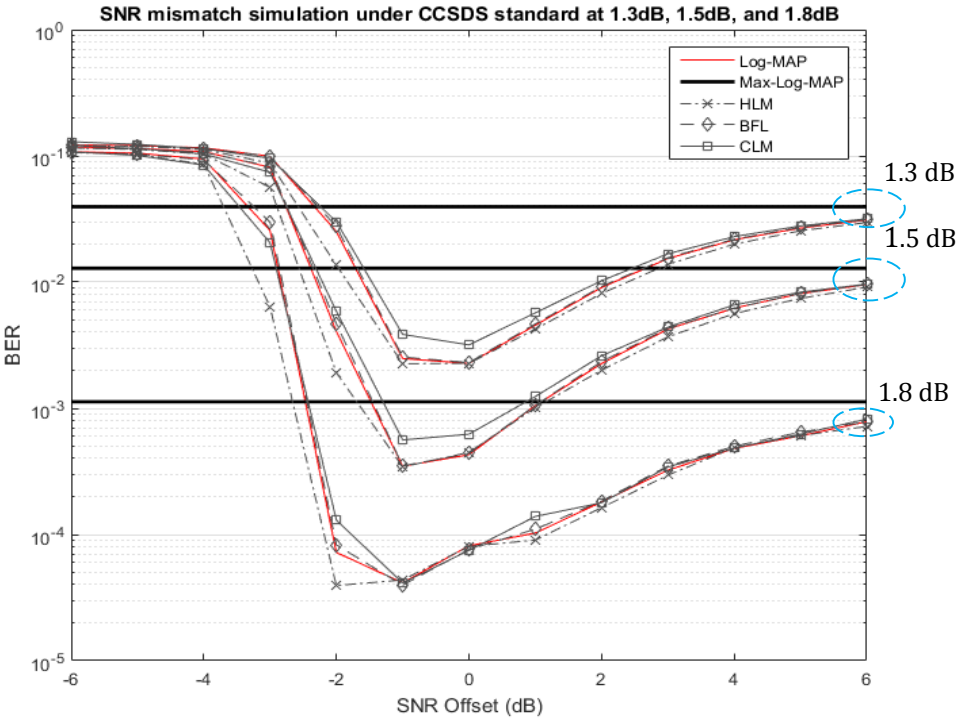


Figure 4-21 HLM, BFL, and CLM SNR mismatch simulation at true SNR of 1. 3 dB, 1.5 dB, and 1.8 dB

Figure 4-21 simulated at higher true SNR values of 1.3, 1.5, and 1.8 dB shows a continued consistent observation made for lower true SNRs with the BFL and HLM consistently recording close performances to the exact Log-MAP solution. As noted in Figure 4-18, Figure 4-19, and Figure 4-20, the BER performance for HLM and LLM (see Figure 4-22) in particular, is consistent with improved performance in the regions of negative SNR offset but relatively weaker performance at positive SNR offsets.

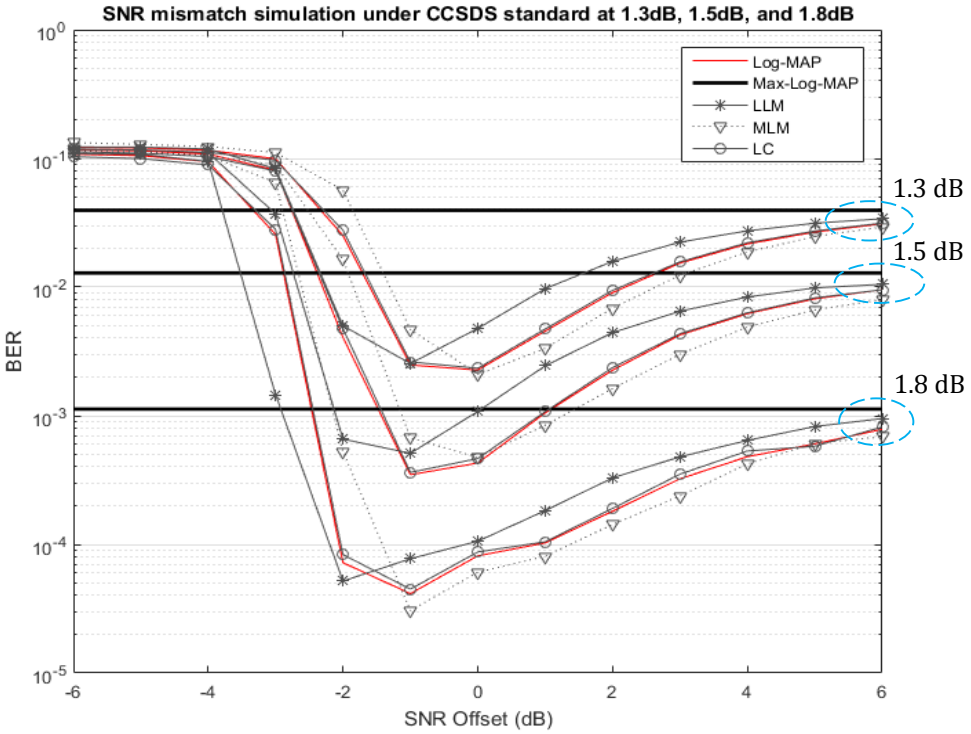


Figure 4-22 LLM, MLM, and LC SNR mismatch simulation at true SNR of 1. 3 dB, 1.5 dB, and 1.8 dB

The SNR mismatch simulation for LLM, MLM, and LC at relatively higher SNRs of 1.3 dB, 1.5 dB, and 1.8 dB are presented in Figure 4-22. The consistent pattern of left shift BER curve dip for higher relatively higher SNRs imply that with higher SNR, the decoder is more resilient toward negative SNR mismatch where the BER performance tends to worsen quickly.

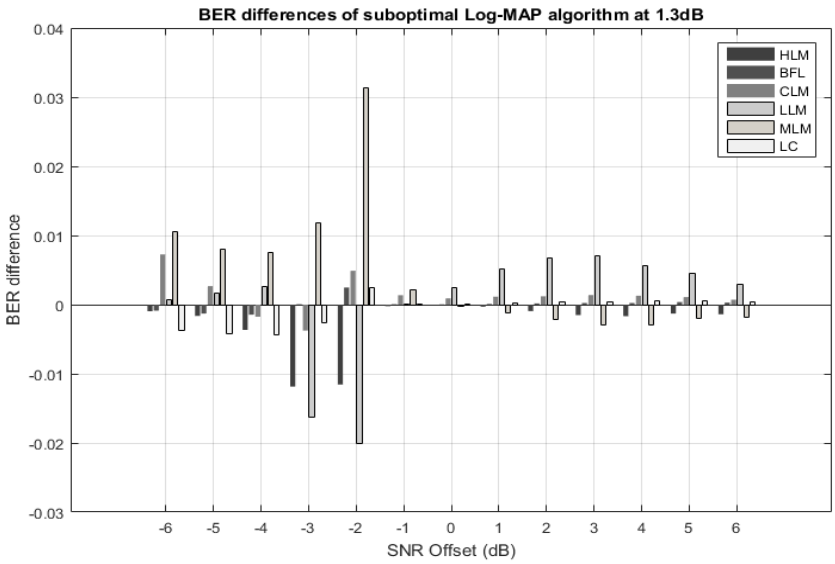


Figure 4-23 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 1.3 dB

Throughout the two sets of simulation results for true SNR of 0.3, 0.8, 1.0, 1.3, 1.5, and 1.8 dB; similar pattern of performance to the 4G standard can be observed. Generally, the suboptimal algorithm performances are more tolerant towards positive SNR offsets and worsens significantly in negative offsets. Consistent with findings in the 4G standard, the BER differences are also less prominent as the true SNR increases.

The HLM and BFL continue to record close performance to the exact Log-MAP with HLM slightly outperforming the exact Log-MAP at times in the negative SNR offset regions. In the CCSDS Turbo decoder, the MLM algorithm's optimal performance is deviated to the left in the negative SNR offset regions and at times, outperforms the exact Log-MAP solution.

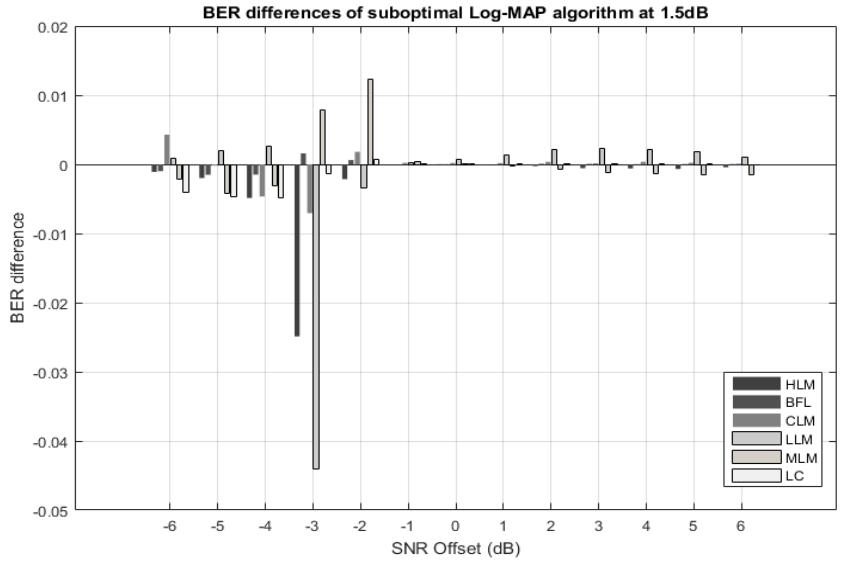


Figure 4-24 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 1.5 dB

Another important trend to note with the simulation results from Figure 4-23, Figure 4-24, and Figure 4-25 which is consistent with the results in Section 4.3.1 is the left shift in the BER curve dip with the increase of the true SNR values which are confirmed with the BER difference plots.

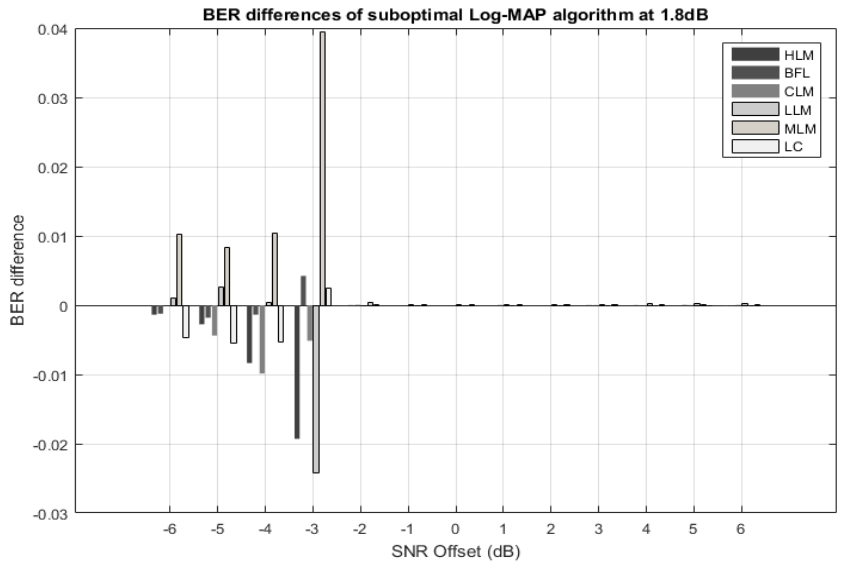


Figure 4-25 BER differences of suboptimal Log-MAP algorithms for the CCSDS decoder at 1.8 dB

Consistently, throughout the 4G-LTE Advanced as well as the CCSDS Turbo decoder, the HLM and BFL algorithm has shown overall good performances

especially at negative SNR offset regions. BER differences for each simulation scenario also confirms this finding with negative differences in these regions.

4.3.2.1.1 SNR mismatch simulation results for BFLN with the CCSDS Decoder

The BFL_N suboptimal solution is simulated under true SNR of 0.3, 0.8, 1.0, 1.3, 1.5, and 1.8 dB for SNR offsets from -6 to +6 as shown in Figure 4-26and Figure 4-27.

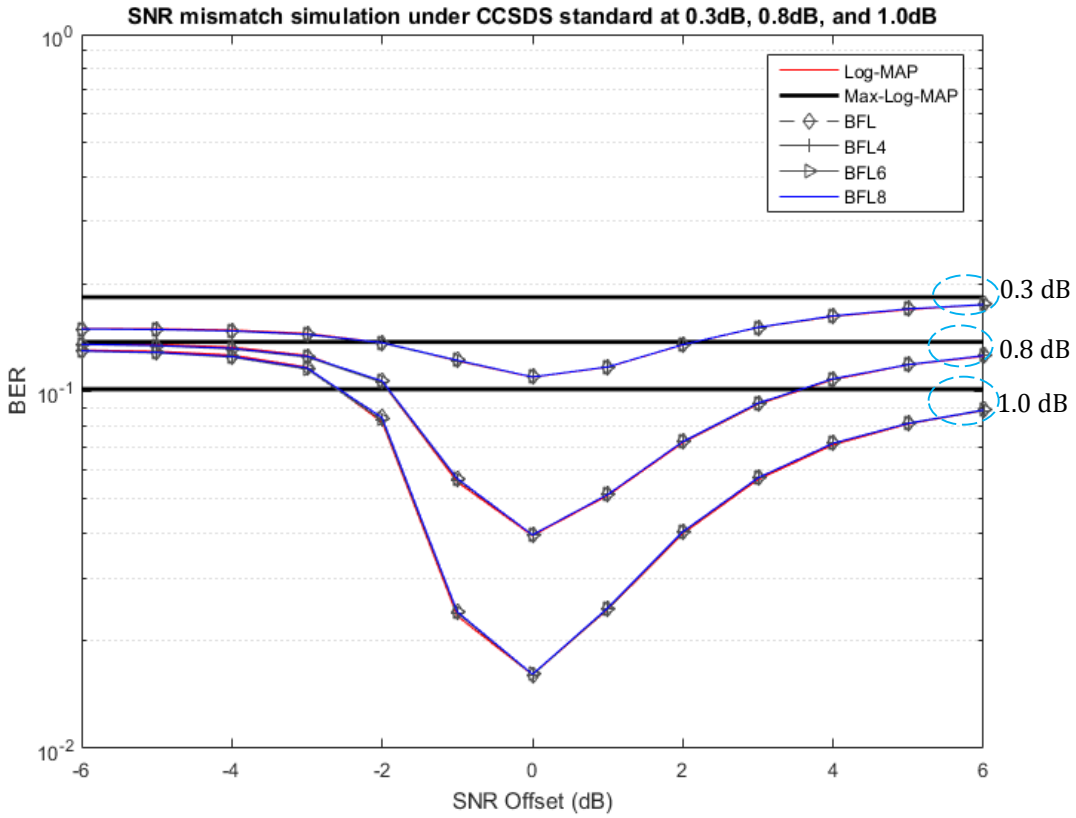


Figure 4-26 BFL_N SNR mismatch simulation at true SNR of 0. 3 dB, 0.8 dB, and 1.0 dB

The BFL_N simulations are repeated with the same setup as the CCSDS Turbo decoder. The CCSDS Turbo decoder was chosen as it is more sensitive to the approximation algorithm used in the correction function compared to the 4G-LTE Advanced (8 states) Turbo decoder. The results from both Figure 4-26 and Figure 4-27 is consistent with the floating point BFL approximation which closely follows the BER performance of the ideal Log-MAP and shows very little

differences with one another. Hence, the usage of any BFL_N approximation should bear minimal BER performance differences under SNR mismatch conditions.

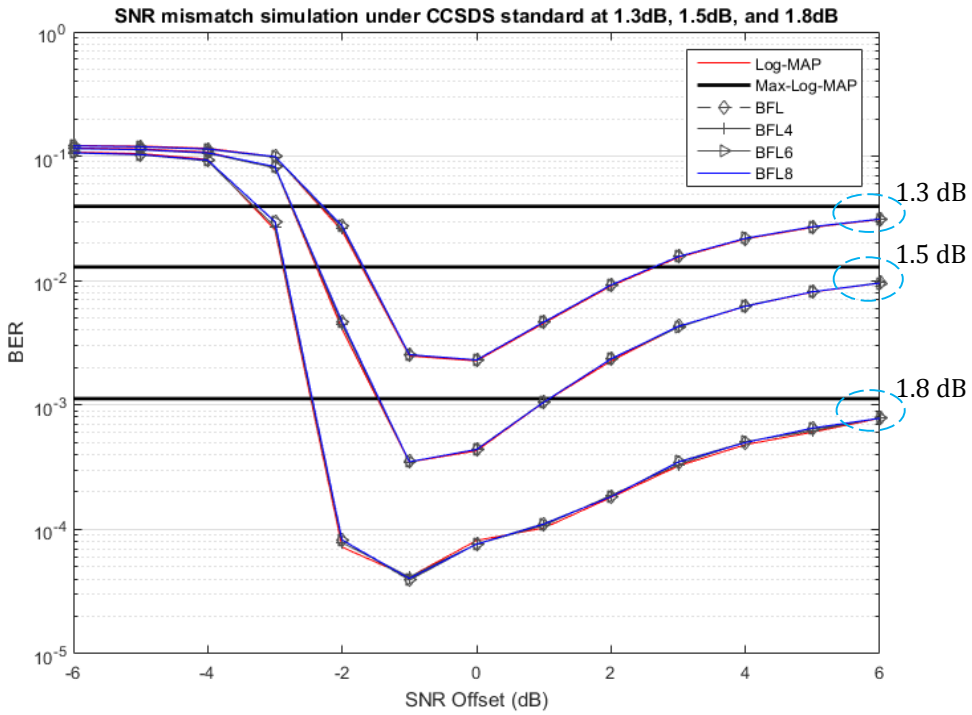


Figure 4-27 BFL_N SNR mismatch simulation at true SNR of 1. 3 dB, 1.5 dB, and 1.8 dB

Results from both figures shows very close adherence of the BFL and BFL_N with the exact Log-MAP solution with BER differences in the order of 10^{-4} between them. With closer inspection however, the performance can be ranked accordingly with the BFL giving the best performance, followed by BFL8, BFL6, and BFL4.

4.4 FPGA Implementation Evaluation for Proposed Approximations

This section evaluates the proposed approximation hardware implementation in terms of complexity, timing, and power analysis. As introduced in Chapter 2,

Section 2.3.2 the design effort was done in MATLAB/Simulink R2017a and VIVADO Design Suite 2017. Details of the FPGA board used are given in Chapter 3, Section3.5, Table 3-19. All analysis data are obtained from Vivado analysis report post implementation for accurate actual performance data on the FPGA.

First, the detailed hardware implementation for the HLM is presented, followed by the BFLN hardware implementation before a comparative study between the proposed approximations and other solutions in the literature are done towards the end of the section. Due to the lack of discussions in existing literatures on the implementation of suboptimal approximations on FPGA, an implementation for each suboptimal approximation was done in this research so that a fair comparison can be made. The design for each of these approximation is made to optimise for low complexity hardware implementation and for speed, focusing on low delay time. The optimisation of approximations in the literatures are done based only on the mathematical approximation given and not optimised for low radix fixed point processing to avoid quantization accuracy effects. Only the BFL 4,6, and 8 are designed to be fixed point processing friendly from the start which is not done by any works in the literature as yet.

4.4.1 HLM Approximation

The HLM approximation design detailed in Section 3.4.2 with XSG in Figure 3-31 are synthesised in Vivado and extracted here. Complexity, timing and power analysis were performed and presented in the following sub sections.

4.4.1.1 Complexity Analysis

The complexity analysis is generated from the utilization report which gives the resources used by the HLM design. The detailed utilisation report is as shown in Table 4-5.

Table 4-5 HLM utilization report

HLM Component\Resource	CLB LUTs	CLB Registers	CARRY8	CLB
Mux3	24	0	1	7
DSP48E2 Slice	1	38	0	8
Total	25	38	1	15

Based on the XSG design in Figure 3-31, the HLM only requires two main components, i.e. the multiplexer and the use of a single DSP48E2 to perform multiplication and addition operation for the linear region of HLM in smaller values of $|x|$.

The number of Control Logic Block (CLB) which is the fundamental building blocks for logic circuit in FPGA are used for Look-Up Tables (LUT), registers, and general logic operations. The CARRY8 block is part of the CLB which is tied to the Multiplexer (Mux). The DSP48E2 slice as explained in Section 0 is an efficient and optimised pre built module within the FPGA capable to run at the maximum system clock speed at 891 MHz. The use of the DSP48E2 slice is accompanied by 38 CLB registers as synthesised by Vivado through the design in XSG. With the use of the DSP48E2 slice, the HLM is clock dependent as will be detailed in the next section.

4.4.1.2 Timing Analysis

The HLM timing analysis report is detailed as follows:

Table 4-6 HLM timing analysis report

Design Timing Summary	HLM
Setup: Worst Negative Slack (WNS) (ns)	0.343
Hold: Worst hold Slack (WHS) (ns)	0.05
Pulse width: Worst Pulse Width Slack (WPWS) (ns)	0
Minimum Period (ns)	1.122
Maximum Frequency (MHz)	891

The HLM, although slightly more complex due to the use of the DSP48E2 slice, is capable of running at 891 MHz maximum speed which is the same as the FPGA board's master clock for logic circuits as detailed in Section 0. Fundamentally, the slack is the difference between the expected arrival time of a signal with its actual arrival time. The HLM timing analysis generally details the setup time (maximum delay analysis), hold time (minimum delay analysis), and worst pulse width slack (pin switching limits). The worst negative slack (WNS) in Vivado indicates the worst slack of all the timing path for the circuit and a positive value for WNS indicates that the path passes the timing constraints in Vivado which was set at the maximum clock speed of 891 MHz. Meanwhile the worst hold slack (WHS) indicates the worst slack of all timing paths for minimum delay analysis. Again, a positive value indicates that the design meets the timing constraint set. The pulse width analysis checks the minimum low and high pulse width, minimum and maximum period required. The worst pulse width slack (WPWS) shows the worst slack of all timing checks with both minimum and maximum delays. All definitions, and details of timing analysis in this section can be referred from Xilinx's user guide in [124].

