WORCESTER POLYTECHNIC INSTITUE

Log Likelihood Ratio Soft Decision Demapper

An FPGA Implementation for a High Data Rate Modem

Major Qualifying Project Report

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Brian Leslie

October 29th, 2013

Abstract

This Project is sponsored by The MITRE Corporation to develop an FPGA implementation of a Log Likelihood Ratio (LLR) soft decision demapper for a High Data Rate (HDR) modem. The main goal of this project is to add support for higher order modulation up to 32APSK for HDR and high bandwidth efficiency. Through preliminary research, several DVB-S2 soft decision LLR algorithms are investigated for different modulation schemes in order to decide which algorithm will be implemented in synthesizable Hardware Description Language (HDL). Algorithms are analyzed based on performance simulation in MATLAB and complexity analysis. The goal is to improve the performance of current system and provide recommendations for future designs of the soft decision demapper for DVB-S2.

The MAX Algorithm was selected to be implemented based on the complexity of operations used in the calculation and the BER curve performance. The MAX Algorithm is implemented in software using MATLAB. The LLR values are generated using the MAX Algorithm and stored in Lookup Tables (LUT). The VHDL code is designed to use the LLR values in the LUTs by applying bi-linear interpolation to correctly determine the LLR value. The design has been tested extensively in software simulations as well as on hardware. The deliverable for MITRE is the implementation of the selected algorithm in VHDL for deployment on an FPGA. The results of the proposed LLR core demonstrate performance improvement over the existing MITRE core.

Acknowledgments

Steve Attardo from The MITRE Corporation for advising this project. Your dedication to help me understand the entire project every step of the way allowed me to succeed at accomplishing the goals of the project. Thank you for the experience with working with you on this project at MITRE.

Prof. Xinming Huang from Worcester Polytechnic Institute for advising this project, ensuring the project was progressing in the right direction, and offering feedback to the project design, and revising the project report.

Brian McHugh, Roberto Landaru, Jose Torres and Hieu Nguyen from The MITRE Corporation for your guidance and support as supervisors of this project.

Joseph Chapman and Adam Woodbury from The MITRE Corporation for strengthening the partnership between WPI and MITRE for the MQP center.

Prof. Sergey Makarov and Prof. Stephen Bitar from Worcester Polytechnic Institute for introducing me to MITRE and encouraging me to pursue this opportunity for my MQP.

Richard Dennen from Worcester Polytechnic Institute and The MITRE Corporation for being a great friend and mentor throughout my undergraduate years and during this project.