Faster clock speed as timing constraints smaller than the minimum period required in Vivado has returned violations of timing constraints. Hence the

maximum clock frequency that the HLM requires to function is 1/1.122 ns which translate to be 891 MHz.

4.4.1.3 Power Analysis

The power analysis for the HLM is detailed as follows:

Table 4-7 HLM power analysis

Resource	HLM
DSP48E2 (W)	0.011
Mux3 (W)	0.004
Total Power for circuit (W)	0.015
Total module power with Gateway (W)	0.302
Total On chip Power (W)	2.874

The power analysis in Vivado is broken up according to components as shown inTable 4-7 for the Mux and DSP48E2 usage each. Totalled up, the total power consumed by the HLM logic circuit is 0.015 W with a total on chip power of 2.874 W reported for the HLM implementation inclusive of power utilisations by the gateways and leaf cells.

4.4.2 BFL Approximation

In this section, the hardware implementation analysis for the BFL 4, 6, and 8 are presented in like manner to the HLM in the previous section. It should be highlighted that although the BFL was designed to be implemented with linear multiplication and addition operation initially, with the implementation of numerator adjustments and exploiting division by 2 in hardware as mere radix point bit shifts as detailed in Table 3-17, the BFLN algorithm has manage to completely avoid any multiplication operation and thereby reducing complexity to greater lengths. The positive effect not only extends to complexity but also timing since the BFL can be configured as combinational logic to allow clock independent operation limited only by its path delays.

4.4.2.1 Complexity Analysis

The utilization report for the BFLN are as follows:

Table 4-8 BFL4 utilization report

BFL4 Component\ Resource	CLB LUTs	CLB Registers	CARRY8	CLB
Addsub	7	0	2	4
Addsub1	9	0	2	3
Addsub2	14	0	2	5
Mux	3	6	0	2

Table 4-9 BFL6 utilization report

BFL6 Component\ Resource	CLB LUTs	CLB Registers	CARRY8	CLB
Addsub	10	0	2	4
Addsub1	12	0	2	3
Addsub2	5	0	2	4
Addsub3	14	0	2	3
Mux	4	8	0	2

Table 4-10 BFL8 utilization report

BFL8 Component\ Resource	CLB LUTs	CLB Registers	CARRY8	CLB
Addsub	11	0	2	6
Addsub1	19	0	2	4
Addsub2	9	0	2	5
Addsub3	14	0	2	4
Addsub4	12	0	3	4
Mux	4	10	0	5

The complexity of BFL 4 to 8 increases as expected due to increased use of radix point for accuracy of computation. This trend is clearly reflected throughout Table 4-8, Table 4-9, Table 4-10 and summarised in Table 4-11.

Table 4-11 Summary of utilization report for BFLN

Total Resources	BFL4	BFL6	BFL8
CLB LUTs: Logic	33	45	69
CLB Registers : Flip Flops	6	8	10
CARRY8	6	8	11
CLB	14	16	28

The utilisation report for the BFLN approximations along HLM and other solutions are provided in Section 4.5.

4.4.2.2 Timing Analysis

The timing analysis for each BFLN approximation is provided in Table 4-12.

Table 4-12 BFLN Timing analysis

Design Timing Summary	**BFL4/6/8**
Setup: Worst Negative Slack (WNS) (ns)	NA
Hold: Worst hold Slack (WHS) (ns)	NA
Pulse width: Worst Pulse Width Slack (WPWS) (ns)	0
Minimum Period (ns)	NA
Maximum Frequency (MHz)	891

The BFLN consists on purely combinational logic circuitry and due to this, the setup, hold, as well as pulse width analysis is not applicable (NA). With the case of combinational logic circuits, the clock speed is only limited by the maximum propagation delay through the circuit which is smaller than shortest system clock half cycle period. The timing analysis tool based on the maximum frequency set of 891 MHz meets all timing constraint with 0 WPWS. The same is also observed for the CLM algorithm which is purely combinational consisting of the LUT.

4.4.2.3 Power Analysis

The BFLN's power analysis breakdown is summarised in Table 4-13 as follows:

Table 4-13 BFLN power analysis

Resource	BFL4	BFL6	BFL8
Addsub (W)	<0.001	<0.001	<0.001
addsub1 (W)	<0.001	<0.001	<0.001
addsub2 (W)	<0.001	<0.001	<0.001
Addsub3	Not applicable	<0.001	<0.001
Addsub4	Not applicable	Not applicable	<0.001
Mux (W)	0.001	0.001	0.001
Total Power for BFL circuit (W)	0.001	0.001	0.001
Total module power with Leaf Cells and Gateway (W)	0.004	0.004	0.004
Total On chip Power (W)	2.571	2.571	2.571

The total power utilisation for BFLN circuit for all BFL fixed point approximations are at 0.001 W which is much smaller compared to the HLM at 0.015 W. This is due to the much lower complexity of the BFLN circuitry that utilises purely combinational circuitry which avoids the multiply and add operation using the DSP48E2. Although highly optimised in the FPGA, the DSP48E2 still takes up considerable resources and with that, requires increased power utilisation.

4.5 Comparative Study between Proposed Approximations and Existing Solutions

As explained in earlier sections, in order for a fair comparison between approximation design methods, each of the approximation is also implemented in XSG with design as specified in Appendix L.

4.5.1.1 Complexity Analysis

The complexity analysis results are obtained from the utilisation report generated for each suboptimal solution, summarised in Table 4-14.

Table 4-14 VIVADO resource utilisation report

FPGA Resource Utilisation	CLM	LLM	MLM	LC	HLM	BFL4	BFL6	BFL8
CLB LUTs: Logic	2	24	10	3	25	33	45	69
CLB Registers : Flip Flops	0	7	21	0	38	6	8	10
CARRY8	0	2	1	2	1	6	8	11
CLB	1	6	3	4	15	14	16	28
DSP48E2s (Multiply Add)	0	1	0	4	1	0	0	0
Constant multiplier	0	1	0	2	0	0	0	0

In order to fairly compare the suboptimal approximations, the approximations are divided into those that require the DSP48E2 slice and those who do not. This is because, with the use of DSP48E2 slice, the relative complexity rises due to pre-built logic designs that comes with it and CLB units required to support it.

The order of complexity (from least to most complex) for approximations that utilises the DSP48E2 is the LLM, followed by the HLM, and the LC approximation. The LC approximation incurs higher usage of DSP48E2 due to constant multiplications as referenced to Appendix L.

For approximations that does not utilise the DSP slice, and depends on simple combinational circuits as well as registers, the complexity is reduced. The order of complexity (from least to most complex) for such designs are as follows: CLM, MLM, BFL4, BFL6, and BFL8. However, an important feature of the BFLN approximations to note is not just reduced hardware complexity, but also accuracy in approximating the exact correction function to give the closest performance to the Log-MAP solution under the 4G-LTE Advanced and CCSDS standards in perfect channel estimation as well as channel mismatch scenarios as detailed in the previous section.

4.5.1.2 Timing Analysis

The overall timing analysis results for various suboptimal Log-MAP solutions are provided in Table 4-15.

Table 4-15 VIVADO timing summary report

Design Timing Summary	CLM	LLM	MLM	LC	HLM	BFL4, BFL6, BFL8
Setup: Worst Negative Slack (WNS) (ns)	NA	0.238	0.391	0	0.343	NA
Hold: Worst hold Slack (WHS) (ns)	NA	0.151	0.066	0.1	0.05	NA
Pulse width: Worst Pulse Width Slack (WPWS) (ns)	NA	0	0	0.685	0	0
Minimum Period (ns)	NA	1.122	1.122	1.235	1.122	NA
Maximum Frequency (MHz)	NA	891	891	810	891	891

Due to the combinational nature of the CLM and BFLN approximations, the timing for these approximations are limited only by their propagation delay and with the constraint of the timing analysis tool within Vivado that requires a clock frequency to be set, both circuits meets the timing constraint to operate at the maximum frequency possible at 891 MHz which is also well over the 500 MHz threshold required for 5G [4]. It is also possible for these algorithm to operate at much higher frequencies, however, considering the limits of the chosen platform which is already catered to the fastest speed grade at -3 specified in Table 3-19, the maximum speed that is allowed for both sets of approximation is 891 MHz.

Other suboptimal approximations that are capable to operate at maximum speed are the HLM, MLM, and LLM. The LC approximation requires a slightly lower clock speed at 810 MHz in order to not violate any timing constraints. To sum up, the logic circuit rank from fastest to slowest are as follows: CLM, BFL4, BFL6, BFL8, LLM, HLM, MLM, and LC. The LLM is considered relatively faster than the HLM or the MLM due to the number of regions for $|x|$ that is firstly detected through the comparator followed by bit shifts for the HLM (lesser bit

shifts) and the MLM(more bit shifts). The slowest among these is the LC implementation which requires comparators and a few multiplication operations.

4.5.1.3 Power Analysis

The power utilisation report for all approximations are detailed in Table 4-16.

Table 4-16 VIVADO power summary report

Resource	CLM	LLM	MLM	LC	HLM	BFL4	BFL6	BFL8
Total Power for circuit (W)	0.007	0.009	0.004	0.002	0.015	0.001	0.001	0.001
Total module power with Leaf Cells and Gateway (W)	0.023	0.086	0.054	0.036	0.302	0.004	0.004	0.004
Total On chip Power (W)	3.481	2.655	2.622	2.603	2.874	2.571	2.571	2.571

In terms of power, the BFLN with radix point implementation records the lowest power utilisation at 0.001 W among all approximations.

The next lowest power utilisation falls to the LC at 0.002 W utilising the optimised DSP48E2 slice, followed by the CLM approximation at 0.01. Although the simplest, the CLM records a higher power utilisation count as the use of comparators, and multiplexers are not optimised for fixed point FPGA applications and require larger sizes of LUT, comparators, and multiplexers although the recorded complexity count is lesser for the CLM.

The remainder approximation power utilisation rank are as follows: MLM, LLM and HLM. Although the HLM operates at higher power utilisation, the BER performance under perfect channel knowledge and SNR mismatch closely adheres to the exact Log-MAP solution.

4.6 Soft Bit Demodulation Simulation Results

As explained in Section 2.4, the log exponential sum also occurs in M-QAM soft bit demodulation. This section presents the BER simulation results for soft bit demodulation. In this section is noted that in all experiments in Section 4.6.1 and Section 4.6.2, used in the Turbo decoder employs the Max-Log-MAP approximation to save simulation time and that the only difference between each curve for each figure, is the use of either Max-Log-MAP, exact Log-MAP or the BFL approximation at the soft bit demodulation portion.

4.6.1 Performance of M-QAM soft bit demodulation with 8 states Turbo decoder

The M-QAM soft bit demodulation for 4G-LTE Advanced Turbo decoder based on the discussed method of soft demodulation in Chapter 2, Section 2.4 are presented in this section.

4.6.1.1 Simulation Setup

The simulation setup is summarised by Table 4-17 as follows:

Table 4-17 Soft bit demodulation simulation setup for 4G-LTE Advanced Turbo decoder

Parameter	Measure
Turbo code polynomial Generator	g[13,15]
Turbo encoder constraint length	4
Number of Turbo decoder iterations	10
Iteration stop criteria	Magic genie
Code rate	1/2
Channel type	Rayleigh, AWGN
M-QAM modulation	256,1024
Interleaver type (Size)	S-random (1360,1700)
Number of Frames	10,000
Total bits simulated	10,200,000

Most of the simulation parameters setup is similar to Table 4-1, but with differences in the modulation order and soft bit demodulation methods to cater to M-QAM soft bit demodulation for Turbo decoding. Results in this research shall be focused on square M-QAM with 256 and 1024-QAM which is the most widely targeted goal for 5G [36], [69], [70]. The results for 256-QAM are as follows:

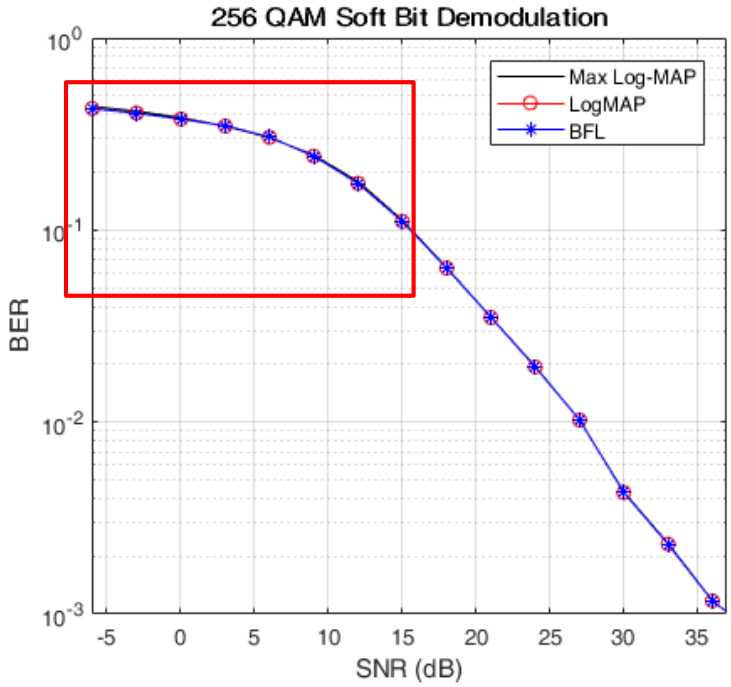


Figure 4-28 256-QAM soft bit demodulation BER for 4G-LTE Advanced Turbo decoder

Figure 4-28 shows the 256-QAM soft bit demodulation BER performance for the 8 states Turbo decoder. Generally, for the 4G-LTE Advanced Turbo decoder, the BER performance between the BFL, Max-Log-MAP, and ideal Log-MAP are close to each other with exception to regions with very weak SNR $\leq$ 0 dB where the section highlighted in red within Figure 4-28 is enlarged in Figure 4-29.

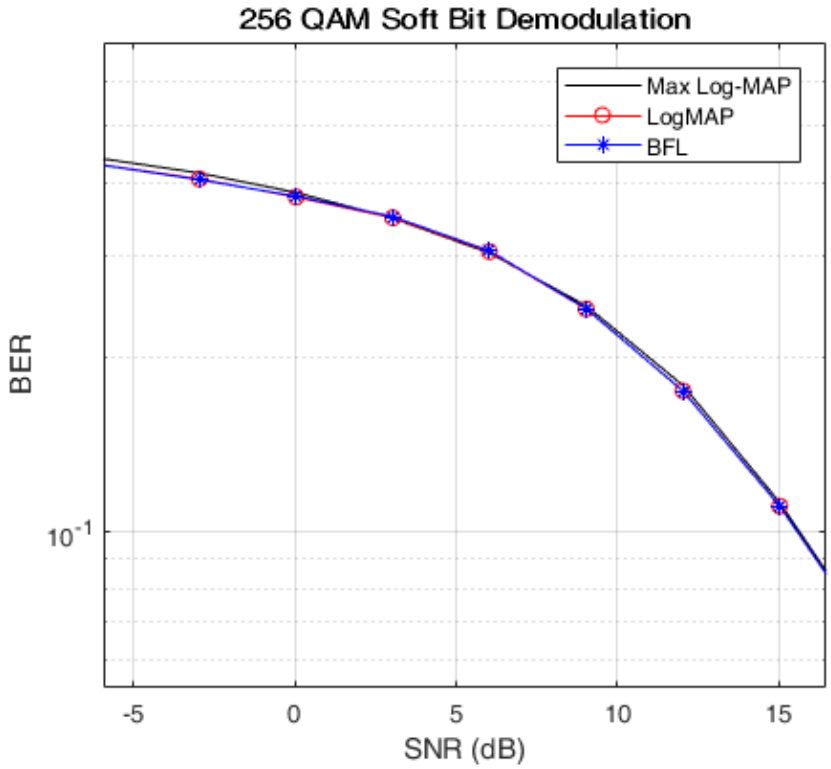


Figure 4-29 Enlarged highlighted area of 256-QAM soft bit demodulation BER for 4G-LTE Advanced Turbo decoder

In general there is only a slight performance gap between the Max-Log-MAP and the exact Log-MAP for the 4G-LTE (8 states) Turbo decoder. Although the approximation is applied to the soft decoding of 256 QAM, only a slight difference is detected at very weak SNRs as highlighted previously. Expectedly, there exists a performance gap between the Max-Log-MAP and the ideal Log-MAP solution. It is encouraging to note that the BFL continues to offer the closest performance to the exact Log-MAP particularly in low SNR conditions.

In order to further investigate the effect of QAM soft bit demodulation, the experiment is repeated with 1024 QAM. The results for the 1024-QAM is presented next in Figure 4-30.

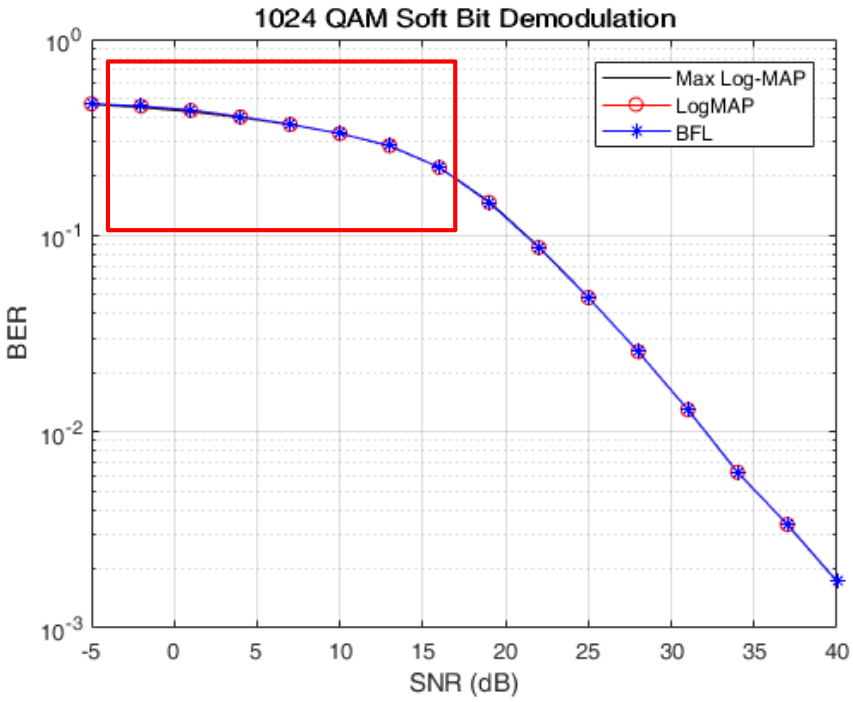


Figure 4-30 1024-QAM soft bit demodulation BER for 4G-LTE Advanced Turbo decoder

It is observed from Figure 4-30, that the BER performance continues to give high similarity between the Max-Log-MAP, ideal Log MAP, and the BFL. At low SNRs however, the performance difference between the Max-Log-MAP and ideal Log-MAP solution is visible as shown in Figure 4-31.

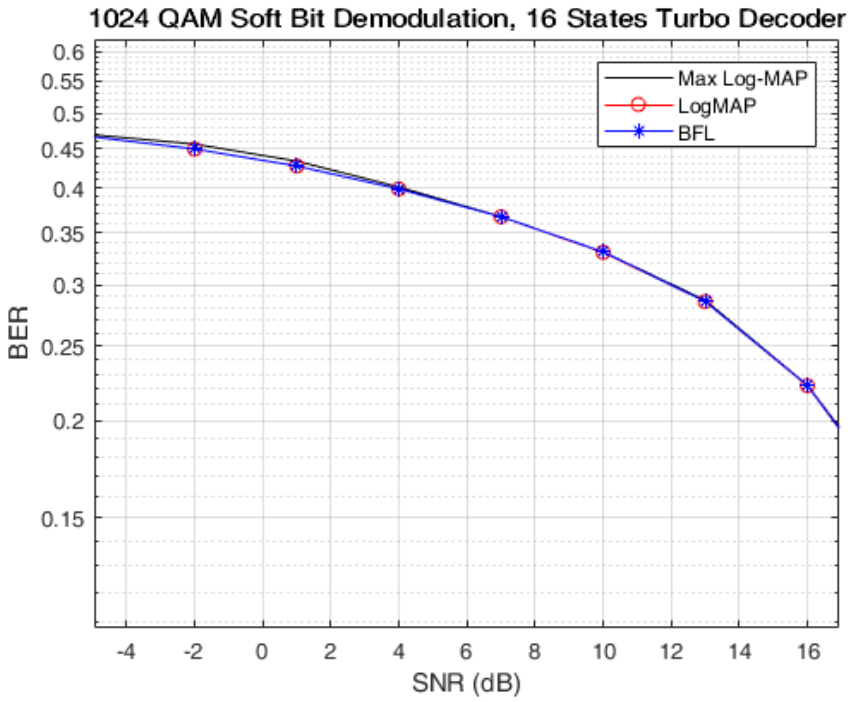


Figure 4-31 Enlarged highlighted area of 1024-QAM soft bit demodulation BER for 4G-LTE Advanced Turbo decoder

As shown in Figure 4-31, the performance gap between the Max Log-MAP and exact Log-MAP remains very close. Again, the BFL offers slight performance improvement over the Max Log-MAP. In order to investigate the effect of employing the BFL approximation to soft bit demodulation, the next section presents simulation results for soft bit demodulation with the CCSDS Turbo decoder (16 states) where it is more sensitive to the accuracy of the correction function.

The results in both 256 and 1024-QAM for higher SNR situations is consistent with findings in the literature that says the Max Log-MAP performance approaches the exact Log-MAP BER performance [125]. The BFL recorded performance difference within 1.6×10^{-3} and 2.7×10^{-3} dB respectfully to the 256 and 1024-QAM BER performance.

4.6.2 Performance of M-QAM soft bit demodulation with 16 states Turbo decoder

In the CCSDS standard, the maximum M-QAM order is set at 256-QAM [54]. As such, the results for the 16 and 256-QAM which are the often used in the standard are presented in the next sections.

4.6.2.1 Simulation Setup

The CCSDS simulation setup for soft bit demodulation is given in Table 4-18

Table 4-18 Soft bit demodulation simulation setup for CCSDS Turbo decoder

Parameter	Measure
Turbo code polynomial generator	g[23,33]
Turbo encoder constraint length	*5*
Number of Iterations	5
Iteration stop criteria	Magic genie
Code rate	1/2
Channel type	AWGN
M-QAM modulation	16, 256
Interleaver (Size)	Pseudo random (1000)
Number of Frames	10,000
Total bits simulated	10,240,000

The setup remains consistent with previous simulation setup for CCSDS Turbo decoder with exception of soft bit detection routines added for M-QAM soft demodulation.

The 16 and 256-QAM soft bit demodulation BER results are presented in Figure 4-32 and Figure 4-34 respectively.

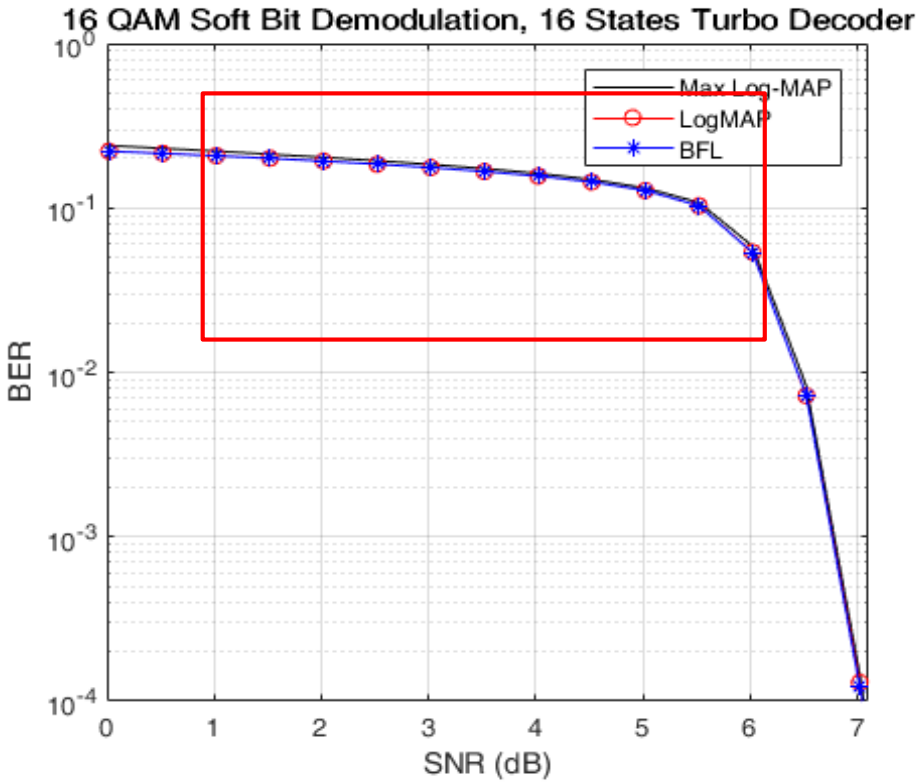


Figure 4-32 16-QAM soft bit demodulation BER for CCSDS Turbo decoder

The BER performance using the CCSDS (16 states) Turbo decoder with 16 QAM modulation is shown in Figure 4-32. Although the performance difference between the Max-Log-MAP, ideal Log-MAP, and BFL continues to be similar to one another, the performance difference at low SNRs ($\mathrm{SNR} \leq 3$) is more apparent. The highlighted area in Figure 4-32 is enlarged and presented in Figure 4-33 as follows:

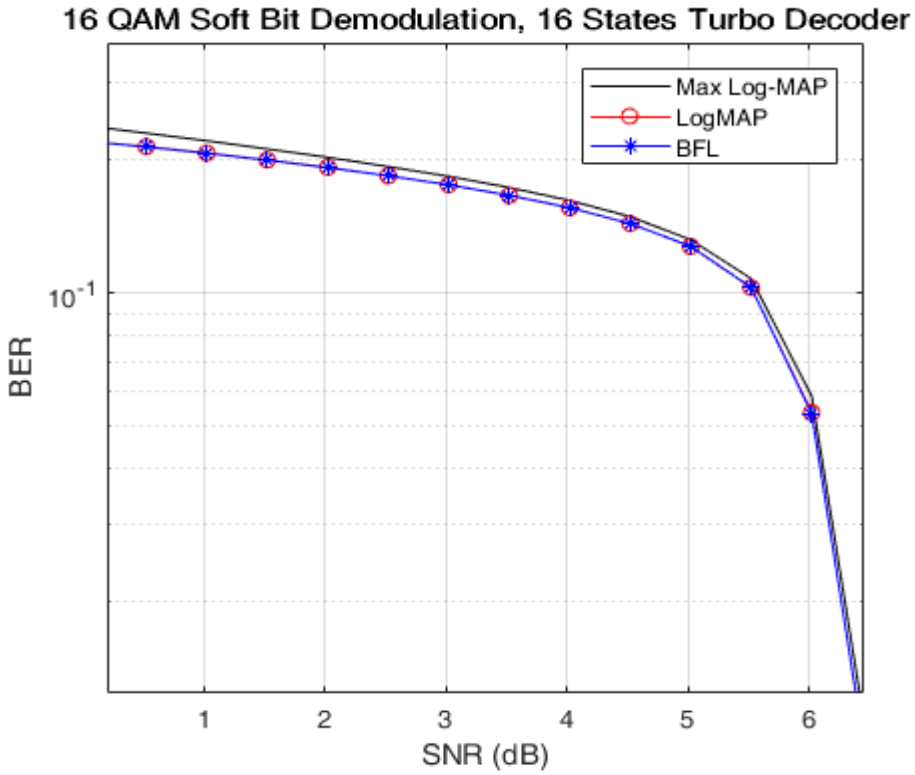


Figure 4-33 Enlarged highlighted area of 16-QAM soft bit demodulation BER for CCSDS Turbo decoder

When enlarged, the performance difference for the CCSDS Turbo decoder using 16 QAM is more apparent and the differences encompass a wider SNR region compared to the 8 states Turbo decoder. Note that the BFL continues to provide a close performance to the ideal Log-MAP. An additional experiment was conducted for the CCSDS Turbo decoder using 256 QAM as illustrated in Figure 4-34.

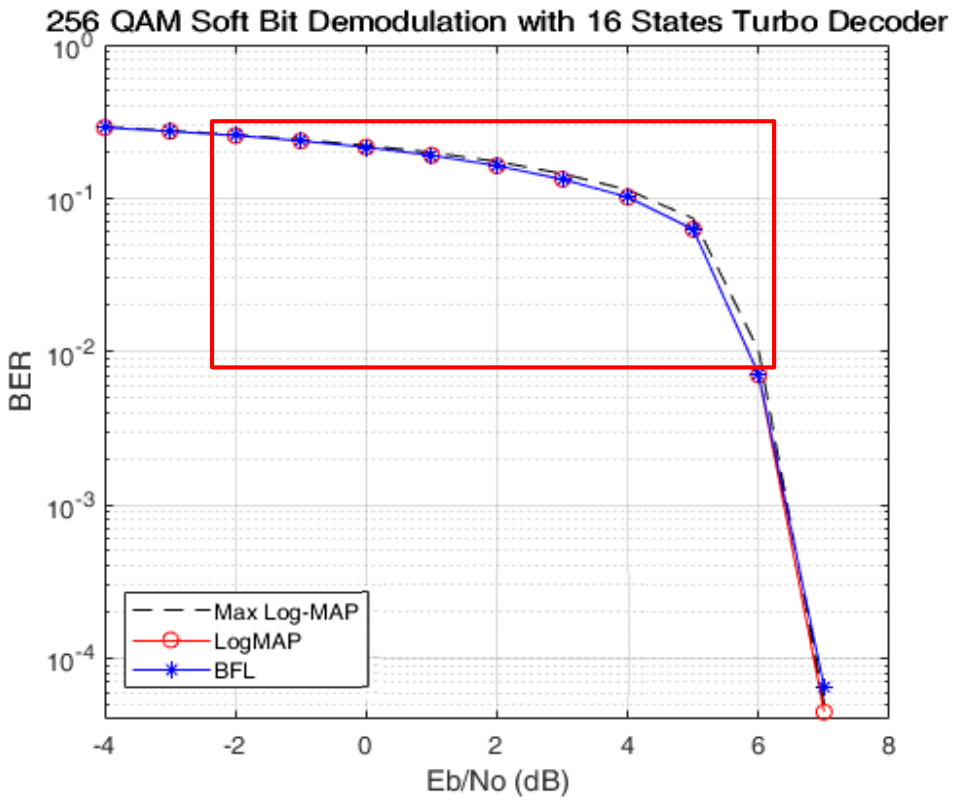


Figure 4-34 256-QAM soft bit demodulation BER for CCSDS Turbo decoder

In Figure 4-34, it is observed that the BER performance improvement with the BFL approximation is observed in the highlighted region. The enlarged illustration of the region is provided in Figure 4-35. The performance between the Max Log-MAP, exact Log-MAP remains close but more distinctive in the CCSDS Turbo decoding standard with the BFL consistently giving a closer and improved performance compared to the Max Log-MAP approximation.

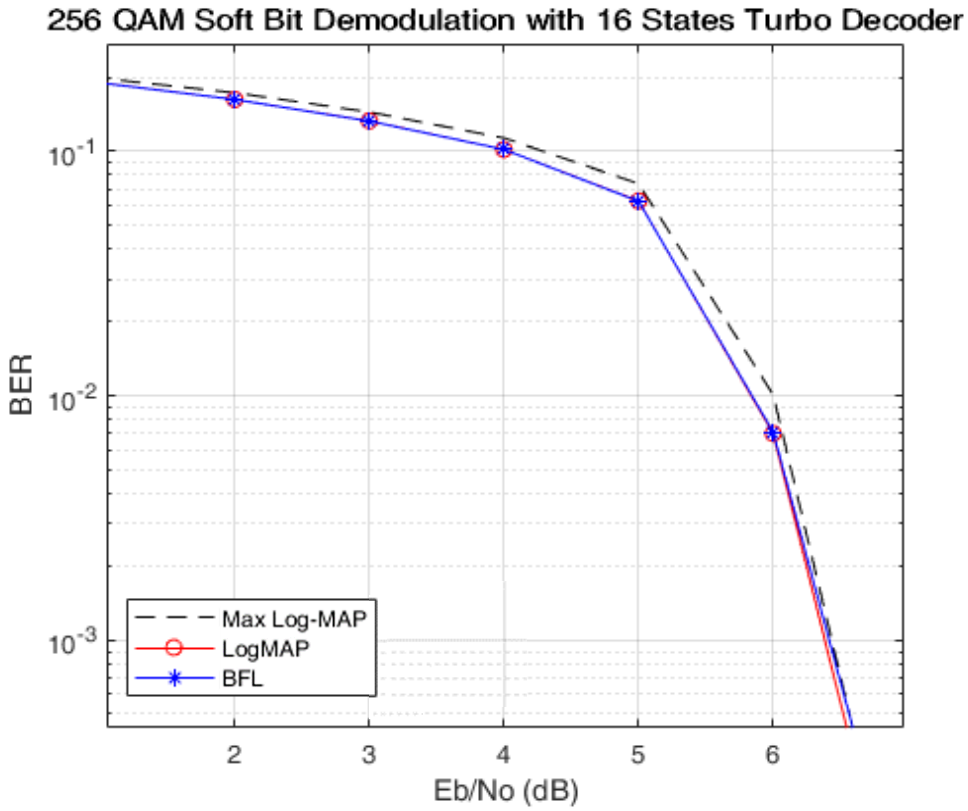


Figure 4-35 Enlarged highlighted area of 256-QAM soft bit demodulation BER for CCSDS Turbo decoder

It is also shown that soft bit demodulation using the CCSDS Turbo decoder is indeed more sensitive for decoders with longer constraint length with the widened BER performance gap between the exact Log-MAP and the Max Log-MAP. It is shown in this section that the BFL having the best GOF statistics against the exact correction function consistently achieving the closest performance to the exact Log-MAP on all fronts of BER testing either for decoding or soft bit demodulation. The BFL recorded performance difference within 4.3×10^{-4} and 6×10^{-4} dB respectfully to the 16 and 256-QAM BER performance. To summarise, based on the results in Section 4.6.1 and Section 4.6.2, the BER performance improvement introduced by the use of the BFL approximation in soft bit demodulation is more apparent at lower order QAM for both types of decoder. The improvement introduced is centered mostly at the lower regions of SNR.

5 CONCLUSION AND FURTHER WORK

The final Chapter of this thesis presents a summary of achievements based on research objectives and recommendations for future work. The results from each approximation are summarised, ranked, referenced, and concluded for suboptimal Turbo decoder implementation on FPGAs.

5.1 Conclusion and Summary of Results

The summary of achievements based on research objectives are provided in Table 5-1 as follows:

Table 5-1 Summary of results based on research objectives

Research Objective	Methodology	Results	Reference
1. Develop a Turbo decoding communication system model to investigate, analyse, and identify key factors affecting suboptimal Log-MAP Turbo decoder performance to develop an approach for optimised design.	Analysis of argument $\|x\|$ distribution in forward, backward, and LLR metrics within the correction function with varying SNR.	With low SNR, the distribution of $\|x\|$ falls more likely to lower values where the accuracy of the correction function is more effective and crucial to the overall accuracy of the decoder. As a result, one of the design criteria for new approximation development is to have a close fit to the exact correction function particularly in lower regions of $\|x\|$ for good BER performance. The BER performance and accuracy of approximation goes hand in hand with improved BER performance for approximations with good GOF statistics.	Figure 3-3, Figure 3-4, Figure 3-5, Figure 3-6
	Correction function approximation accuracy analysis with GOF statistics and mean error distribution	Calculation for the mean error distribution for all suboptimal Log-MAP correction function with goodness of fit measurements are conducted as indicator to BER performance and a set of design criteria was proposed.	Section 3.3.3 to Section 3.4.1
2. Design optimised approximations to the correction function based on the design criteria developed.	Design of the HLM approximation optimised for complexity, and strategic accuracy for lower regions of $\|x\|$	Development of mathematical approximation of the HLM with mean error distribution analysis and GOF statistics. The HLM follows after the BFL as the second best approximation with R-squared of 0.9732.	Section 3.4.2
	Design of the BFL approximation optimised for complexity, and overall accuracy via exhaustive, best fit search method	Development of mathematical approximation of the BFL with mean error distribution analysis and GOF statistics with R squared measure of 0.9932. The BFL recorded the best GOF statistics among other linear approximations in the literatures it was compared with.	Section 3.4.3
	Development of BFL 4, 6, 8 radix point approximation for fixed point processing with optimised complexity implementation on FPGA	Proposed BFLN approximations for fixed point processing using only bit shifting as replacement to the multiplier, plus the use of comparator, and adders making the BFL a purely combinational circuit.	Section 3.4.3.1

3. Evaluate the implementation of the proposed suboptimal Turbo decoder through BER performance simulation with relevant, current solutions in the literatures. Simulations under SNR mismatch conditions are also conducted to evaluate the performance of the Turbo decoder.	System simulation under perfect channel knowledge conditions under the 4G-LTE Advanced system and CCSDS Turbo decoder	Results shows very close BER performance between the BFL, HLM and the exact Log-MAP with the BFL displaying the closest performance to the exact Log-MAP. The BFL and HLM recorded BER performance within 0.015 and 0.017 dB respectively under the 4G-LTE Advance system and 3×10^{-4} and 5×10^{-4} dB respectfully for the CCSDS system.	Figure 4-2, Figure 4-4, Section 4.2
	System simulation under inaccurate channel coefficient and SNR estimation conditions under the 4G-LTE Advanced system and CCSDS Turbo decoder	Results continues to show very close BER performance between the BFL, BFLN, HLM and the exact Log-MAP.	Figure 4-7, Figure 4-8, Figure 4-11, Figure 4-12, Figure 4-16, Figure 4-17, Figure 4-21, Figure 4-22, Figure 4-26, Figure 4-27, Section 4.3
4. Design the FPGA hardware implementations of the proposed approximations with a focus on low complexity, low latency, and low power consumption.	Development of the HLM hardware implementation on FPGA	Respective complexity, timing, and power analysis was presented for the HLM. The HLM is capable of running at the maximum FPGA clock speed of 891 MHz, with total power utilisation of 0.015 W for the HLM.	Section 4.4.1
	Development of the BFL hardware implementation on FPGA	Respective complexity, timing, and power analysis was presented for the BFL. The BFL is capable of running at the maximum FPGA clock speed of 891 MHz with potential to run at faster speed due to its pure combinational nature. The total power utilisation for the BFL is the lowest compared to competing algorithms at 0.001W.	Section 4.4.2
5. Evaluate the proposed Jacobian logarithm approximations as applied to M-QAM soft bit demodulation through simulation.	Development of M-QAM soft bit demodulation simulation environment for both 4G-LTe Advanced and CCSDS standards	The BFL shows a slight improvement over the Max Log-MAP approach in the 4G-LTE Advanced system which is more distinguishable in the CCSDS standard. The BFL recorded BER performance within 1.6×10^{-3} (256-QAM) and 2.7×10^{-3} dB (1024-QAM) respectively under the 4G-LTE Advance system and 4.3×10^{4} (16-QAM) and 6×10^{-4} dB (256-QAM) respectfully for the CCSDS system.	Section 4.6

Table 5-1 briefly summarises the research achievements based on each research objectives. The next section gives an overall assessment of linear suboptimal Log-MAP solution reviewed in this thesis pertaining to the research aim of the thesis.

5.1.1 Overall Assessment, of Jacobian Logarithm Approximation for FPGA implementation in Turbo Decoding and Soft Bit Demodulation

This research has resulted in two optimised suboptimal Log-MAP solution developed from a reliable design approach for good performance in normal channel conditions and SNR mismatch. A link of good GOF statistics for suboptimal Log-MAP approximations and BER performance was also made and hardware implementation of the proposed algorithms on FPGA was developed based on complexity, speed, and power optimisation as design constraints. In order to summarise the body of work in this research, Table 5-2 is used to rank and summarise all parameters and results associated with each suboptimal Log-MAP. Further, the relevant Section and page number to which the ranking is based on is also cited for ease of reference.

Each feature of the correction function approximation is accompanied with a weight to indicate the importance of the particular feature. The approach to assigning each weight is based on the general needs and aim of the Jacobian logarithm approximation implementation on FPGA for wireless communication Turbo decoding and soft demodulation application. The three common theme within the scope of future wireless communication technology are reliability, low power, and high throughput [3], [4] as outlined in Chapter 1; which translates to BER performance, power consumption and maximum speed supported by each approximation. The weight among these should be equally

important. The proposed weight for each of these criteria is 1/3 to make a total of 1.

Within the criteria of BER performance, experiments of application of the Jacobian logarithm approximation under true SNR, SNR mismatch, and soft bit demodulation was conducted. It has been also consistently shown that the BFL, with the highest accuracy fitted to the correction function shows the closest adherence to the exact Log-MAP performance that provides for the best overall reliability. With this, under the criteria of BER performance, accuracy based on GOF statistics should be a main criterion. The proposed weight for this is 2/3 out of the 1/3 weight. Aside from the direct link between good GOF statistics with BER performance, an aspect of particular interest arises with the assessment of suboptimal Log-MAP approximation performance in channel coefficient inaccuracies for 4G-LTE Advance scenarios. It was shown that with channel coefficient variance inaccuracies for as little as 0.4, pushes the decoder into the negative SNR offset mode of operation. Interestingly, the HLM, along with the BFL continues to provide for good overall resilience. Hence this should be another criterion for consideration in terms of BER performance without overshadowing accuracy.

On the other hand, within the confines of power consumption, the FPGA power analysis would provide for a good means of reference to rank the evaluated approximations and finally, the FPGA timing analysis would the measure to rank the criteria for speed.

The weighted percentage based on all criteria mentioned are calculated as follows:

Table 5-2 Summary of results from simulation and FPGA implementation

Feature	Figure/ Table, Section	Weight	Rank							
			CLM	LLM	MLM	LC	HLM	BFL4	BFL6	BFL8
Accuracy (GOF Statistics)	Table 3-15, Table 3-18, Section 3.4.3Table 3-18	2/9	8	7	6	5	4	3	2	1
Performance under negative SNR mismatch	Figure 4-7, Figure 4-8, Figure 4-11, Figure 4-12, Section 4.3	1/9	8	5	7	6	1	4	3	2
FPGA power analysis	Table 4-16, Section 4.5.1.3	1/3	6	7	5	4	8	1	2	3
FPGA timing analysis	Table 4-15, Section 4.5.1.2	1/3	1	5	7	8	6	2	3	4
Weighted metric:			5.00	6.11	6.11	5.78	5.67	2.11	2.44	2.78
Overall ranking:			4	7	7	6	5	1	2	3

The lowest weighted metric that indicates higher optimisation value belongs to the BFLN family of fixed point processing implementation. The HLM is ranked fifth after the CLM approximation due to CLM's strong strength in hardware complexity and speed. However, it should be noted that in the case for BER performance, the HLM offers better performance than the CLM.

Limitations of research work

- Sample size for distribution of $|x|$ is limited to 56,000 data points for 1000 bits due to huge size of overall data log file. Higher sample size would give a more accurate representation of the distribution but it is expected that the difference would not be too large. Since the aim of the exercise is to determine the pattern of distribution of $|x|$ with varying SNR, the use of 56,000 data points is sufficient for this research.
- Evaluation of proposed algorithm are confined to ML exhaustive search method for soft bit demodulation. The performance gap between the Max Log-MAP approach and the exact Log-MAP approach is very close

which justifies for the approach of restrictive search methods as reviewed in the literatures. With a restrictive search approach, the performance gap between the Max Log-MAP and the exact Log-MAP is widened with trade-off to implementation complexity. This provides for a more effective application scenario for the BFL approximation to close the gap between the Max Log-MAP and the exact Log-MAP BER performance. Although with the use of BFL for soft bit demodulation presented in Section 4.6 is not very apparent, the BFL has indeed shown an improved performance compared to the Max Log-MAP in lower SNRs.