Table of Contents

Abstract 1

Acknowledgments 2

1 Introduction 8

1.1 Background 8

1.1.1 Digital Communication 8

1.1.2 Satellite Communication 9

1.1.3 Digital Video Broadcasting – Satellite Second Generation (DVB-S2) 9

1.2 Problem Statement 11

2 Requirements and Specifications 12

2.1 General Goals 12

2.2 DVB-S2 Standard Specifications 12

2.3 MITRE Requirements 13

3 Methodology 15

3.1 DVB-S2 15

3.1.1 Modulation/Demodulation Schemes 17

3.1.2 FEC Encoding 25

3.2 Log Likelihood Ratio 25

3.2.1 Simplified LLR Explanation 26

3.2.2 LLR Bitwise Comparison 28

3.2.3 True LLR Algorithm 31

3.2.4 MAX Algorithm 33

3.2.5 Euclidean Algorithm 34

3.2.6 3rd party IP LLR Algorithm 35

3.3 Algorithm Comparison 36

3.3.1 Complexity 36

3.3.2 Algorithm Multiplier Comparison 39

3.3.3 Complexity of Operations used in Calculation 40

3.3.2 Performance from MATLAB Models 41

4 Design and Implementation ... 49
4.1 The Chosen Algorithm ... 49
4.1.1 Max Algorithm Block Diagram ... 49
4.2 VHDL Design ... 51
4.2.1 Integration into Current LLR core Architecture ... 51
4.2.2 Integration of MAX Algorithm ... 51
4.2.3 Lookup Table ... 52
4.2.4 Bi-linear Interpolation ... 54
4.2.5 Data Flow ... 59
4.2.6 Rounding ... 61
4.2.7 Timing ... 62
5 Testing and Verification ... 63
5.1 MATLAB Testing ... 63
5.2 C++ Model Testing ... 63
5.3 VHDL Data Flow Testing ... 63
5.4 Hardware Results ... 64
5.4.1 FPGA Slice Logic Utilization ... 65
5.4.2 Digital LDPC Mini Testing ... 66
5.4.3 AGC Scaling ... 67
5.4.4 Final Hardware Results ... 69
6 Conclusion ... 74
Appendix A ... 78
Appendix A.1 – QPSK Modulation (2 BPS) – 10^5 Input Bits ... 78
Appendix A.2 – QPSK Modulation (2 BPS) – 10^6 Input Bits ... 79
Appendix A.3 – 8PSK Modulation (3 BPS) – 10^5 Input Bits ... 82
Appendix A.4 – 8PSK Modulation (3 BPS) – 10^6 Input Bits ... 84
Appendix A.5 – 16APSK Modulation (4 BPS) – 10^5 Input Bits ... 86
Appendix A.6 – 16APSK Modulation (4 BPS) – 10^6 Input Bits ... 88
Appendix A.7 – 32APSK Modulation (5 BPS) – 10^5 Input Bits ... 90
Appendix A.8 – 32APSK Modulation (5 BPS) – 10^6 Input Bits ... 92

Appendix B VHDL Testing Results 94
Appendix B.1 Parallel by Two Testing Results 94
Appendix B.2 Parallel by Four Testing Results 97
Appendix C Terms and Abbreviations 100
Appendix C.1 Field Terminology 100
Appendix D MATLAB Scripts and Functions 101
Appendix D.1 demapper_euclid_algorithm.m 101
Appendix D.2 demapper_MAX_algorithm.m 103
Appendix D.3 demapper_trueLLR_algorithm.m 105
Appendix D.4 generate_DVBS2_BER.m 107
Appendix D.5 generate_llrParams.m 119
Appendix D.6 generate_LUT.m 121
Appendix D.7 demapper_interp_llr.m 124
Appendix D.8 create_test_vectors.m 130
Appendix E MITRE LLR Core 132
Appendix E.1 VHDL 133

List of Figures

Figure 1: The Application - Using DVB-S2 to push high-data-rate application 10
Figure 2: Bit Data Flow Diagram 16
Figure 3: Bit mapping for the QPSK Constellation [14] 18
Figure 4: Bit mapping for the 8PSK Constellation[14] 20
Figure 5: Bit mapping for the 16APSK Constellation [14] 21
Figure 6: Bit mapping for the 32APSK Constellation 23
Figure 7: Constellation Map for 32APSK with DVB-S2 Power Levels 24
Figure 8: Constellation Map for QPSK Demodulation Example 26
Figure 9: Example of 2D LLR Values 28
Figure 10: LLR Values of Bit 0 for 32APSK 29
Figure 11: LLR Values of Bit 1 for 32APSK 29
Figure 12: LLR Values of Bit 2 for 32APSK 30
Figure 13: LLR Values of Bit 3 for 32APSK 30
Figure 14: LLR Values of Bit 4 for 32APSK 31

Figure 15: 8PSK Constellation for LLR Calculation 32
Figure 16: The True LLR Algorithm's LLR bit-wise Calculation Equations 32
Figure 17: The MAX Algorithm's LLR bit-wise Calculation Equations [2] 34
Figure 18: The Euclid Algorithm's LLR bit-wise Calculation Equations [2] 35
Figure 19: True LLR Algorithm Complexity Breakdown 37
Figure 20: MAX Algorithm Complexity Breakdown 38
Figure 21: Euclidean Algorithm Complexity Breakdown 38
Figure 22: 8PSK Modulation Constellation Diagram with PSK Points 39
Figure 23: Hardware BER Results for QPSK, 8PSK, 16APSK, and 32APSK 42
Figure 24: Original LLR Algorithm Comparison Plot for QPSK 43
Figure 25: Original LLR Algorithm Comparison Plot for 8PSK * 44
Figure 26: Original LLR Algorithm Comparison Plot for 16APSK * 44
Figure 27: Original LLR Algorithm Comparison Plot for 32APSK 45
Figure 28: Updated LLR Algorithm Comparison Plot for QPSK 46
Figure 29: Updated LLR Algorithm Comparison Plot for 8PSK 47
Figure 30: Updated LLR Algorithm Comparison Plot for 16APSK 47
Figure 31: Updated LLR Algorithm Comparison Plot for 32APSK 48
Figure 32: Block Diagram One for MAX Algorithm 50
Figure 33: Block Diagram Two for MAX Algorithm 51
Figure 34: LUT Width Diagram 53
Figure 35: Bi-Linear Interpolation Diagram 56
Figure 36: Linear Interpolation for Point R1 57
Figure 37: Linear Interpolation for Point R2 58
Figure 38: Linear Interpolation for Point P 59
Figure 39: LLR Core Data Flow Handshake 60
Figure 40: 3rd party IP Hardware Results for all Modes and Modulations 69
Figure 41: QPSK 1/2 Short BER Performance Comparison 70
Figure 42: QPSK 1/2 Normal BER Performance Comparison 71
Figure 43: QPSK 7/8 Short BER Performance Comparison 72
Figure 44: QPSK 7/8 Normal BER Performance Comparison 73