5.2 Future Work

Suggestions for further work are as follows:

1. Evaluation of a thirty-two state Turbo decoder and its response to various suboptimal solutions. Further verification of the correlation of performance in higher constraint length decoders with a good GOF statistics can be done. At the moment, there are no standards for implementing a 32 state Turbo decoder in wireless communication. This body of work should encompass an evaluation of the feasibility of a 32 state Turbo decoder on ASIC/FPGA hardware, proposing sets of new parameters of the generator polynomial for iterative Turbo decoding, interleaving, and code rate puncturing methods.

2. Investigation to the effects of various suboptimal solution to the quantum Turbo decoder. The quantum Turbo decoder is a relatively new area being developed for quantum communication with different dynamics to the system.

3. Integration of the Turbo decoder with Turbo equalisation to extend the area of application and to conduct a study on the dynamics of various suboptimal solution to the performance of the system

REFERENCES

[1] E. Calvanese Strinati *et al.*, "5GCHAMPION - Disruptive 5G Technologies for Roll-Out in 2018," *Electron. Telecommun. Res. Inst.*, vol. 40, no. 1, pp. 10–25, Feb. 2018.

[2] S. H. Won *et al.*, "Development of 5G CHAMPION testbeds for 5G services at the 2018 Winter Olympic Games," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2017, pp. 1–5.

[3] Y. Hu, M. C. Gursoy, and A. Schmeink, "Relaying-Enabled Ultra-Reliable Low-Latency Communications in 5G," *IEEE Netw.*, vol. 32, no. 2, pp. 62–68, Mar. 2018.

[4] "IMT Vision – Framework and Overall Objectives of the Future Development of IMT for 2020 and Beyond," *Recommendation ITU-R M.2083-0*, 2015.

[5] S. Mumtaz, A. Bo, A. Al-Dulaimi, and K.-F. F. Tsang, "Guest Editorial 5G and Beyond Mobile Technologies and Applications for Industrial IoT (IIoT)," *IEEE Trans. Ind. Informatics*, vol. 14, no. 6, pp. 2588–2591, Jun. 2018.

[6] R. G. Maunder, "A Vision for 5G Channel Coding," *Accelercomm White Paper*, 2016.

[7] M. F. Brejza, L. Li, R. G. Maunder, B. M. Al-Hashimi, C. Berrou, and L. Hanzo, "20 Years of Turbo Coding and Energy-Aware Design Guidelines for Energy-Constrained Wireless Applications," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 8–28, 2016.

[8] A. Sharma and M. Salim, "Polar Code: The Channel Code contender for 5G scenarios," in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, 2017, pp. 676–682.

[9] T. Richardson and S. Kudekar, "Design of Low-Density Parity Check Codes for 5G New Radio," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 28–34, Mar-2018.

[10] R. G. Maunder, "The 5G Channel Code Contenders," *Accelercomm White Paper*, 2016.

[11] "Universal Mobile Telecommunications System (UMTS); Multiplexing and channel coding (TDD)," *3GPP TS 25.222-V9.3.0- Release 9*, 2012.

[12] “LTE; Evolved Universal Terrestrial Radio Access (E-UTRA): Multiplexing and channel coding,” *3GPP TS 36.212-V14.2.0-Release 14*, 2017.

[13] I. Bhurtah, P. C. Catherine, and K. M. S. Soyjaudah, “Enhancing the error-correcting performance of LDPC codes for LTE and WiFi,” in *International Conference on Computing, Communication & Automation*, 2015, pp. 1406–1410.

[14] N. Wehn, S. Scholl, P. Schläfer, T. Lehnigk-Emden, and M. Alles, “Challenges and Limitations for Very High Throughput Decoder Architectures for Soft-Decoding,” in *Advanced Hardware Design for Error Correcting Codes*, 1st ed., Springer International Publishing, 2015.

[15] Nokia, “R1-162897 Performance and complexity of turbo, LDPC and polar codes,” *3GPP TSG-RAN WG1 #84bis*, 2016.

[16] “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” *IEEE Std 802.11*, 2012.

[17] “IEEE Standard for Air Interface for Broadband Wireless Access Systems,” *IEEE Std 802.16*, 2012.

[18] E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[19] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “VLSI Implementation of Fully Parallel LTE Turbo Decoders,” *IEEE Access*, vol. 4, pp. 323–346, 2016.

[20] T. Mohsenin, D. N. Truong, and B. M. Baas, “A low-complexity message-passing algorithm for reduced routing congestion in LDPC decoders,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 57, no. 5, pp. 1048–1061, 2010.

[21] P. Giard, C. Thibeault, and W. J. Gross, “Unrolled Hardware Architectures for Polar Decoders,” in *High-Speed Decoders for Polar Codes*, Cham: Springer International Publishing, 2017, pp. 55–71.

[22] H. K. Boyapati, R. V. Rajakumar, and S. Chakrabarti, “Quantifying the improvement in energy savings for LTE eNodeB baseband subsystem with technology scaling and multi-core architectures,” in *2012 National Conference on Communications (NCC)*, 2012, pp. 1–5.

[23] Z. Na, Z. Pan, M. Xiong, J. Xia, and W. Lu, “Soft Decision Control Iterative Channel Estimation for the Internet of Things in 5G Networks,” *IEEE Internet Things J.*, vol. PP, no. c, pp. 1–1, 2018.

[24] M. El Adawy, A. Kamaleldin, H. Mostafa, and S. Said, “Performance evaluation of turbo encoder implementation on a heterogeneous FPGA-CPU platform using SDSoC,” in *2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT)*, 2017, vol. 2018–Febru, pp. 286–290.

[25] V. Vladyslav, K. Volodymyr, Z. Sergei, and U. Anna, “Adaptive turbo codes for safety in wireless Internet of Things,” in *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2018, pp. 188–193.

[26] D. K. Halim, T. C. Ming, N. M. Song, and D. Hartono, “Software-based turbo decoder implementation on low power multi-processor system-on-chip for Internet of Things,” in *2017 4th International Conference on New Media Studies (CONMEDIA)*, 2017, vol. 30, no. 3, pp. 137–141.

[27] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate (Corresp.),” *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[28] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proceedings of ICC ’93 - IEEE International Conference on Communications*, 1993, vol. 2, pp. 1064–1070.

[29] P. Robertson, E. Villebrun, and P. Hoeher, “A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain,” in *Proceedings IEEE International Conference on Communications ICC ’95*, 1995, vol. 2, pp. 1009–1013.

[30] Hao Wang, Hongwen Yang, and Dacheng Yang, “Improved log-MAP decoding algorithm for turbo-like codes,” *IEEE Commun. Lett.*, vol. 10, no. 3, pp. 186–188, Mar. 2006.

[31] Z. Liu, B. Wu, and T.-C. Ye, “Improved Turbo Decoding With Multivariable Taylor Series Expansion,” *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 37–40, Jan. 2018.

[32] J. Erfanian, S. Pasupathy, and G. Gulak, “Reduced complexity symbol detectors with parallel structure for ISI channels,” *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1661–1671, Feb. 1994.

[33] S. Kim and M. Zhang, “Universal soft demodulation schemes for M-ary phase shift keying and quadrature amplitude modulation,” *IET Commun.*, vol. 10, no.

3, pp. 316–326, Feb. 2016.

[34] S.-J. Park, "Bitwise Log-Likelihood Ratios for Quadrature Amplitude Modulations," *IEEE Commun. Lett.*, vol. 19, no. 6, pp. 921–924, Jun. 2015.

[35] J. Mao, M. A. Abdullahi, P. Xiao, and A. Cao, "A low complexity 256QAM soft demapper for 5G mobile system," in *2016 European Conference on Networks and Communications (EuCNC)*, 2016, pp. 16–21.

[36] B. Sun and I. Guvenc, "Performance of higher order modulations under different impairments," in *2015 IEEE 16th Annual Wireless and Microwave Technology Conference (WAMICON)*, 2015, pp. 1–5.

[37] E. G. Larsson and J. Jaldén, "Fixed-Complexity Soft MIMO Detection via Partial Marginalization," vol. 8, no. 56, pp. 3397–3407, 2009.

[38] Meixiang Zhang, Sooyoung Kim, and Youngmin Kim, "Universal soft decision demodulator for M-ary adaptive modulation systems," in *2012 18th Asia-Pacific Conference on Communications (APCC)*, 2012, pp. 574–578.

[39] D. R. N. Yoge and N. Chandrachoodan, "GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications," in *2012 25th International Conference on VLSI Design*, 2012, pp. 149–154.

[40] Xin Wei, Xin Qi, Limin Xiao, Zhiyuan Shi, and Lianfen Huang, "Software-defined Radio based On Cortex-A9," in *7th International Conference on Communications and Networking in China*, 2012, vol. 1, pp. 758–761.

[41] "Toward 5G Xilinx Solutions and Enablers for Next-Generation Wireless Systems 5G Vision," *Xilinx WP476 White Paper*, 2016.

[42] M. Zivkovic, J. Otterbach, M. Tavares, D. Harman, and D. Samardzija, "An SDR-based turbo-SIC implementation: Towards a 5G new radio advanced receiver for uplink boosting," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–7.

[43] Xilinx Inc., "Delivering a Generation Ahead at 20nm and 16nm," 2018. [Online]. Available: https://www.xilinx.com/about/generation-ahead-16nm.html.

[44] "TM Synchronization and Channel Coding-Summary of Concept and Rationale," *CCSDS 130.1-G-2*, 2012.

[45] Y. Y. Ivanov, A. N. Romanyuk, A. I. Kulyk, and O. V. Stukach, "A novel suboptimal piecewise-linear-log-MAP algorithm for turbo decoding," in *2015 International Siberian Conference on Control and Communications (SIBCON)*, 2015, pp. 1–8.

[46] Lin Zhang and Shun-Zheng Yu, “A simplified log-MAP turbo decoder by fitting method,” in *The 7th International Conference on Advanced Communication Technology, 2005, ICACT 2005.*, 2005, vol. 2, no. m, pp. 854–857.

[47] H. Samadian and A. M. Nia, “Linear-constant log-map, a fast accurate algorithm for map decoding,” *J. Franklin Inst.*, vol. 349, no. 5, pp. 1721–1733, Jun. 2012.

[48] S. Talakoub and B. Shahrrava, “A linear Log-MaP algorithm for turbo decoding over AWGN channels,” in *2004 IEEE Electro/Information Technology Conference*, 2004, pp. 293–296.

[49] E. B. David Declercq, Marc Fossorier, *Academic Press Library in Mobile and Wireless Communications*, 1st ed. Elsevier, 2014.

[50] L. Hanzo, T. H. Liew, B. L. Yeap, R. Y. S. Tee, and S. X. Ng, *Turbo Coding, Turbo Equalisation and Space-Time Coding*, 2nd ed. Chichester, UK: John Wiley & Sons, Ltd, 2011.

[51] K. Deergha Rao, *Channel Coding Techniques for Wireless Communications*. New Delhi: Springer India, 2015.

[52] E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press Elsevier, 2014.

[53] M. C. Valenti and J. Sun, “The UMTS Turbo Code and an Efficient Decoder Implementation Suitable for Software-Defined Radios,” *Int. J. Wirel. Inf. Networks*, vol. 8, no. 4, pp. 203–215, 2001.

[54] “Recommendation for Space Data System Standards; TM Synchronization and Channel Coding,” *CCSDS 131.0-B-2*, 2017.

[55] C. Condo and G. Masera, “Unified turbo/LDPC code decoder architecture for deep-space communications,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 50, no. 4, pp. 3115–3125, Oct. 2014.

[56] K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones, and F. Pollara, “The Development of Turbo and LDPC Codes for Deep-Space Applications,” *Proc. IEEE*, vol. 95, no. 11, pp. 2142–2156, Nov. 2007.

[57] “Product Specification: PCE04C CCSDS Turbo Encoder,” *Small World Communications*, 2009.

[58] C. Berrou and A. Glavieux, “Near optimum error correcting coding and decoding: turbo-codes,” *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, 1996.

[59] B. Das, M. P. Sarma, K. K. Sarma, and N. Mastorakis, “Block interleaver design for high data rate wireless networks,” in *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, 2015, pp. 1000–1005.

[60] J. Sharma, S. Mishra, and N. Sharma, “An analysis of interleaver types for better BER performance in linear turbo equalizer,” in *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 2017, no. 2, pp. 1–5.

[61] V. Sawant and A. Bhise, “Performance analysis of encrypted structured random interleaver,” in *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2017, pp. 699–702.

[62] S. Dolinar and D. Divsalar, “Weight distributions for turbo codes using random and nonrandom permutations,” *TDA Prog. Rep.*, vol. 42, no. 122, pp. 56–65, 1995.

[63] W. J. Gross and P. G. Gulak, “Simplified MAP algorithm suitable for implementation of turbo decoders,” *Electron. Lett.*, vol. 34, no. 16, p. 1577, 1998.

[64] M. A. Foda, M. A. A. El Ghany, and K. Hoffman, “Efficient polynomial regression algorithm for LTE turbo decoding,” in *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2017, pp. 246–269.

[65] D.-H. Nguyen and H. Nguyen, “An improved Log-MAP algorithm based on polynomial regression function for LTE Turbo decoding,” in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 2163–2167.

[66] B. Classon, K. Blankenship, and V. Desai, “Turbo Decoding With The Constant-Log-Map Algorithm,” in *2nd International Symposium on Turbo Codes & Related Topics*, 2000, pp. 467–470.

[67] Jung-Fu Cheng and T. Ottosson, “Linearly approximated log-MAP algorithms for turbo decoding,” in *VTC2000-Spring. 2000 IEEE 51st Vehicular Technology Conference Proceedings (Cat. No.00CH37026)*, 2000, vol. 3, pp. 2252–2256.

[68] J. Vogt and A. Finger, “Improving the max-log-MAP turbo decoder,” *Electron. Lett.*, vol. 36, no. 23, pp. 1937–1939, 2000.

[69] M. Xu, F. Lu, J. Wang, L. Cheng, D. Guidotti, and G.-K. Chang, “Key Technologies for Next-Generation Digital RoF Mobile Fronthaul With Statistical Data Compression and Multiband Modulation,” *J. Light. Technol.*, vol. 35, no. 17, pp.

3671–3679, Sep. 2017.

[70] T. Ota, M. Nakamura, and H. Otsuka, “Performance evaluation of OFDM-based 256- and 1024-QAM in multipath fading propagation conditions,” in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 554–556.

[71] R. Tian, T. Ota, and H. Otsuka, “Influence of phase error on OFDM-based 4096-QAM with turbo coding,” in *2018 International Conference on Information Networking (ICOIN)*, 2018, vol. 2018–Janua, pp. 352–355.

[72] Y. Wei and L. Changxing, “Leading Innovation in Microwave Broadband Transmission for Pre5G/5G,” *ZTE Technologies*, 2017. [Online]. Available: http://wwwen.zte.com.cn/endata/magazine/ztetechnologies/2017/no6/articles/201711/t20171116_466189.html.

[73] F. Bellili, A. Methenni, S. Ben Amor, S. Affes, and A. Stephenne, “Time Synchronization of Turbo-Coded Square-QAM-Modulated Transmissions: Code-Aided ML Estimator and Closed-Form Cramér–Rao Lower Bounds,” *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10776–10792, Dec. 2017.

[74] H. Chung, S. Hong, B. K. Lim, L. Ma, and B. K. Yi, “Performance analysis and multidimensional gray coding scheme for a real-number M-ary QAM,” in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, 2016, pp. 954–959.

[75] B. P. Lathi and Z. Ding, *Modern Digital and Analog Communication*, 5th ed. Oxford University Press, 2018.

[76] S. Haykin, *Digital Communication Systems*, 1st ed. Wiley, 2013.

[77] J. G. Proakis, M. Salehi, and G. Bauch, *Contemporary Communication Systems Using MATLAB*, 3rd ed. Cengage Learning, 2012.

[78] E. Matricciani, “A Model of the Probability Distribution of the Signal-to-Noise Ratio Estimated from BER Measurements,” in *2011 IEEE Vehicular Technology Conference (VTC Fall)*, 2011, no. 1, pp. 1–5.

[79] Neha, “Implementation and Performance Analysis of Convolution Error Correcting Codes with Code Rate=1/2,” in *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, 2016, pp. 482–486.

[80] R. A. Baby, “Convolution coding and applications: A performance analysis under AWGN channel,” in *2015 International Conference on Communication*

Networks (ICCN), 2015, pp. 84–88.

[81] H. Yuan, M. Wu, and P. Kam, "A modification of the BCJR algorithm over the flat fading channel and its application to turbo decoding," in *2017 IEEE/CIC International Conference on Communications in China (ICCC)*, 2017, pp. 1–6.

[82] Pingping Shang, Sooyoung Kim, and Kwonhue Choi, "Soft MMSE receiver for turbo coded MIMO system," in *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2011, pp. 471–475.

[83] M. Rizk, A. Baghdadi, M. Jezequel, Y. Mohanna, and Y. Atat, "Flexible and efficient architecture design for MIMO MMSE-IC linear turbo-equalization," in *2013 Third International Conference on Communications and Information Technology (ICCIT)*, 2013, pp. 340–344.

[84] F. M. L. Tavares, G. Berardinelli, N. H. Mahmood, T. B. Sorensen, and P. Mogensen, "On the Impact of Receiver Imperfections on the MMSE-IRC Receiver Performance in 5G Networks," in *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)*, 2014, pp. 1–6.

[85] S. Ahmadi, *LTE-Advanced: A Practical Systems Approach to Understanding 3GPP LTE Releases 10 and 11 Radio Access Technologies*, 1st ed. Academic Press, 2014.

[86] M. El-Khamy, Jinhong Wu, Jungwon Lee, H. Roh, and Inyup Kang, "Near-optimal turbo decoding in presence of SNR estimation error," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, no. 2, pp. 3737–3742.

[87] J. Wu, I. Kang, J. Lee, and M. El-Khamy, "Online log-likelihood ratio scaling for robust turbo decoding," *IET Commun.*, vol. 8, no. 2, pp. 217–226, Jan. 2014.

[88] Hyundong Shin and Jae Hong Lee, "Channel reliability estimation for turbo decoding in rayleigh fading channels with imperfect channel estimates," *IEEE Commun. Lett.*, vol. 6, no. 11, pp. 503–505, Nov. 2002.

[89] Y. Zhou and B. Shahrrava, "Effect of SNR mismatch on the performance of turbo decoding algorithms," in *2009 IEEE International Conference on Electro/Information Technology*, 2009, pp. 15–18.

[90] K. Xu, B. Shahrrava, and J. Wan, "A unified framework for SNR mismatch in turbo algorithms," in *2010 International Conference on Wireless Communications & Signal Processing (WCSP)*, 2010, vol. 2, no. 2, pp. 1–4.

[91] M. Ali Khalighi, "Sensitivity to SNR mismatch in interference cancelling based

turbo equalizers," in *The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2002, vol. 2, pp. 379–383.

[92] N. Mehta, "XILINX 7 Series FPGAs : The Logical Advantage," *Xilinx WP405 White Paper*, 2012.

[93] "UltraScale Architecture DSP Slice," *Xilinx UG 579*, 2015.

[94] H. Homulle and E. Charbon, "Performance characterization of Altera and Xilinx 28 nm FPGAs at cryogenic temperatures," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 25–31.

[95] "Multiplying the Value of 16nm - Staying a Generation Ahead Wireless and Waveform Processing Packet Processing and Transport Image and Video Processing High Performance Computing Connected Control," *Xilinx*, 2015.

[96] "UltraScale Architecture and Product Data Sheet : Overview," *Xilinx DS890*, 2018.

[97] "Virtex-7 T and XT FPGAs Data Sheet : DC and AC Switching Characteristics," *Xilinx DS893*, 2018.

[98] S. Kim and M. Zhang, "Efficient soft demodulation scheme for Digital Video Broadcasting via Satellite – Second Generation system," *IET Commun.*, vol. 8, no. 1, pp. 124–132, Jan. 2014.

[99] B. Zheng, X. Wang, M. Wen, and F. Chen, "Soft Demodulation Algorithms for Generalized Spatial Modulation Using Deterministic Sequential Monte Carlo," *IEEE Trans. Wirel. Commun.*, vol. 16, no. 6, pp. 3953–3967, Jun. 2017.

[100] M. Ivanov, F. Brannstrom, A. Alvarado, and E. Agrell, "On the Exact BER of Bit-Wise Demodulators for One-Dimensional Constellations," *IEEE Trans. Commun.*, vol. 61, no. 4, pp. 1450–1459, Apr. 2013.

[101] Q. Sun and W. Qi, "Soft-demodulation algorithm for 64QAM and it's application in HSPA+," in *2012 IEEE 11th International Conference on Signal Processing*, 2012, no. 1, pp. 2309–2312.

[102] Q. Xie, Z. Wang, and Z. Yang, "Simplified Soft Demapper for APSK with Product Constellation Labeling," *IEEE Trans. Wirel. Commun.*, vol. 11, no. 7, pp. 2649–2657, Jul. 2012.

[103] E. Akay and E. Ayanoglu, "Low complexity decoding of bit-interleaved coded modulation for M-ary QAM," in *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, 2004, vol. 2, p. 901–905 Vol.2.

[104] R. Ghaffar and R. Knopp, “Low Complexity Metrics for BICM SISO and MIMO Systems,” in *2010 IEEE 71st Vehicular Technology Conference*, 2010, pp. 1–6.

[105] Q. Wang, Q. Xie, Z. Wang, S. Chen, and L. Hanzo, “A Universal Low-Complexity Symbol-to-Bit Soft Demapper,” *IEEE Trans. Veh. Technol.*, vol. 63, no. 1, pp. 119–130, Jan. 2014.

[106] Kwangmin Hyun, Dongweon Yoon, Ki Seol Kim, Lei Wang, and Sang Kyu, “Bitwise Decomposition of M-QAM Signals,” in *2005 5th International Conference on Information Communications & Signal Processing*, 2005, pp. 1067–1070.

[107] M. L. Rizzo, *Statistical Computing with R*, 2nd ed. Chapman & Hall/CRC, 2015.

[108] O. Y. Bursalioglu, G. Caire, and D. Divsalar, “Joint source-channel coding for deep space image transmission using rateless codes,” in *2011 Information Theory and Applications Workshop*, 2011, pp. 1–10.

[109] Mathworks, “Evaluating Goodness of Fit.” [Online]. Available: http://www.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html#bq_5kwr-3.

[110] F. de Dinechin, “Multiplication by Rational Constants,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 59, no. 2, pp. 98–102, Feb. 2012.

[111] H. F. Ugurdag, A. Bayram, V. E. Levent, and S. Goren, “Efficient Combinational Circuits for Division by Small Integer Constants,” in *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, 2016, pp. 1–7.

[112] A. K. Oudjida, M. L. Berrandjia, and N. Chaillet, “A new low-power recoding algorithm for multiplierless single/multiple constant multiplication,” in *2013 IEEE Faible Tension Faible Consommation*, 2013, no. 1, pp. 1–4.

[113] H. F. Ugurdag, F. de Dinechin, Y. S. Gener, S. Goren, and L.-S. Didier, “Hardware Division by Small Integer Constants,” *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2097–2110, Dec. 2017.

[114] R. Nane *et al.*, “A Survey and Evaluation of FPGA High-Level Synthesis Tools,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[115] Guohui Wang, Hao Shen, Yang Sun, J. R. Cavallaro, A. Vosoughi, and Yuanbin Guo, “Parallel Interleaver Design for a High throughput HSPA+/LTE Multi-Standard Turbo Decoder,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 61, no. 5, pp. 1376–1389, May 2014.

[116] S. Belfanti, C. Roth, M. Gautschi, C. Benkeser, and Qiuting Huang, "A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS," in *2013 Symposium on VLSI Circuits*, 2013, pp. C284–C285.

[117] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15GBit/s turbo code decoder for LTE advanced base station applications," in *2012 7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 2012, pp. 21–25.

[118] "Virtex UltraScale+ FPGA Data Sheet : DC and AC Switching Characteristics," *Xilinx DS923*, 2016.

[119] "Vivado Design Suite User Guide: Model-Based DSP Design Using System Generator," *Xilinx UG 897*, 2015.

[120] A. Finnerty and H. Ratigner, "Reduce Power and Cost by Converting from Floating Point to Fixed Point," *Xilinx WP491 White Paper*, 2017.

[121] K. Andrews, V. Stanton, S. Dolinar, V. Chen, J. Berner, and F. Pollara, "Turbo-decoder implementation for the deep space network," 2002.

[122] L. Hadj Abderrahmane and M. Bacha, "High gain channel coding for satellite communication system," in *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, 2014, pp. 742–746.

[123] M. A. Khalighi, "Effect of mismatched snr on the performance of log-MAP turbo detector," *IEEE Trans. Veh. Technol.*, vol. 52, no. 5, pp. 1386–1397, Sep. 2003.

[124] "Vivado Design Suite User Guide: Design Analysis and Closure Techniques," *Xilinx UG906*, 2012.

[125] M. Ivanov, *Bit-Wise Decoders for Coded Modulation and Broadcast Coded Slotted ALOHA*. 2016.

LIST OF PUBLICATIONS

- L. L. Lim, D. Wee, and G. Lim, "Hybrid Log-MAP Algorithm for Turbo Decoding Over AWGN Channel," ICWMC 2011: The Seventh International Conference on Wireless and Mobile Communications vol. 0, no. 1, pp. 211–214, 2011.

APPENDIX A

Title: Channel characterisation with constellation diagram / scatter plot and eye diagram for transmitted M-QAM symbols MATLAB codes

Brief description: The codes transmits M-QAM symbols which are Turbo encoded and transmitted over the flat fading Rayleigh with AWGN channel. Received symbols are plotted using the constellation / scatter plot. The eye diagram is also across two samples for 1000 symbols transmitted. Results are presented in Table 2-6.

Reference: Section 2.4.

```
clc
close all
rng(0,'twister')        % Seeding control
clear
```

%% Generate bits to transmit

```
M=16; % Change QAM order here
k_bits = log2(M); % Bits per symbol
numSymPerFrame = 50; % Number of symbols transmitted
total_frame=1;
frame_size=k_bits*numSymPerFrame;
normfactor=1/sqrt(2*sqrt(M/4)*sum([1:2:(sqrt(M)-1)].^2)/(M/4)); % normalizing factor for
QAM
TotalBitsSimulated=total_frame*frame_size;
DataFrame = randi([0 1], frame_size, total_frame); % generate full data frames to be
transmitted.
interleaver=randintrlv(1:frame_size,5);
enc_tail_bit_len=3;% 3 for 4G-LTE, 4 for CCSDS, equals to constraint length
enc_op_len=frame_size+enc_tail_bit_len;
```

%% Transmission

```
EbNodBVec =(25) % Eb/No set to 25 dB, change here for other values
berEst = zeros(size(EbNodBVec));
snrdB=zeros(1,length(EbNodBVec));
```

%% Setting up transmission for each SNR

```
    n = 1;
```

%% Convert Eb/No to SNR

```
    snrdB(n)= EbNodBVec(n) + 10*log10(k_bits)
```

%% Reset number of error, bit counters, and soft bit LLR values

```
    numErrs = 0;
    rxdata_llr_LogJac=zeros(1,frame_size);
```

%% Sending data frame by frame

```
            p=1;
            Sig=DataFrame(:,p);
```

%% Turbo encoder

```
            intrlvSig=intrlv(Sig,interleaver);
            % Magic Genie sign
            mgc_sig=sign(2*Sig-1);
            % Encode
            [u1_k, v1_k]=enc5G(Sig);
            [u2_k, v2_k]=enc5G(intrlvSig); %u2_k not transmitted as it is the
interleaved version of the input u1_k
```

```
            % Puncturing and forming txSig frame
            v1_k(2:2:enc_op_len)=0; %Zero-ing all even index of parity 1
            v2_k(1:2:enc_op_len)=0; %Zero-ing all odd index of parity 2
            parity_vec=v1_k + v2_k;
            txSig_vec=[u1_k';parity_vec];
            txSig=[txSig_vec(:); zeros(k_bits-rem(enc_op_len*2,k_bits),1)]; % append
zeros to make txSig divisible by 16 with 0 remainder for modulation
```

%% Modulate

```
            txSig_mod =
qammod(txSig,M,'gray','InputType','bit','UnitAveragePower',true);
```

%% Rayleigh with AWGN channel

%% AWGN channel noise

```
            L=length(txSig_mod);
            AvgTxPow=mean(abs(txSig_mod).^2); % measure txSig power, ensures received
signal is guaranteed to have specified SNR
            N0=AvgTxPow*10^(-snrdB(n)/10);
                if (isreal(txSig_mod)) % for BPSK
                    noiseVar=N0;
                    noiseSigma=sqrt(N0);
                    noise=noiseSigma*randn(L,1);
                else % for QAM with I & Q components
                    noiseVar=N0/2;
                    noiseSigma=sqrt(N0/2);
                    noise=noiseSigma*(randn(L,1)+1i*randn(L,1));
                end
```

%% Rayleigh channel

```
            channel_taps=1; % Flat fading Rayleigh
            H = 1/sqrt(2)*(randn(1,channel_taps)+1i*randn(1,channel_taps));
            rxSig1=H.*txSig_mod+noise;
```

%% Constellation diagram

```
            cd = comm.ConstellationDiagram('ShowReferenceConstellation',false);
            step(cd,rxSig1)
```

%% Eye diagram plot

```
            eyediagram(rxSig1,2)
```

APPENDIX B

Title: Argument $|x|$ distribution with varying E_b/N_0 MATLAB codes

Brief description: The $|x|$ values input to the correction function is captured for 70,000 bits simulated with 5 iterations. The captured data is cleaned with GVim and binned to give Figure 3-3, Figure 3-4, Figure 3-5, and combined in Figure 3-6.

Reference: Section 3.2

```
F_N=7; %Number of frames to tx.
 rand('state', 14);
 dataS=round(rand(1,1000)*1000);

 rand('state', 110);
 noiseS=round(rand(1,1000)*1000);

for SD=1:F_N %Run for total number of frames, 1 frame at a time
```

%% Generate frame size of 1000 bits

```
rand('state', dataS(SD));
L1=1000 %input length
d1_k=round(rand(1,L1));
L2=L1+4; % 4 tail bits
```

%% Random interleaver

```
intrlv=20;
d2_k=randintrlv(d1_k,intrlv);
```

%% Encode

```
%RSC1, no termination
[u1, v1]=Enc16(d1_k);
%RSC2, with termination
[u2, v2]=Enc16(d2_k);
```

%% SNR

```
ebnodb=0.2 %% Change Eb/No here
ebno=10^(ebnodb/10);
rate=1/2; % 1/3 if not punctured
L_c=4*ebno*rate;
sigma = 1/sqrt(2*rate*ebno);
randn('state',noiseS(SD));
```

%% Transmit

```
% Puncture
 n1=2:2:L2;
 v1(n1)=0; % Deleting all even index of bpsk_v1
 n2=1:2:L2;
 v2(n2)=0; % Deleting all odd index of bpsk_v2
 parity=v1+v2; % Combining parity bits together

% Multiplexing
 tx=[u1;parity];
 tx_sig=[tx(:)',u2(L1+1:L2)];

% Modulate, AWGN noise
 bpsk_tx= 2*tx_sig -1; % BPSK
 len_tx=length(bpsk_tx);
 rx_sig=bpsk_tx+sigma*randn(1,len_tx); %AWGN noise

% Separating signal to decode, demultiplexing
 u2rx_tail=rx_sig(len_tx-3:len_tx); %Decoder2 tail bits
 mux_sig=rx_sig(1:len_tx-4);
 L3=length(mux_sig);
 nu1=1:2:L3;
```

```
      u1rx=mux_sig(nu1); % Systematic bits
      u2rx=[randintrlv(u1rx(1:L1),intrlv),u2rx_tail]; % Interleaved Systematic bit with
tail bits
      nu2=2:2:L3;
      parrx=mux_sig(nu2); % Parity bits (punctured)
      v1rx=parrx;
      v1rx(n1)=0;
      v2rx=parrx;
      v2rx(n2)=0;
```

%% Iteration

```
Iter=5; % Number of iteration
L_e21=zeros(1,L2); % Initializing a-priori L_e to zeros for first iteration
diary('O_2dBraw.txt') % Data logging begins and saved to txt file
```

```
%     for i=1:iter
%         %% Decoder 1
%         [L_e12, LLR1] = Dec16(L_c,v1rx,u1rx,L_e21);
%         L_e12=randintrlv(L_e12(1:L1),intrlv);
%         L_e12(L1+1:L2)=0; %Insert L_e=0 for tail bits
%         %% Decoder 2
%         [L_e21, LLR2] = Dec16(L_c,v2rx,u2rx,L_e12);
%         L_e21=randdeintrlv(L_e21(1:L1),intrlv);
%         L_e21(L1+1:L2)=0; %Insert L_e=0 for tail bits
%         LLR2(L1+1:L2)=[];% Delete tail bits
%         LLR2=randdeintrlv(LLR2,intrlv);
%     end
%     diary('off');% Data logging off

end
```

APPENDIX C

Title: Binned mean SSE error distribution of $|x|$ for all approximations against the exact correction function MATLAB code

Brief description: The MATLAB codes calculates the SSE with 0.01 resolution of all approximation against the exact correction function and categorises the results to 50 bins with increasing values of $|x|$ before a mean for each bin is computed and graphed to show the error distribution between approximations and the exact correction function.

Reference: Section 3.3.1

%% Error distribution plot for all approximations in region 0<=|x|<=5.99.

```
clc
x=0:0.01:5.99;
lenx=length(x);
```

%% Exact Log-MAP with Jacobian Log

```
y=log(1+exp(-x));
```

%% CLM

```
ygg=zeros(1, lenx);
ygg(x<=1.5)=0.5;
```

%% LLM

```
yMc=max((log(2)-x/2),0);
```

%% LC

```
ySam=zeros(1,lenx);
ySam(x<=2.45)=-0.24*x(x<=2.45)+0.596;
ySam(2.45<x&x<=3.5)=0.048;
```

%% MLM

```
yWg=log(2)./(2.^(floor(x+0.5)));
```

%% FLM

```
yZy=zeros(1,lenx);
for ind=1:lenx
        if x(ind)<=4

            yZy(ind)=0.6782-0.3920*x(ind)+0.0576*(x(ind))^2;
        else

            yZy(ind)=0;
        end
end
```

%% HLM

```
yHLM=zeros(1,lenx);

for ind=1:lenx
    if x(ind)<=1.5
    yHLM(ind)=0.6512-0.3251*x(ind);
    else
    yHLM(ind)=0.1635./(2.^floor(0.5*x(ind)));
    end
end
```

%% BFL 2018

```
yBFL=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4347
    yBFL(k)=0.6547-0.3323*x(k);
    elseif x(k)>1.4347 & x(k)<3.6865
    yBFL(k)=0.2914-0.079*x(k);
    end

end
```

%% BFL4

```
yBFL4=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.375
    yBFL4(k)=0.625-0.3125*x(k);
    elseif x(k)>1.375 & x(k)<3.625
    yBFL4(k)=0.25-0.0625*x(k);
    end

end
```

%% BFL6

```
yBFL6=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4219
    yBFL6(k)=0.6406-0.3281*x(k);
    elseif x(k)>1.4219 & x(k)<3.6719
    yBFL6(k)=0.2813-0.0781*x(k);
    end

end
```

%% BFL8

```
yBFL8=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4336
    yBFL8(k)=0.6523-0.3320*x(k);
    elseif x(k)>1.4336 & x(k)<3.6836
    yBFL8(k)=0.2891-0.0781*x(k);
    end
end
```

%% Mean Error Distribution Plots

```
xxx=median(reshape(x,[12,50])); % bar graph width center for each bin
nbins=50; % number of bins
```

%% CLM

```
plotCLM=((ygg-y).^2);
histogram(plotCLM,nbins);
qa=reshape(plotCLM,[12,50]);
qya=mean(qa);
bar(xxx,qya)
ylabel('SSE Mean')
xlabel('|x| bins')
title('CLM with T=1.5 and C=1/2 Error Distribution')
grid on
```

%% LLM

```
plotMc=((yMc-y).^2);
q=reshape(plotMc,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Linear Log-MAP Error Distribution')
grid on
```

%% MLM

```
plotWg=((yWg-y).^2);
q=reshape(plotWg,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Multistep Log-MAP Error Distribution')
grid on
```

%% LC

```
plotLC=((ySam-y).^2);
q=reshape(plotLC,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Linear Constant Log-MAP Error Distribution')
grid on
```

%% FLM

```
plotZy=((yZy-y).^2);
q=reshape(plotZy,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Second order Log-MAP Error Distribution')
grid on
```

%% HLM

```
plotHLM=((yHLM-y).^2);
q=reshape(plotHLM,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Hybrid Log-MAP Error Distribution')
grid on
```

%% BFL

```
plotBFL=((yBFL-y).^2);
q=reshape(plotBFL,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Best Fit Linear Error Distribution')
grid on
```

%% BFL4

```
plotBFL4=((yBFL4-y).^2);
q=reshape(plotBFL4,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Best Fit Linear 4 Radix Error Distribution')
grid on
```

%% BFL6

```
plotBFL6=((yBFL6-y).^2);
q=reshape(plotBFL6,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
```

```
title('Best Fit Linear 6 Radix Error Distribution')
grid on
```

%% BFL8

```
plotBFL8=((yBFL8-y).^2);
q=reshape(plotBFL8,[12,50]);
qy=mean(q);
bar(xxx,qy)
ylabel('SSE Mean')
xlabel('|x| bins')
title('Best Fit Linear 8 Radix Error Distribution')
grid on
```

APPENDIX D

Title: GOF statistics MATLAB code calculation for all approximations against the exact correction function for $0 \leq |x| < 4$

Brief description: The MATLAB codes calculates the GOF statistics to give results in each GOF statistics Table in Section 3.3.2.

Reference: Section 3.3.2

```
% Plotting the actual correction term and approximations for first region
% 0<=|x|<=1.5 and 0<=|x|<=4

x=0:0.01:4;
% x=0:0.01:1.5;
lenx=length(x);
```

%% Exact Log-MAP with Jacobian Log

```
y=log(1+exp(-x));
```

%% CLM

```
yCLM=zeros(1, lenx);
yCLM(x<=1.5)=0.5;
```

%% LLM

```
yLLM=max((log(2)-x/2),0);
```

%% LC

```
yLC=zeros(1,lenx);
yLC(x<=2.45)=-0.24*x(x<=2.45)+0.596;
yLC(2.45<x&x<=3.5)=0.048;
```

%% MLM

```
yMLM=log(2)./(2.^(floor(x+0.5)));
```

%% FLM

```
yFLM=zeros(1,lenx);
for ind=1:lenx
        if x(ind)<=4

            yFLM(ind)=0.6782-0.3920*x(ind)+0.0576*(x(ind))^2;
        else

            yFLM(ind)=0;
        end
end
```

%% HLM

```
yHLM=zeros(1,lenx);

for ind=1:lenx
    if x(ind)<=1.5
    yHLM(ind)=0.6512-0.3251*x(ind);
    else
    yHLM(ind)=0.1635./(2.^floor(0.5*x(ind)));
    end
end
```

%% BFL

```
yBFL=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4347
    yBFL(k)=0.6547-0.3323*x(k);
    elseif x(k)>1.4347 & x(k)<3.6865
    yBFL(k)=0.2914-0.079*x(k);
    end

end
```

%% BFL4

```
yBFL4=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.375
    yBFL4(k)=0.625-0.3125*x(k);
    elseif x(k)>1.375 & x(k)<3.625
    yBFL4(k)=0.25-0.0625*x(k);
    end

end
```

%% BFL6

```
yBFL6=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4219
    yBFL6(k)=0.6406-0.3281*x(k);
    elseif x(k)>1.4219 & x(k)<3.6719
    yBFL6(k)=0.2813-0.0781*x(k);
    end

end
```

%% BFL8

```
yBFL8=zeros(1,lenx);
for k=1:length(x)
    if x(k)<=1.4336
    yBFL8(k)=0.6523-0.3320*x(k);
    elseif x(k)>1.4336 & x(k)<3.6836
    yBFL8(k)=0.2891-0.0781*x(k);
    end
end
```

%% GoF Tests

%% Sum of Squared Errors

```
fprintf('Sum of Squared Errors')
sseCLM=sum((yCLM-y).^2)
sseLLM=sum((yLLM-y).^2)
sseMLM=sum((yMLM-y).^2)
sseLC=sum((yLC-y).^2)
sseFLM=sum((yFLM-y).^2)
sseHLM=sum((yHLM-y).^2)
sseBFL=sum((yBFL-y).^2)
sseBFL4=sum((yBFL4-y).^2)
sseBFL6=sum((yBFL6-y).^2)
sseBFL8=sum((yBFL8-y).^2)
```

%% Mean Squared Error

```
fprintf('Mean Squared Error')
mseCLM=sseCLM/lenx
mseLLM=sseLLM/lenx
mseMLM=sseMLM/lenx
mseLC=sseLC/lenx
mseFLM=sseFLM/lenx
mseHLM=sseHLM/lenx
mseBFL=sseBFL/lenx
mseBFL4=sseBFL4/lenx
mseBFL6=sseBFL6/lenx
mseBFL8=sseBFL8/lenx
```

%% Root Mean Squared Error

```
fprintf('Root Mean Squared Error')
rmseCLM=sqrt(mseCLM)
rmseLLM=sqrt(mseLLM)
rmseMLM=sqrt(mseMLM)
rmseLC=sqrt(mseLC)
rmseFLM=sqrt(mseFLM)
rmseHLM=sqrt(mseHLM)
rmseBFL=sqrt(mseBFL)
rmseBFL4=sqrt(mseBFL4)
rmseBFL6=sqrt(mseBFL6)
rmseBFL8=sqrt(mseBFL8)
```

%% R Squared

```
fprintf('R Squared')
rsqCLM=1-sseCLM/sum((y-mean(y)).^2)
rsqLLM=1-sseLLM/sum((y-mean(y)).^2)
rsqMLM=1-sseMLM/sum((y-mean(y)).^2)
rsqLC=1-sseLC/sum((y-mean(y)).^2)
rsqFLM=1-sseFLM/sum((y-mean(y)).^2)
rsqHLM=1-sseHLM/sum((y-mean(y)).^2)
rsqBFL=1-sseBFL/sum((y-mean(y)).^2)
rsqBFL4=1-sseBFL4/sum((y-mean(y)).^2)
rsqBFL6=1-sseBFL6/sum((y-mean(y)).^2)
rsqBFL8=1-sseBFL8/sum((y-mean(y)).^2)
```

APPENDIX E

Title: MATLAB code for exhaustive search of best linear fit to the correction function.

Brief description: The MATLAB code divides the START, STOP range and performs and exhaustive search using all combinations of possible points set to find the optimal points that will yield the smallest SSE.

Reference: Section 3.4.3

```
clear
close all
clc
```

%% Setup of region of search and resolution

```
delx=0.01; % fixed at 0.01 for SSE computation
delOP=0.1;

% PointC
C_start=0.6;
C_end=0.8;
C_all=C_start:delOP:C_end;
lenC=length(C_all);
% PointB
B_start=1;
B_end=1.6;
B_all=B_start:delOP:B_end;
lenB=length(B_all);
% PointD
D_start=0.1;
D_end=0.3;
D_all=D_start:delOP:D_end;
lenD=length(D_all);
% PointF
F_start=3.5;
F_end=4;
F_all=F_start:delOP:F_end;
lenF=length(F_all);

% X values
x=0:delx:10;
lenx=length(x);
% YLogJac values
y=log(1+exp(-x));
```

%% All Combos of B,F,C,D1,D2

```
Combos=allcomb(B_all, F_all,C_all,D_all);
[ComboR,ComboC]=size(Combos); %BFCDE
SSE_T=zeros(1,ComboR);
SSE_R1=zeros(1,ComboR);
SSE_R2=zeros(1,ComboR);
SSE_R3=zeros(1,ComboR);
```

%% Sweep over all parameters and compute SSEs

```
for CombInd=1:ComboR
    BFCD=Combos(CombInd,:);
    B=BFCD(1);
    F=BFCD(2);
    C=BFCD(3);
    D=BFCD(4);

    yAp=zeros(1,lenx);
    for ind=1:lenx
        if x(ind)<=B
            m1=(D-C)/B;
            yAp(ind)=m1*x(ind)+C;
        elseif B<x(ind)&&x(ind)<=F
            m2=-D/(F-B);
            yAp(ind)=m2*(x(ind)-B)+D;
        else
            yAp(ind)=0;
        end
```

```
    end
    SSE_T(CombInd)=sum((yAp-y).^2);
    SSE_R1(CombInd)= sum((yAp(x<=B)-y(x<=B)).^2);%Can be calculated in more detail
seperately
    SSE_R2(CombInd)= sum((yAp(B<x&x<=F)-y(B<x&x<=F)).^2);
    SSE_R3(CombInd)= sum((yAp(x>F)-y(x>F)).^2);
end
```

%% Find minimum SSEs for Optimal values

```
fprintf('Min SSE for each region:');
[minSSE_R1,minIndSSE_R1]=min(SSE_R1)
[minSSE_R2,minIndSSE_R2]=min(SSE_R2)
[minSSE_R3,minIndSSE_R3]=min(SSE_R3)

fprintf('Optimal overall SSE across all regions:');
[minSSE_T,minIndSSE_T]=min(SSE_T)
SSE_R1(minIndSSE_T)
SSE_R2(minIndSSE_T)
SSE_R3(minIndSSE_T)

fprintf('B F C D');
OptVals=Combos(minIndSSE_T,:)

    BOp=OptVals(1);
    FOp=OptVals(2);
    COp=OptVals(3);
    DOp=OptVals(4);


yOpt=zeros(1,lenx);
for ind=1:lenx
        if x(ind)<=BOp
            m1=(DOp-COp)/BOp;
            yOpt(ind)=m1*x(ind)+COp;
        elseif BOp<x(ind)&&x(ind)<=FOp
            m2=-DOp/(FOp-BOp);
            c2=(DOp/(FOp-BOp))*FOp;
            yOpt(ind)=m2*(x(ind)-BOp)+DOp;
        else
            yOpt(ind)=0;
        end
end
```

%% Plot resulting BFL based on exhaustive search

```
plot(x,y)
hold on
plot(x,yOpt,'r')
scatter([0,BOp,FOp]',[COp,DOp,0]','filled')
text(0.1,COp,['C_{Op} = ',num2str(COp)],'FontSize',11)
text(BOp+0.3,DOp,['(B_{Op},D_{Op}) = ','(',num2str(BOp), ' , ',
num2str(DOp),')'],'FontSize',11)
text(FOp,0.05,['F_{Op} = ',num2str(FOp)],'FontSize',11)
title('BFL Approximation')
ylabel('f_{BFL}(|x|)')
xlabel('|x|')
grid on
hold off
```

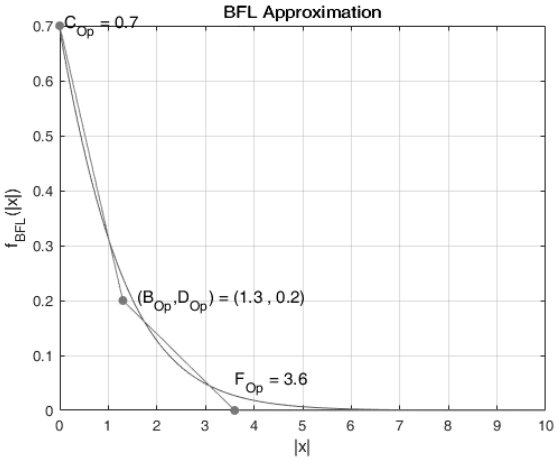


Figure A-E1 resulting output from BFL MATLAB code for iteration 1 (Table 3-12)

APPENDIX F

Title: 4G-LTE Advance Turbo codes BER MATLAB script

Brief description: 4G-LTE Advance Turbo codes simulation to generate BER curves of various suboptimal Turbo decoder in Figure 4-2.

Reference: Section 4.2.1

```
clear
rng(10,'twister') %Initialize and shuffle between seed with Mersenne 'twister'
```

%% Parameters

```
FRAMES=50000;
IP_FRAME_SIZE=637;
ENC_OP_LEN=IP_FRAME_SIZE+3;
ITER=10; %Number of iterations
RATE=1/2; %code rate
VAR_M=0; %0 for known channel, channel coefficient is known by receiver
```

%% SNR Start, interval and stop values

```
START=0;
INTERVAL=0.2;
STOP=3;
plots=zeros(3,length(START:INTERVAL:STOP));
TotalE=zeros(1,length(START:INTERVAL:STOP));
interleaver = get_interleaver(IP_FRAME_SIZE);
PUNCTURE=0; % 1 for yes(rate 1/2) and 0 for no(rate 1/3)

if(PUNCTURE==1)
    RATE=1/2;
end
```

%% Processing for each frame at a time

```
for F=1:FRAMES
```

%% Generate bits of frame F

```
    d1_k=round(rand(1,IP_FRAME_SIZE));
    mgc_d1_k=sign(2*d1_k-1);
    d2_k=intrlv(d1_k,interleaver); % Interleaved d1_k for encoder 2 input
```

%% Encode

```
    [u1_k, v1_k]=enc5G(d1_k);
    [u2_k, v2_k]=enc5G(d2_k); %u2_k not transmitted as it is the interleaved version of
the input u1_k
```

%% Preparing to transmit

```
    u_k=[u1_k; u2_k; v1_k; v2_k];
    % Modulating
    bpsk_tx=2*u_k-1;
```

%% For each frame, corrupt and decode following ebnodb specified

```
    q=1;
    for ebnodb=START:INTERVAL:STOP
        ebno=10^(ebnodb/10);
        N0=1/ebno;
        sigma=sqrt(N0);
        L_c=4*ebno*RATE; % For no h estimation offset
```

```
%% Real Rayleigh Channel
    h=randn(4,ENC_OP_LEN);

%% Real AWGN noise
    n=sigma*randn(4,ENC_OP_LEN);

%% Apply channel effect
    rx_sig=h.*bpsk_tx+n;

%% Fix phase rotation and fading from channel with equalization
    rx=(abs(rx_sig+h).^2-abs(rx_sig-h).^2)/N0;
    % Demultiplexing
    u1rx = rx(1,:);
    u2rx = rx(2,:);
    u2rx(1:IP_FRAME_SIZE)=intrlv(u1rx(1:IP_FRAME_SIZE),interleaver);
    v1rx = rx(3,:);
    v2rx = rx(4,:);

%% Puncturing
    if (PUNCTURE==1)
        v1rx(2:2:ENC_OP_LEN)=0; %Zero-ing all even index of parity 1
        v2rx(1:2:ENC_OP_LEN)=0; %Zero-ing all odd index of parity 2
    end

%% Decoding
    L_e21=zeros(1,ENC_OP_LEN); % A-priori extrinsic value before iteration
    L_e12=zeros(1,ENC_OP_LEN); % Pre allocation

    for i=1:ITER

%% Decoder 1
        [L_e12, LLR1] = dec5G(L_c,v1rx,u1rx,L_e21);
        L_e12=intrlv(L_e12(1:IP_FRAME_SIZE),interleaver);
        L_e12(IP_FRAME_SIZE+1:ENC_OP_LEN)=0;

%% Decoder 2
        [L_e21, LLR2] = dec5G(L_c,v2rx,u2rx,L_e12);
        L_e21=deintrlv(L_e21(1:IP_FRAME_SIZE),interleaver);
        L_e21(IP_FRAME_SIZE+1:ENC_OP_LEN)=0;
        LLR2(IP_FRAME_SIZE+1:ENC_OP_LEN)=[]; % Delete tail bits
        LLR2=deintrlv(LLR2,interleaver);

%% Magic genie stopping rule
        if isequal(mgc_d1_k,sign(LLR2))
            break
        end
    end

%% BER
    [Terr,BER] = biterr(round(0.5.*(sign(LLR2)+1)),d1_k);
    q=q+1;
  end
  TotalE=TotalE+plots(3,:);
end
```

%% Plotting BER curve

```
figure(1)
x=plots(1,:); %ebnodb
y=TotalE/(IP_FRAME_SIZE*FRAMES); %BER
semilogy(x,y,'r')
xlabel('E_b/N_o')
ylabel('BER')
% legend('Ray LM')
xlim([START STOP])
hold all
```

Title: MATLAB function for Turbo encoder (`enc5G`)

Brief description: 4G-LTE Advanced Turbo encoder function

```
function [u_k, v_k, t_b]=enc5G(rsc1_in) %define RSC function
```

```
encoder_mem=[0 0 0]; %input, and 3 memory element
v_k=zeros(1,length(rsc1_in)); %Output v_k
t_b=zeros(1,2); % Tail bits

for i=1:length(rsc1_in)
    mem1p=mod(rsc1_in(i)+encoder_mem(2)+encoder_mem(3),2);
    v_k(i)=mod(mem1p+encoder_mem(1)+encoder_mem(3),2);

    %Time index increment, shift register
    encoder_mem(3)=encoder_mem(2);
    encoder_mem(2)=encoder_mem(1);
    encoder_mem(1)=mem1p;
end

u_k=rsc1_in;
```

%% Trellis termination

```
%Short feedback path to input.

%Bit length increases by 3 bits
for i=length(rsc1_in)+1:length(rsc1_in)+3;

    u_k(i)=mod(encoder_mem(2)+encoder_mem(3),2);
    v_k(i)=mod(0+encoder_mem(1)+encoder_mem(3),2);
    t_b(i-length(rsc1_in))=v_k(i-length(rsc1_in));
%Time index increment, shift register
    encoder_mem(3)=encoder_mem(2);
    encoder_mem(2)=encoder_mem(1);
    encoder_mem(1)=0;

end
```

Title: MATLAB function for Turbo decoder (`dec5G`)

Brief description: 4G-LTE Advanced Turbo decoder function

```
function [L_ex, LLR] = dec5G(L_c,vkrx,ukrx,L_e)
```

```
%No termination
%Inputs
%L_c, channel reliability
%L_e, extrinsic likelihood
%vkrx, received parity
%ukrx, received systematic
%Outputs,
%L_ex, extrinsic likelihood
%LLR, log likelihood ratio
L=length(vkrx);
```

%% Partial Branch Metrics, gamma_e

```
pbm0=(L_c/2)*repmat(vkrx,8,1).*repmat([ -1 -1 1 1 1 1 -1 -1 ]',1,L);
pbm1=(L_c/2)*repmat(vkrx,8,1).*repmat([ 1 1 -1 -1 -1 -1 1 1 ]',1,L);
```

%% Full Branch Metrics, gamma

```
fbm0=-0.5*(repmat(L_e,8,1)+L_c*repmat(ukrx,8,1))+pbm0;
fbm1= 0.5*(repmat(L_e,8,1)+L_c*repmat(ukrx,8,1))+pbm1;
```

%% Forward Branch Metrics, alpha

```
smfwd=zeros(8,L);
smfwd(2:8,1)=-10^10; %Trellis starts at state 0 at first.
AlDen=zeros(1,L-1);
for k=1:L-1
        smfwd(:,k+1)=[logjac(smfwd(1,k)+fbm0(1,k),smfwd(2,k)+fbm1(2,k))
,logjac(smfwd(4,k)+fbm0(4,k),smfwd(3,k)+fbm1(3,k))...
            ,logjac(smfwd(5,k)+fbm0(5,k),smfwd(6,k)+fbm1(6,k))
,logjac(smfwd(8,k)+fbm0(8,k),smfwd(7,k)+fbm1(7,k))...
            ,logjac(smfwd(2,k)+fbm0(2,k),smfwd(1,k)+fbm1(1,k))
,logjac(smfwd(3,k)+fbm0(3,k),smfwd(4,k)+fbm1(4,k))...
            ,logjac(smfwd(6,k)+fbm0(6,k),smfwd(5,k)+fbm1(5,k))
,logjac(smfwd(7,k)+fbm0(7,k),smfwd(8,k)+fbm1(8,k)) ];
        AlDen(k)=min(smfwd(:,k+1));
        smfwd(:,k+1)=smfwd(:,k+1)- AlDen(k);
end
```

%% Backward Branch metrics, beta

```
smbwd=zeros(8,L);
smbwd(2:8,L)=-10^10; %Trellis is terminated
AlDenB=zeros(1,L-1);
for k=L-1:-1:1

smbwd(:,k)=[logjac(smbwd(1,k+1)+fbm0(1,k+1),smbwd(5,k+1)+fbm1(1,k+1)),logjac(smbwd(5,k+1
)+fbm0(2,k+1),smbwd(1,k+1)+fbm1(2,k+1))...

,logjac(smbwd(6,k+1)+fbm0(3,k+1),smbwd(2,k+1)+fbm1(3,k+1)),logjac(smbwd(2,k+1)+fbm0(4,k+
1),smbwd(6,k+1)+fbm1(4,k+1))...

,logjac(smbwd(3,k+1)+fbm0(5,k+1),smbwd(7,k+1)+fbm1(5,k+1)),logjac(smbwd(7,k+1)+fbm0(6,k+
1),smbwd(3,k+1)+fbm1(6,k+1))...

,logjac(smbwd(8,k+1)+fbm0(7,k+1),smbwd(4,k+1)+fbm1(7,k+1)),logjac(smbwd(4,k+1)+fbm0(8,k+
1),smbwd(8,k+1)+fbm1(8,k+1))];
        AlDenB(k)=min(smbwd(:,k+1));
        smbwd(:,k)=smbwd(:,k)- AlDenB(k);
end
```

%% LLR

```
POS=zeros(8,L);
NEG=zeros(8,L);

NEG=smfwd+fbm0+[smbwd(1,:);smbwd(5,:);smbwd(6,:);smbwd(2,:);smbwd(3,:);smbwd(7,:);smbwd(
8,:);smbwd(4,:)];
POS=smfwd+fbm1+[smbwd(5,:);smbwd(1,:);smbwd(2,:);smbwd(6,:);smbwd(7,:);smbwd(3,:);smbwd(
4,:);smbwd(8,:)];
```

```
POSNum=zeros(1,L);
NEGDen=zeros(1,L);

for k=1:L

POSNum(k)=logjac(logjac(logjac(POS(1,k),POS(2,k)),logjac(POS(3,k),POS(4,k))),logjac(logj
ac(POS(5,k),POS(6,k)),logjac(POS(7,k),POS(8,k))));

NEGDen(k)=logjac(logjac(logjac(NEG(1,k),NEG(2,k)),logjac(NEG(3,k),NEG(4,k))),logjac(logj
ac(NEG(5,k),NEG(6,k)),logjac(NEG(7,k),NEG(8,k))));
 end

LLR=POSNum-NEGDen;
```

%% L_e, extrinsic information

```
L_ex=0.85*(LLR-L_e-L_c*ukrx);
```

Title: Log Jacobian function for various suboptimal approximations

Brief description: 4G-LTE Advanced Turbo decoder `logjac` function. This MATLAB code is identical for CCSDS simulation usage.

```
function [y] = logjac(a,b)
```

%% Approximates all values of correction function for input x>6 to 0

```
if abs(a-b)>6
    y=max(a,b); % correction function is 0
    return % exits logjac function
end
```

%% MAX Log-MAP

```
% x=abs(a-b);
% y=max(a,b);
```

%% Log-MAP

```
y=max(a,b)+log(1+exp(-1*abs(a-b)));%Log MAP
```

%% MLM

```
% y=max(a,b)+0.6931/(2^floor(abs(a-b)+0.5)); %log(2)=0.6931
```

%% LC

```
% x=abs(a-b);
% if x<=2.45
%     y=max(a,b)-0.24*x + 0.596;
% elseif 2.45< x & x <=3.5
%     y=max(a,b)+0.048;
% else
%     y=max(a,b);
% end
```

%% LLM

```
% x=abs(a-b);
% y=max(a,b)+ max(0,0.6931-0.5*x); %log(2)=0.6931
```

%% CLM

```
% x=abs(a-b);
% if x<1.5
%     y=max(a,b)+1/2;
% else
%     y=max(a,b);
% end
```

%% HLM

```
% p=1.5;
% q=0.5;
% x=abs(a-b);
% if x<p
%     y=max(a,b)+0.6512-0.3251*x;
% else
%     y=max(a,b)+0.1635./(2.^floor(q*x));
% end
```

%% BFL

```
% x=abs(a-b);
% if x<=1.4347
%     y=max(a,b)+0.6547-0.3323*x;
```

```
% elseif x>1.4347 & x<3.6865
%     y=max(a,b)+0.2914-0.079*x;
% else
%     y=max(a,b);
% end
```

%% BFL4

```
% x=abs(a-b);
% if x<=1.375
%     y=max(a,b)+0.625-0.3125*x;
% elseif x>1.375 & x<3.625
%     y=max(a,b)+0.25-0.0625*x;
% else
%     y=max(a,b);
% end
```

%% BFL6

```
% x=abs(a-b);
% if x<=1.4219
%     y=max(a,b)+0.6406-0.3281*x;
% elseif x>1.4219 & x<3.6719
%     y=max(a,b)+0.2813-0.0781*x;
% else
%     y=max(a,b);
% end
```

%% BFL8

```
% x=abs(a-b);
% if x<=1.4336
%     y=max(a,b)+0.6523-0.3320*x;
% elseif x>1.4336 & x<3.6836
%     y=max(a,b)+0.2891-0.0781*x;
% else
%     y=max(a,b);
% end
```

APPENDIX G

Title: CCSDS Turbo codes BER MATLAB script

Brief description: CCSDS Turbo codes simulation to generate BER curves of various suboptimal Turbo decoder in Figure 4-4.

Reference: Section 4.2.2

%% Turbo encoding and decoding, based on Hanzo

```
F_N=10000; %Number of frames to tx.
rand('state', 14);
dataS=round(rand(1,1000)*1000);

rand('state', 110)
noiseS=round(rand(1,1000)*1000);
```

%% SNR

```
start=0;
interval=0.2;
stop=3;

TotalE=zeros(1,length(start:interval:stop));% Keep tab on total errors for each ebno

for SD=1:F_N %Run for total number of frames, 1 frame at a time
```

%% Generate frame size of 1000 bits

```
rand('state', dataS(SD));
L1=1000; % Input length
d1_k=round(rand(1,L1));
L2=L1+4; %4 tail bits
```

%% Random interleaver

```
intrlv=20; % Change random interleaving seed here
d2_k=randintrlv(d1_k,intrlv);
```

%% Encode

```
%RSC1, no termination
[u1, v1]=Enc16(d1_k);
%RSC2, with termination
[u2, v2]=Enc16(d2_k);
```

%% SNR

```
q=1;
plots=zeros(3,length(start:interval:stop)); %first row, ebnodb, second row, ber, third
row number of error,terr

for ebnodb=start:interval:stop
    ebno=10^(ebnodb/10);
    rate=1/2; % 1/3 if not punctured
    L_c=4*ebno*rate;
    sigma = 1/sqrt(2*rate*ebno);
    randn('state',noiseS(SD));
```

%% Transmit & Puncture

```
    n1=2:2:L2;
    v1(n1)=0; % Deleting all even index of bpsk_v1
    n2=1:2:L2;
    v2(n2)=0; % Deleting all odd index of bpsk_v2
    parity=v1+v2; % Combining parity bits together
```

%% Multiplexing

```
    tx=[u1;parity];
    tx_sig=[tx(:)',u2(L1+1:L2)];
```

%% Modulate, AWGN noise

```
    bpsk_tx= 2*tx_sig -1; % BPSK
    len_tx=length(bpsk_tx);
    rx_sig=bpsk_tx+sigma*randn(1,len_tx); %AWGN noise
```

%% Separating signal to decode, demultiplexing

```
    u2rx_tail=rx_sig(len_tx-3:len_tx); %Decoder2 tail bits
    mux_sig=rx_sig(1:len_tx-4);
    L3=length(mux_sig);
    nu1=1:2:L3;
    u1rx=mux_sig(nu1); % Systematic bits
    u2rx=[randintrlv(u1rx(1:L1),intrlv),u2rx_tail]; % Interleaved Systematic bit with
tail bits
    nu2=2:2:L3;
    parrx=mux_sig(nu2); % Parity bits (punctured)
    v1rx=parrx;
    v1rx(n1)=0;
    v2rx=parrx;
    v2rx(n2)=0;
```

%% Iteration

```
    iter=5; % Number of iteration
    L_e21=zeros(1,L2); % Initializing a-priori L_e to zeros for decoder 1
    for i=1:iter
```

%% Decoder 1

```
        [L_e12, LLR1] = Dec16(L_c,v1rx,u1rx,L_e21);
        L_e12=randintrlv(L_e12(1:L1),intrlv);
        L_e12(L1+1:L2)=0; %Insert L_e=0 for tail bits
```

%% Decoder 2

```
        [L_e21, LLR2] = Dec16(L_c,v2rx,u2rx,L_e12);
        L_e21=randdeintrlv(L_e21(1:L1),intrlv);
        L_e21(L1+1:L2)=0; %Insert L_e=0 for tail bits
        LLR2(L1+1:L2)=[];% Delete tail bits
        LLR2=randdeintrlv(LLR2,intrlv);
    end
```

%% BER

```
    [Terr,BER] = biterr(round(0.5.*(sign(LLR2)+1)),d1_k);
    plots(:,q)=[ebnodb, BER, Terr];
    q=q+1;
end
TotalE=TotalE+plots(3,:);
end
```

%% Plotting BER curve

```
hold all
x=plots(1,:); %ebnodb
y=TotalE/(1000*F_N); %BER
semilogy(x,y,'m')
xlabel('E_b/N_o')
ylabel('BER')
```

Title: MATLAB function for Turbo encoder (`enc16`)

Brief description: CCSDS Turbo encoder function

%% RSC1 Turbo encoder

```
function [u_k, v_k]=Enc16(rsc_in) %define RSC function
```

```
N=length(rsc_in);
encoder_mem=[0 0 0 0]; %4 memory element
v_k=zeros(1,N+4); %Output v_k, parity

for i=1:N
    temp=xor(rsc_in(i),xor(encoder_mem(3),encoder_mem(4)));
    v_k(i)=xor(xor(temp,encoder_mem(1)),xor(encoder_mem(3),encoder_mem(4))); %Non
systematic output, gen poly[11011]

    %Time index increment, shift register
    encoder_mem(4)=encoder_mem(3);
    encoder_mem(3)=encoder_mem(2);
    encoder_mem(2)=encoder_mem(1);
    encoder_mem(1)=temp;
end
u_k=zeros(1,N+4);
u_k(1:N)=rsc_in;
```

%% Trellis termination

```
%Short feedback path to input.
temp=0;
%Bit length increases
for i=N+1:N+4;

    u_k(i)=xor(encoder_mem(3),encoder_mem(4));
    v_k(i)=xor(encoder_mem(1),xor(encoder_mem(3),encoder_mem(4)));

%Time index increment, shift register
    encoder_mem(4)=encoder_mem(3);
    encoder_mem(3)=encoder_mem(2);
    encoder_mem(2)=encoder_mem(1);
    encoder_mem(1)=temp;
end
```

Title: MATLAB function for Turbo decoder (`dec16`)

Brief description: CCSDS Turbo decoder function. The logjac function is identical to the 4G-LTE Advanced logjac function in Appendix F

```
function [L_ex, LLR] = Dec16(L_c,vkrx,ukrx,L_e)
```

```
%No termination
%Inputs
%L_c, channel reliability
%L_e, extrinsic likelihood
%vkrx, received parity
%ukrx, received systematic
%Outputs,
%L_ex, extrinsic likelihood
%LLR, log likelihood ratio
L=length(vkrx);
```

%% Partial Branch Metrics, gamma_e

```
pbm0=(L_c/2)*repmat(vkrx,16,1).*repmat([ -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 1 1]',1,L);
pbm1=(L_c/2)*repmat(vkrx,16,1).*repmat([ 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1]',1,L);
```

%% Full Branch Metrics, gamma

```
fbm0=-0.5*(repmat(L_e,16,1)+L_c*repmat(ukrx,16,1))+pbm0;
fbm1= 0.5*(repmat(L_e,16,1)+L_c*repmat(ukrx,16,1))+pbm1;
```

%% Forward Branch Metrics, alpha

```
smfwd=zeros(16,L);
smfwd(2:16,1)=-10^10; %Trellis starts at state 0 at first.
AlDen=zeros(1,L-1);
AlDen1=zeros(1,8);
AlDen2=zeros(1,8);
for k=1:L-1

smfwd(:,k+1)=[logjac(smfwd(1,k)+fbm0(1,k),smfwd(2,k)+fbm1(2,k)),logjac(smfwd(4,k)+fbm0(4
,k),smfwd(3,k)+fbm1(3,k)),logjac(smfwd(5,k)+fbm0(5,k),smfwd(6,k)+fbm1(6,k)),logjac(smfwd
(8,k)+fbm0(8,k),smfwd(7,k)+fbm1(7,k)), ...

logjac(smfwd(9,k)+fbm0(9,k),smfwd(10,k)+fbm1(10,k)),logjac(smfwd(12,k)+fbm0(12,k),smfwd(
11,k)+fbm1(11,k)),logjac(smfwd(13,k)+fbm0(13,k),smfwd(14,k)+fbm1(14,k)),logjac(smfwd(16,
k)+fbm0(16,k),smfwd(15,k)+fbm1(15,k)),...
            logjac(smfwd(2,k)+fbm0(2,k),smfwd(1,k)+fbm1(1,k))
,logjac(smfwd(3,k)+fbm0(3,k),smfwd(4,k)+fbm1(4,k)),logjac(smfwd(6,k)+fbm0(6,k),smfwd(5,k
)+fbm1(5,k)),logjac(smfwd(7,k)+fbm0(7,k),smfwd(8,k)+fbm1(8,k)),...

logjac(smfwd(10,k)+fbm0(10,k),smfwd(9,k)+fbm1(9,k)),logjac(smfwd(11,k)+fbm0(11,k),smfwd(
12,k)+fbm1(12,k)),logjac(smfwd(14,k)+fbm0(14,k),smfwd(13,k)+fbm1(13,k)),logjac(smfwd(15,
k)+fbm0(15,k),smfwd(16,k)+fbm1(16,k))];
        %AlDen(k)=max(smfwd(:,k+1));

AlDen1(k)=logjac(logjac(logjac(smfwd(1,k+1),smfwd(2,k+1)),logjac(smfwd(3,k+1),smfwd(4,k+
1))),logjac(logjac(smfwd(5,k+1),smfwd(6,k+1)),logjac(smfwd(7,k+1),smfwd(8,k+1))));

AlDen2(k)=logjac(logjac(logjac(smfwd(9,k+1),smfwd(10,k+1)),logjac(smfwd(11,k+1),smfwd(12
,k+1))),logjac(logjac(smfwd(13,k+1),smfwd(14,k+1)),logjac(smfwd(15,k+1),smfwd(16,k+1))))
;
        AlDen(k)=logjac(AlDen1(k),AlDen2(k));
        smfwd(:,k+1)=smfwd(:,k+1)- AlDen(k);
end
```

%% Backward Branch metrics, beta

```
smbwd=zeros(16,L);
smbwd(2:16,L)=-10^10; %Trellis is terminated

for k=L-1:-1:1

smbwd(:,k)=[logjac(smbwd(1,k+1)+fbm0(1,k+1),smbwd(9,k+1)+fbm1(1,k+1)),logjac(smbwd(9,k+1
)+fbm0(2,k+1),smbwd(1,k+1)+fbm1(2,k+1)),logjac(smbwd(10,k+1)+fbm0(3,k+1),smbwd(2,k+1)+fb
m1(3,k+1)),logjac(smbwd(2,k+1)+fbm0(4,k+1),smbwd(10,k+1)+fbm1(4,k+1)), ...

logjac(smbwd(3,k+1)+fbm0(5,k+1),smbwd(11,k+1)+fbm1(5,k+1)),logjac(smbwd(11,k+1)+fbm0(6,k
```

```
+1),smbwd(3,k+1)+fbm1(6,k+1)),logjac(smbwd(12,k+1)+fbm0(7,k+1),smbwd(4,k+1)+fbm1(7,k+1))
,logjac(smbwd(4,k+1)+fbm0(8,k+1),smbwd(12,k+1)+fbm1(8,k+1)),...

logjac(smbwd(5,k+1)+fbm0(9,k+1),smbwd(13,k+1)+fbm1(9,k+1)),logjac(smbwd(13,k+1)+fbm0(10,
k+1),smbwd(5,k+1)+fbm1(10,k+1)),logjac(smbwd(14,k+1)+fbm0(11,k+1),smbwd(6,k+1)+fbm1(11,k
+1)),logjac(smbwd(6,k+1)+fbm0(12,k+1),smbwd(14,k+1)+fbm1(12,k+1)),...

logjac(smbwd(7,k+1)+fbm0(13,k+1),smbwd(15,k+1)+fbm1(13,k+1)),logjac(smbwd(15,k+1)+fbm0(1
4,k+1),smbwd(7,k+1)+fbm1(14,k+1)),logjac(smbwd(16,k+1)+fbm0(15,k+1),smbwd(8,k+1)+fbm1(15
,k+1)),logjac(smbwd(8,k+1)+fbm0(16,k+1),smbwd(16,k+1)+fbm1(16,k+1))];

AlDen3(k)=logjac(logjac(logjac(smbwd(1,k+1),smbwd(2,k+1)),logjac(smbwd(3,k+1),smbwd(4,k+
1))),logjac(logjac(smbwd(5,k+1),smbwd(6,k+1)),logjac(smbwd(7,k+1),smbwd(8,k+1))));

AlDen4(k)=logjac(logjac(logjac(smbwd(9,k+1),smbwd(10,k+1)),logjac(smbwd(11,k+1),smbwd(12
,k+1))),logjac(logjac(smbwd(13,k+1),smbwd(14,k+1)),logjac(smbwd(15,k+1),smbwd(16,k+1))))
;
            AlDenB(k)=logjac(AlDen3(k),AlDen4(k));

    smbwd(:,k)=smbwd(:,k)- AlDenB(k);
end
```

%% LLR

```
NEG=smfwd+fbm0+[smbwd(1,:);smbwd(9,:);smbwd(10,:);smbwd(2,:);smbwd(3,:);smbwd(11,:);smbw
d(12,:);smbwd(4,:);smbwd(5,:);smbwd(13,:);smbwd(14,:);smbwd(6,:);smbwd(7,:);smbwd(15,:);
smbwd(16,:);smbwd(8,:)];
POS=smfwd+fbm1+[smbwd(9,:);smbwd(1,:);smbwd(2,:);smbwd(10,:);smbwd(11,:);smbwd(3,:);smbw
d(4,:);smbwd(12,:);smbwd(13,:);smbwd(5,:);smbwd(6,:);smbwd(14,:);smbwd(15,:);smbwd(7,:);
smbwd(8,:);smbwd(16,:)];

POSNum=zeros(1,L);
NEGDen=zeros(1,L);
for k=1:L

POSNum(k)=logjac(logjac(logjac(logjac(POS(1,k),POS(2,k)),logjac(POS(3,k),POS(4,k))),logj
ac(logjac(POS(5,k),POS(6,k)),logjac(POS(7,k),POS(8,k)))),...

logjac(logjac(logjac(POS(9,k),POS(10,k)),logjac(POS(11,k),POS(12,k))),logjac(logjac(POS(
13,k),POS(14,k)),logjac(POS(15,k),POS(16,k)))));

NEGDen(k)=logjac(logjac(logjac(logjac(NEG(1,k),NEG(2,k)),logjac(NEG(3,k),NEG(4,k))),logj
ac(logjac(NEG(5,k),NEG(6,k)),logjac(NEG(7,k),NEG(8,k)))),...

logjac(logjac(logjac(NEG(9,k),NEG(10,k)),logjac(NEG(11,k),NEG(12,k))),logjac(logjac(NEG(
13,k),NEG(14,k)),logjac(NEG(15,k),NEG(16,k)))));
end
LLR=POSNum-NEGDen;
```

%% L_e, extrinsic information

```
L_ex=LLR-L_e-L_c*ukrx;
```

APPENDIX H

Title: 4G-LTE Advanced Turbo codes BER MATLAB script for SNR Mismatch

Brief description: 4G-LTE Advanced Turbo codes simulation to generate BER curves of various suboptimal Turbo decoder under SNR mismatch conditions in Figure 4-7 and Figure 4-8.

Reference: Section 4.3.1

```
clear
rng(7,'twister') %Initialize and shuffle between seed with Mersenne 'twister'
```

%% Parameters

```
FRAMES=50000; %10K good enough %For SNRMM 500 frames is ok
IP_FRAME_SIZE=637; %640 or 5114 for UMTS
ENC_OP_LEN=IP_FRAME_SIZE+3;
ITER=10; %Number of iterations
RATE=1/2;
VAR_M=0.1; % 0 for known channel
```

%% Fix SNR

```
ebnodb=2.4;
ebno=10^(ebnodb/10);
N0=1/ebno;
sigma=sqrt(N0);
```

%% SNR Offset values

```
START=-6;
INTERVAL=1;
STOP=6;
plots=zeros(3,length(START:INTERVAL:STOP));
TotalE=zeros(1,length(START:INTERVAL:STOP));
interleaver = get_interleaver(IP_FRAME_SIZE);
```

%% Processing for each frame at a time

```
for F=1:FRAMES
```

%% Generate bits of frame F

```
    d1_k=round(rand(1,IP_FRAME_SIZE));
    mgc_d1_k=sign(2*d1_k-1);
    d2_k=intrlv(d1_k,interleaver); % Interleaved d1_k for encoder 2 input
```

%% Encode

```
    [u1_k, v1_k]=enc5G(d1_k);
    [u2_k, v2_k]=enc5G(d2_k); %u2_k not transmitted as it is the interleaved version of
the input u1_k
```

%% Preparing to transmit

%% Modulating

```
    bpsk_tx=2*[u1_k;u2_k;v1_k;v2_k]-1;
    [txR,txC]=size(bpsk_tx);
```

%% For each frame, corrupt and decode following EbN0db specified

```
    q=1;
    for OFFSET=START:INTERVAL:STOP
```

```
ebno_off=10^((ebnodb+OFFSET)/10);
N0_off=1/ebno_off;
L_c=(4*ebno_off)/(VAR_M*(2*ebno_off + 1) +1)*RATE;
```

%% Real Rayleigh Channel

```
h=randn(4,ENC_OP_LEN);
```

%% Rayleigh Channel Estimation offset

```
offset=sqrt(VAR_M)*randn(4,ENC_OP_LEN);
H=h+offset;
```

%% Real AWGN noise

```
n=sigma*randn(4,ENC_OP_LEN);
```

%% Apply channel effect

```
rx_sig= h.*bpsk_tx + n;
```

%% Fix phase rotation and fading from channel with equalization

```
rx=real(rx_sig.*conj(H))/N0_off;% use wrong offset values at the decoder
```

%% De Mux

```
u1rx=rx(1,:);
u2rx=rx(2,:);
v1rx=rx(3,:);
v2rx=rx(4,:);
u2rx(1:IP_FRAME_SIZE)=intrlv(u1rx(1:IP_FRAME_SIZE),interleaver);
```

%% Puncturing

```
    v1rx(2:2:ENC_OP_LEN)=0; %Zero-ing all even index of parity 1
    v2rx(1:2:ENC_OP_LEN)=0; %Zero-ing all odd index of parity 2
```

%% Decoding

```
L_e21=zeros(1,ENC_OP_LEN); % A-priori extrinsic value before iteration
L_e12=zeros(1,ENC_OP_LEN); % Pre allocation
```

%% Decoding

```
for i=1:ITER
```

%% Decoder 1

```
    [L_e12, LLR1] = dec5G(L_c,v1rx,u1rx,L_e21);
    L_e12=intrlv(L_e12(1:IP_FRAME_SIZE),interleaver);
    L_e12(IP_FRAME_SIZE+1:ENC_OP_LEN)=0;
```

%% Decoder 2

```
    [L_e21, LLR2] = dec5G(L_c,v2rx,u2rx,L_e12);
    L_e21=deintrlv(L_e21(1:IP_FRAME_SIZE),interleaver);
    L_e21(IP_FRAME_SIZE+1:ENC_OP_LEN)=0;
    LLR2(IP_FRAME_SIZE+1:ENC_OP_LEN)=[]; % Delete tail bits
    LLR2=deintrlv(LLR2,interleaver);
```

%% Magic genie stopping rule

```
            if isequal(mgc_d1_k,sign(LLR2))
                break
            end
        end
```

%% BER

```
        [Terr,BER] = biterr(round(0.5.*(sign(LLR2)+1)),d1_k);
        plots(:,q)=[OFFSET, BER, Terr];
        q=q+1;
    end
    TotalE=TotalE+plots(3,:);
end
```

%% Plotting BER curve

```
figure(1)
x=plots(1,:); %ebnodb
y=TotalE/(IP_FRAME_SIZE*FRAMES) %BER
semilogy(x,y)
xlabel('E_b/N_o')
ylabel('BER')
xlim([START STOP])
hold all
```

APPENDIX I

Title: CCSDS Turbo codes BER MATLAB script for SNR Mismatch

Brief description: CCSDS Turbo codes simulation to generate BER curves of various suboptimal Turbo decoder under SNR mismatch conditions in Figure 4-16 and Figure 4-17.

Reference: Section4.3.2

%% Seeds

```
F_N=10000; %Number of frames to tx.
rand('state', 14);
dataS=round(rand(1,1000)*1000);

rand('state', 110);
noiseS=round(rand(1,1000)*1000);
```

%% Offsets

```
start=-6;
interval=1;
stop=6;
```

%% SNR is fixed

```
ebnodb=0.3;
ebno=10^((ebnodb)/10);
rate=1/2;
sigma = 1/sqrt(2*rate*ebno);

TotalE=zeros(1,length(start:interval:stop));% Keep tab on total errors for each offset

for SD=1:F_N %Run for total number of frames, 1 frame at a time
```

%% Generate frame size of 1000 bits

```
rand('state', dataS(SD));
L1=1000; % Input length
d1_k=round(rand(1,L1));
L2=L1+4; %4 tail bits
```

%% Random interleaver

```
intrlv=20; % Change random interleaving seed here
d2_k=randintrlv(d1_k,intrlv);
```

%% Encode

```
%RSC1, no termination
[u1, v1]=Enc16(d1_k);
%RSC2, with termination
[u2, v2]=Enc16(d2_k);
```

%% Corrupt signal with AWGN

```
q=1;
plots=zeros(3,length(start:interval:stop)); %first row, Offset, second row, ber, third
row number of error,terr
   for offset=start:interval:stop

     ebnoes=10^((ebnodb+offset)/10);
     L_c=4*ebnoes*rate;
     randn('state',noiseS(SD));
```

%% Transmit

```
    % Puncture
    n1=2:2:L2;
    v1(n1)=0; % Deleting all even index of bpsk_v1
    n2=1:2:L2;
    v2(n2)=0; % Deleting all odd index of bpsk_v2
    parity=v1+v2; % Combining parity bits together

    % Multiplexing
    tx=[u1;parity];
    tx_sig=[tx(:)',u2(L1+1:L2)];

    % Modulate, AWGN noise
    bpsk_tx= 2*tx_sig -1; % BPSK
    len_tx=length(bpsk_tx);
    rx_sig=bpsk_tx+sigma*randn(1,len_tx); %AWGN noise

    % Separating signal to decode, demultiplexing
    u2rx_tail=rx_sig(len_tx-3:len_tx); %Decoder2 tail bits
    mux_sig=rx_sig(1:len_tx-4);
    L3=length(mux_sig);
    nu1=1:2:L3;
    u1rx=mux_sig(nu1); % Systematic bits
    u2rx=[randintrlv(u1rx(1:L1),intrlv),u2rx_tail]; % Interleaved Systematic bit with
tail bits
    nu2=2:2:L3;
    parrx=mux_sig(nu2); % Parity bits (punctured)
    v1rx=parrx;
    v1rx(n1)=0;
    v2rx=parrx;
    v2rx(n2)=0;
```

%% Iteration

```
    iter=5; % Number of iteration
    L_e21=zeros(1,L2); % Initializing a-priori L_e to zeros for decoder 1
    for i=1:iter
```

%% Decoder 1

```
        [L_e12, LLR1] = Dec16(L_c,v1rx,u1rx,L_e21);
        L_e12=randintrlv(L_e12(1:L1),intrlv);
        L_e12(L1+1:L2)=0; %Insert L_e=0 for tail bits
```

%% Decoder 2

```
        [L_e21, LLR2] = Dec16(L_c,v2rx,u2rx,L_e12);
        L_e21=randdeintrlv(L_e21(1:L1),intrlv);
        L_e21(L1+1:L2)=0; %Insert L_e=0 for tail bits
        LLR2(L1+1:L2)=[];% Delete tail bits
        LLR2=randdeintrlv(LLR2,intrlv);
    end
```

%% BER

```
    [Terr,BER] = biterr(0.5.*(sign(LLR2)+1),d1_k);
    plots(:,q)=[offset, BER, Terr];
    q=q+1;
end
TotalE=TotalE+plots(3,:);
end
```

%% Plotting BER curve

```
hold all
x=plots(1,:); %SNR Mismatch
y=TotalE/(1000*F_N); %BER
semilogy(x,y,'r')
xlabel('E_b/N_o Offsets')
ylabel('BER')
```

APPENDIX J

Title: 256-QAM soft bit demodulation sample under 4G-LTE Advanced Turbo code

Brief description: MATLAB codes used to simulate soft bit decoding for the eight states Turbo decoder in wireless mobile standard that results in Figure 4-28.

Reference: Section 4.6.1.

%% Generate bits to transmit

```
M=256; % QAM order
k_bits = log2(M); % Bits per symbol
Proposedbitstosimulate=M*1000*k_bits*5
numSymPerFrame = 125; % Number of modulated symbols per frame
total_frame=10240;
frame_size=k_bits*numSymPerFrame;
normfactor=1/sqrt(2*sqrt(M/4)*sum([1:2:(sqrt(M)-1)].^2)/(M/4)); % normalizing factor for
QAM
TotalBitsSimulated=total_frame*frame_size
DataFrame = randi([0 1], frame_size, total_frame); % generate full data frames to be
transmitted.
interleaver = get_interleaver(frame_size);
enc_tail_bit_len=3;% 4 for CCSDS/ encoder constraint length
enc_op_len=frame_size+enc_tail_bit_len;
ITER=7; %Number of iterations
```

%% Transmission

```
EbNodBVec =(-15:3:30);% Eb/No values (dB)
berEst = zeros(size(EbNodBVec));
snrdB=zeros(1,length(EbNodBVec));
```

%% Setting up transmission for each SNR

```
for n = 1:length(EbNodBVec)
```

%% Convert Eb/No to SNR

```
    snrdB(n)= EbNodBVec(n) + 10*log10(k_bits)
```

%% Reset number of error, bit counters, and soft bit LLR values

```
    numErrs = 0;
    rxdata_llr_LogJac=zeros(1,frame_size);
```

%% Sending data frame by frame

```
      for p=1:total_frame
          Sig=DataFrame(:,p);
```

%% Turbo encoder

```
          intrlvSig=intrlv(Sig,interleaver);

          % Magic Genie sign
          mgc_sig=sign(2*Sig-1);

          % Encode
          [u1_k, v1_k]=enc5G(Sig);
          [u2_k, v2_k]=enc5G(intrlvSig); %u2_k not transmitted as it is the
interleaved version of the input u1_k

          % Puncturing and forming txSig frame
          v1_k(2:2:enc_op_len)=0; %Zero-ing all even index of parity 1
          v2_k(1:2:enc_op_len)=0; %Zero-ing all odd index of parity 2
          parity_vec=v1_k + v2_k;
          txSig_vec=[u1_k';parity_vec];
```

```
            txSig=[txSig_vec(:); zeros(k_bits-rem(enc_op_len*2,k_bits),1)]; % append
zeros to make txSig divisible by 16 with 0 remainder for modulation
```

%% Modulate

```
            txSig_mod =
qammod(txSig,M,'gray','InputType','bit','UnitAveragePower',true);
```

%% Rayleigh with AWGN channel

```
            % AWGN channel noise
            L=length(txSig_mod);
            AvgTxPow=mean(abs(txSig_mod).^2);% measure txSig power, ensures received
signal is guaranteed to have specified SNR
            N0=AvgTxPow*10^(-snrdB(n)/10);
                if (isreal(txSig_mod))
                    noiseVar=N0;
                    noiseSigma=sqrt(N0);
                    noise=noiseSigma*randn(L,1);
                else
                    noiseVar=N0/2;
                    noiseSigma=sqrt(N0/2);
                    noise=noiseSigma*(randn(L,1)+1i*randn(L,1));
                end
```

%% Rayleigh channel

```
            channel_taps=1;% Flat fading Rayleigh
            H = 1/sqrt(2)*(randn(1,channel_taps)+1i*randn(1,channel_taps));
            rxSig1=H.*txSig_mod+noise;
```

%% Equalization

```
              rxSig = pinv(H'*H+N0)*H'*rxSig1; %MMSE
```

%% Soft Bit 256-QAM demodulation of noisy signal

```
            sigma=sqrt(N0/2);
            q=-1/(2*sigma^2);
            wr=real(rxSig);
            wi=imag(rxSig);

            for k=1:L

                LLR0Jac=logjacP(q*(wr(k)-normfactor*[1 3 5 7 9 11 13 15]).^2)-
logjacP(q*(wr(k)-normfactor*[-1 -3 -5 -7 -9 -11 -13 -15]).^2);
                LLR1Jac=logjacP(q*(wr(k)-normfactor*[1 3 5 7 -1 -3 -5 -7]).^2)-
logjacP(q*(wr(k)-normfactor*[9 11 13 15 -9 -11 -13 -15]).^2);
                LLR2Jac=logjacP(q*(wr(k)-normfactor*[5 7 9 11 -5 -7 -9 -11]).^2)-
logjacP(q*(wr(k)-normfactor*[1 3 13 15 -1 -3 -13 -15]).^2);
                LLR3Jac=logjacP(q*(wr(k)-normfactor*[3 5 11 13 -3 -5 -11 -13]).^2)-
logjacP(q*(wr(k)-normfactor*[1 7 9 15 -1 -7 -9 -15]).^2);

                LLR4Jac=logjacP(q*(wi(k)-normfactor*[-1 -3 -5 -7 -9 -11 -13 -15]).^2)-
logjacP(q*(wi(k)-normfactor*[1 3 5 7 9 11 13 15]).^2);
                LLR5Jac=logjacP(q*(wi(k)-normfactor*[1 3 5 7 -1 -3 -5 -7]).^2)-
logjacP(q*(wi(k)-normfactor*[9 11 13 15 -9 -11 -13 -15]).^2);
                LLR6Jac=logjacP(q*(wi(k)-normfactor*[5 7 9 11 -5 -7 -9 -11]).^2)-
logjacP(q*(wi(k)-normfactor*[1 3 13 15 -1 -3 -13 -15]).^2);
                LLR7Jac=logjacP(q*(wi(k)-normfactor*[3 5 11 13 -3 -5 -11 -13]).^2)-
logjacP(q*(wi(k)-normfactor*[1 7 9 15 -1 -7 -9 -15]).^2);

                rxdata_llr_LogJac((k_bits*(k-1)+1):(k_bits*k))=[LLR0Jac, LLR1Jac,
LLR2Jac , LLR3Jac, LLR4Jac, LLR5Jac, LLR6Jac, LLR7Jac];
            end

            rxdata_LogJac=rxdata_llr_LogJac;
```

%% Turbo Decoder

%% Removing zero padding

```
            rxdata_LogJac(length(txSig)-(k_bits-
rem(enc_op_len*2,k_bits))+1:length(txSig))=[ ];
            % De Multiplexing
            rxdata_LogJac=reshape(rxdata_LogJac, 2, enc_op_len);
```

```
            u1rx=rxdata_LogJac(1,:);
            v1rx=zeros(1,enc_op_len);
            v2rx=zeros(1,enc_op_len);
            v1rx(1:2:enc_op_len)=rxdata_LogJac(2,1:2:enc_op_len); % odd positions of row
two in rxdata_LogJac
            v2rx(2:2:enc_op_len)=rxdata_LogJac(2,2:2:enc_op_len); % even positions of
row two in rxdata_LogJac
            u2rx=u1rx(1:frame_size);
            u2rx=[intrlv(u2rx,interleaver),zeros(1,enc_tail_bit_len)];
            % Reset all extrinsic values before decoding
            L_e21=zeros(1,enc_op_len); % Dec2 to Dec1
            L_e12=zeros(1,enc_op_len); % Dec1 to Dec2

            % Channel Reliability %% HERE
            L_c=4*abs(H)*sqrt(AvgTxPow)/N0;
```

%% Turbo Decoding %HERE

```
            for i=1:ITER
```

%% Decoder 1

```
                [L_e12, LLR1] = dec5G(L_c,v1rx,u1rx,L_e21);
                L_e12=intrlv(L_e12(1:frame_size),interleaver);
                L_e12(frame_size+1:enc_op_len)=0;
```

%% Decoder 2

```
                [L_e21, LLR2] = dec5G(L_c,v2rx,u2rx,L_e12);
                L_e21=deintrlv(L_e21(1:frame_size),interleaver);
                L_e21(frame_size+1:enc_op_len)=0;
                LLR2(frame_size+1:enc_op_len)=[]; % Delete tail bits
                LLR2=deintrlv(LLR2,interleaver);
```

%% Magic genie stopping rule

```
                if isequal(mgc_sig,sign(LLR2))
                    break
                end
            end

            dec_Sig=sign(LLR2);
            rxdata=round((dec_Sig+1)/2);
            % Calculate the number of bit errors
            nErrors = biterr(Sig,rxdata');

            % Increment the error and bit counters
            numErrs = numErrs + nErrors;
        end

        % Estimate the BER
        berEst(n) = numErrs/TotalBitsSimulated;
end
```

%% Plot BER curve

```
semilogy(snrdB,berEst)
ylabel('BER')
xlabel('SNR (dB)')
```

APPENDIX K

Title: 256-QAM soft bit demodulation sample under CCSDS Turbo code

Brief description: MATLAB codes used to simulate soft bit decoding for the sixteen states Turbo decoder for deep space communication that results in Figure 4-34.

Reference: Section 4.6.2

```
clc
close all
rng(0,'twister')        % Seeding control
clear
```

%% Generate bits to transmit

```
M=256; % QAM order
k_bits = log2(M); % Bits per symbol
Proposedbitstosimulate=M*1000*k_bits*5
numSymPerFrame = 125; % Number of modulated symbols per frame
total_frame=10240;
frame_size=k_bits*numSymPerFrame
normfactor=1/sqrt(2*sqrt(M/4)*sum([1:2:(sqrt(M)-1)].^2)/(M/4)); % normalizing factor for
QAM
TotalBitsSimulated=total_frame*frame_size
DataFrame = randi([0 1], frame_size, total_frame); % generate full data frames to be
transmitted.
interleaver = get_interleaver(frame_size);
enc_tail_bit_len=4;% 4 for CCSDS/ encoder constraint length
enc_op_len=frame_size+enc_tail_bit_len;
ITER=5; %Number of iterations
```

%% Transmission

```
EbNodBVec =(-5:1:9);% Eb/No values (dB)
berEst = zeros(size(EbNodBVec));
snrdB=zeros(1,length(EbNodBVec));
```

%% Setting up transmission for each SNR

```
for n = 1:length(EbNodBVec)
```

%% Convert Eb/No to SNR

```
    snrdB(n)= EbNodBVec(n) + 10*log10(k_bits)
```

%% Reset number of error, bit counters, and soft bit LLR values

```
    numErrs = 0;
    rxdata_llr_LogJac=zeros(1,frame_size);
```

%% Sending data frame by frame

```
        for p=1:total_frame
            Sig=DataFrame(:,p);
```

%% Turbo encoder

```
            intrlvSig=intrlv(Sig,interleaver);

            % Magic Genie sign
            mgc_sig=sign(2*Sig-1);

            % Encode
            [u1_k, v1_k]=Enc16(Sig);
            [u2_k, v2_k]=Enc16(intrlvSig); %u2_k not transmitted as it is the
```

```
interleaved version of the input u1_k

            % Puncturing and forming txSig frame
            v1_k(2:2:enc_op_len)=0; %Zero-ing all even index of parity 1
            v2_k(1:2:enc_op_len)=0; %Zero-ing all odd index of parity 2
            parity_vec=v1_k + v2_k;
            txSig_vec=[u1_k;parity_vec];
            txSig=[txSig_vec(:); zeros(k_bits-rem(enc_op_len*2,k_bits),1)]; % append
zeros to make txSig divisible by 16 with 0 remainder for modulation
```

%% Modulate

```
            txSig_mod =
qammod(txSig,M,'gray','InputType','bit','UnitAveragePower',true);
```

%% QAM constellation diagram plot

```
QAM_plot =
qammod(Sig,M,'gray','InputType','bit','UnitAveragePower',true,'PlotConstella
tion',true);
cd = comm.ConstellationDiagram('ShowReferenceConstellation',false);
```

%% Rayleigh with AWGN channel

```
            % AWGN channel noise
            L=length(txSig_mod);
            AvgTxPow=mean(abs(txSig_mod).^2);                                    % measure
txSig power, ensures received signal is guaranteed to have specified SNR
            N0=AvgTxPow*10^(-snrdB(n)/10);
                if (isreal(txSig_mod))
                    noiseVar=N0;
                    noiseSigma=sqrt(N0);
                    noise=noiseSigma*randn(L,1);
                else
                    noiseVar=N0/2;
                    noiseSigma=sqrt(N0/2);
                    noise=noiseSigma*(randn(L,1)+1i*randn(L,1));
                end
```

%% AWGN channel

```
              rxSig=txSig_mod+noise; % Only AWGN noise
```

%% Constellation diagram

```
cd=comm.ConstellationDiagram('ShowReferenceConstellation',false);
step(cd,rxSig1)
```

%% Soft Bit 256-QAM demodulation of noisy signal

```
            sigma=sqrt(N0/2);
            q=-1/(2*sigma^2);
            wr=real(rxSig);
            wi=imag(rxSig);

            for k=1:L

                LLR0Jac=logjacP(q*(wr(k)-normfactor*[1 3 5 7 9 11 13 15]).^2)-
logjacP(q*(wr(k)-normfactor*[-1 -3 -5 -7 -9 -11 -13 -15]).^2);
                LLR1Jac=logjacP(q*(wr(k)-normfactor*[1 3 5 7 -1 -3 -5 -7]).^2)-
logjacP(q*(wr(k)-normfactor*[9 11 13 15 -9 -11 -13 -15]).^2);
                LLR2Jac=logjacP(q*(wr(k)-normfactor*[5 7 9 11 -5 -7 -9 -11]).^2)-
logjacP(q*(wr(k)-normfactor*[1 3 13 15 -1 -3 -13 -15]).^2);
                LLR3Jac=logjacP(q*(wr(k)-normfactor*[3 5 11 13 -3 -5 -11 -13]).^2)-
logjacP(q*(wr(k)-normfactor*[1 7 9 15 -1 -7 -9 -15]).^2);

                LLR4Jac=logjacP(q*(wi(k)-normfactor*[-1 -3 -5 -7 -9 -11 -13 -15]).^2)-
logjacP(q*(wi(k)-normfactor*[1 3 5 7 9 11 13 15]).^2);
                LLR5Jac=logjacP(q*(wi(k)-normfactor*[1 3 5 7 -1 -3 -5 -7]).^2)-
logjacP(q*(wi(k)-normfactor*[9 11 13 15 -9 -11 -13 -15]).^2);
                LLR6Jac=logjacP(q*(wi(k)-normfactor*[5 7 9 11 -5 -7 -9 -11]).^2)-
logjacP(q*(wi(k)-normfactor*[1 3 13 15 -1 -3 -13 -15]).^2);
                LLR7Jac=logjacP(q*(wi(k)-normfactor*[3 5 11 13 -3 -5 -11 -13]).^2)-
logjacP(q*(wi(k)-normfactor*[1 7 9 15 -1 -7 -9 -15]).^2);

                rxdata_llr_LogJac((k_bits*(k-1)+1):(k_bits*k))=[LLR0Jac, LLR1Jac,
LLR2Jac , LLR3Jac, LLR4Jac, LLR5Jac, LLR6Jac, LLR7Jac];
            end
```

%% Received data

```
            rxdata_LogJac=rxdata_llr_LogJac;
```

%% Turbo Decoder

Removing zero padding

```
            rxdata_LogJac(length(txSig)-(k_bits-
rem(enc_op_len*2,k_bits))+1:length(txSig))=[ ];
            % De Multiplexing
            rxdata_LogJac=reshape(rxdata_LogJac, 2, enc_op_len);
            u1rx=rxdata_LogJac(1,:);
            v1rx=zeros(1,enc_op_len);
            v2rx=zeros(1,enc_op_len);
            v1rx(1:2:enc_op_len)=rxdata_LogJac(2,1:2:enc_op_len); % odd positions of row
two in rxdata_LogJac
            v2rx(2:2:enc_op_len)=rxdata_LogJac(2,2:2:enc_op_len); % even positions of
row two in rxdata_LogJac
            u2rx=u1rx(1:frame_size);
            u2rx=[intrlv(u2rx,interleaver),zeros(1,enc_tail_bit_len)];

            % Reset all extrinsic values before decoding
            L_e21=zeros(1,enc_op_len); % Dec2 to Dec1
            L_e12=zeros(1,enc_op_len); % Dec1 to Dec2

            % Channel Reliability %% HERE
            L_c=4*sqrt(AvgTxPow)/N0;
```

%% Turbo Decoding %HERE

```
            for i=1:ITER
```

%% Decoder 1

```
                [L_e12, LLR1] = Dec16(L_c,v1rx,u1rx,L_e21);
                L_e12=intrlv(L_e12(1:frame_size),interleaver);
                L_e12(frame_size+1:enc_op_len)=0;
```

%% Decoder 2

```
                [L_e21, LLR2] = Dec16(L_c,v2rx,u2rx,L_e12);
                L_e21=deintrlv(L_e21(1:frame_size),interleaver);
                L_e21(frame_size+1:enc_op_len)=0;
                LLR2(frame_size+1:enc_op_len)=[]; % Delete tail bits
                LLR2=deintrlv(LLR2,interleaver);
```

%% Magic genie stopping rule

```
                if isequal(mgc_sig,sign(LLR2))
                    break
                end
            end

            dec_Sig=sign(LLR2);
            rxdata=round((dec_Sig+1)/2);
            % Calculate the number of bit errors
            nErrors = biterr(Sig,rxdata');

            % Increment the error and bit counters
            numErrs = numErrs + nErrors;
        end

        % Estimate the BER
        berEst(n) = numErrs/TotalBitsSimulated;
end
```

%% Plot BER curve

```
semilogy(snrdB,berEst)
ylabel('BER')
xlabel('SNR (dB)')
```

APPENDIX L

Title: XSG designs of various suboptimal Log-MAP approximations

Brief description: MATLAB XSG designs for various hardware implementation for comparative study with the proposed HLM and BFL approximation.

Reference: Section 4.5

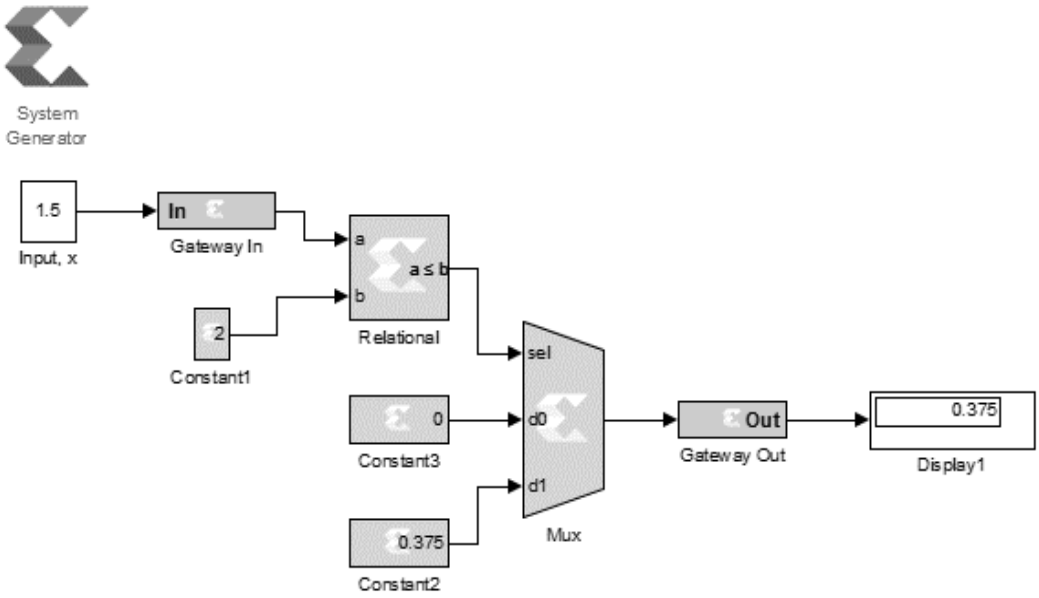


Figure A-L1 XSG CLM design

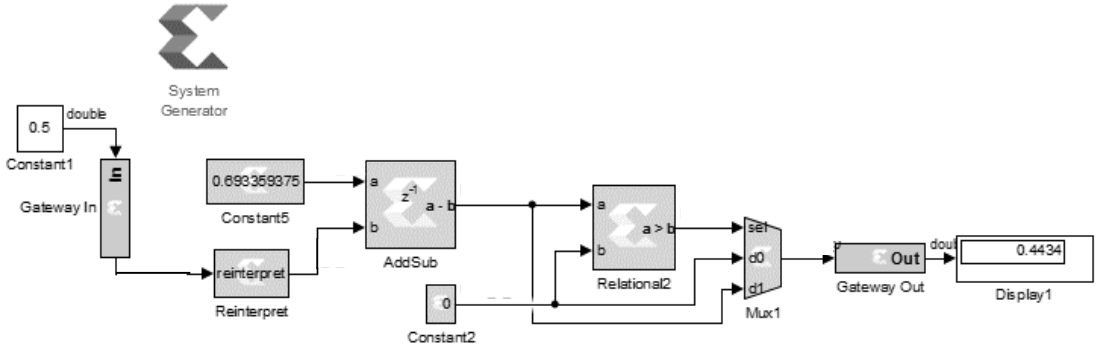


Figure A-L2 XSG LLM design

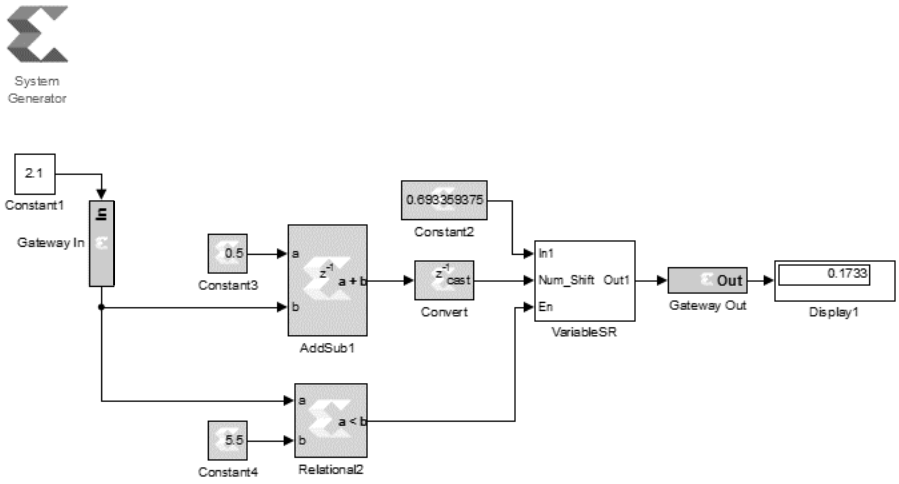


Figure A-L3 XSG MLM design

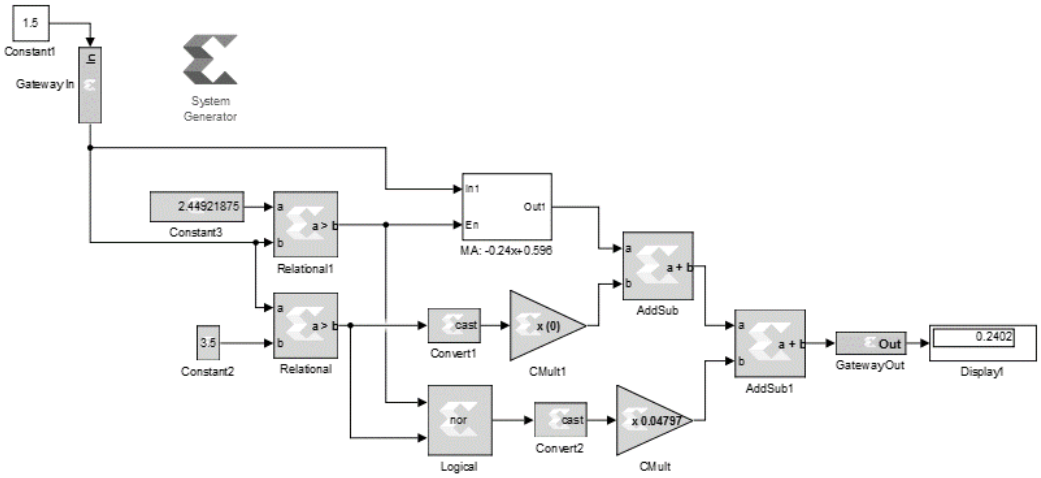


Figure A-L4 XSG LC design