List of Tables

Table 1: DVB-S2 Demodulation Schemes 18
Table 2: QPSK Symbol Values 19
Table 3: 8PSK Symbol Values 20
Table 4: 16APSK Symbol Values 22
Table 5: 32APSK Symbol Values 24
Table 6: FEC LDPC Coding Rates 25

Table 7: Number of PSK Points in relationship to Modulation 39
Table 8: Multiplier Complexity for LLR Algorithms 39
Table 9: MATLAB Performance Configurations 41
Table 10: LUT Width BER Comparison 54
Table 11: LLR Demapper Modulation Test Plan 65
Table 12: FPGA Slice Logic Utilization for the Proposed MITRE Core 65
Table 13: Slice Logic Utilization for Current MITRE Core and 3rd party IP 32APSK Core 66
Table 14: Slice Logic Utilization for Current MITRE Core and 3rd party IP 64APSK Core 66
Table 15: QPSK 7/8 Normal AGC Scaling Test Results 68
Table 16: QPSK 1/2 Short Full Hardware Results 71
Table 17: QPSK 1/2 Normal Full Hardware Results 72
Table 18: QPSK 7/8 Short Full Hardware Results 73
Table 19: QPSK 7/8 Normal Full Hardware Results 74

1 Introduction

In the past decade there have been numerous innovations and advances in digital communication and multimedia [1]. As a result, the need for satellite communications for numerous applications has grown rapidly [1]. Applications for satellite communication include television, telephone, digital cinema, radio, military, and internet access [2]. Satellite communication describes the transfer of information using the satellite to bridge the transmission and receiving stations. The ability to access High Definition Television (HDTV) via satellite to share information is a primary application of this evolving technology. Digital Video Broadcasting Satellite (DVB-S) standards have been developed and deployed in order to bring digital television to people around the world [3].

1.1 Background

1.1.1 Digital Communication

Digital communication is the backbone for today's society, connecting the world with a high speed, reliable, information infrastructure [4]. The design of a digital communication system starts with describing the channel which includes the received power, available bandwidth, noise statistics, and other impairments such as fading [5]. The data rate and error performance are specific requirements of a digital communication system [5]. Digital communication using satellite communication introduces a level of error because of noise in the physical channel. As a result of this introduction of error, many communication systems are "coded" which refers to the inclusion of error-correction coding schemes [5]. The DVB-S2 standard uses several Forward Error Correction (FEC) code rates which describe the ratio between data bits and parity bits. In order for error-correction to work properly the encoder must provide enough information to compensate for the inaccuracy of the demapper/demodulator [5]. [6, 7]

1.1.2 Satellite Communication

Satellite technology allows for information to be sent and received between locations with long distances from each other, therefore creating a large scale wireless network. This network consists of ground stations which have the equipment necessary to package up the information and convert it into a digital signal sent to the satellite in space. Once a satellite receives the information it needs to redirect that information to the destination, usually another ground station. When there are multiple ground stations and satellites, the result is a communications network that allows a specific ground station to communicate to any other ground station. Linear and non-linear distortions along with phase noise contributions can cause important link performance impairments. With a proper demodulation algorithm and constellation design, it is possible to keep non-ideal satellite channel induced losses within acceptable limits [8].

When a device is connected to a network, the value of that device increases because of its increased ability to communicate with other devices [9]. This connection to the network allows the device to have access to real-time data such that users can make quick and effective decisions. This expandability and flexibility to create a network that is accessible around the world gives military applications an advantage too. Satellite communication has provided the Department of Defense (DOD) a cost effective means of reliable communication [10]. [11, 12]

1.1.3 Digital Video Broadcasting – Satellite Second Generation (DVB-S2)

There is a growing demand in digital communications for more bandwidth, larger margins, greater flexibility, less satellite cost, lower amplifier power, and smaller antenna size. In order for satellite communication to work correctly, standards need to be defined on how information is transmitted and received. These standards have evolved to incorporate changing digital design architectures and algorithms.

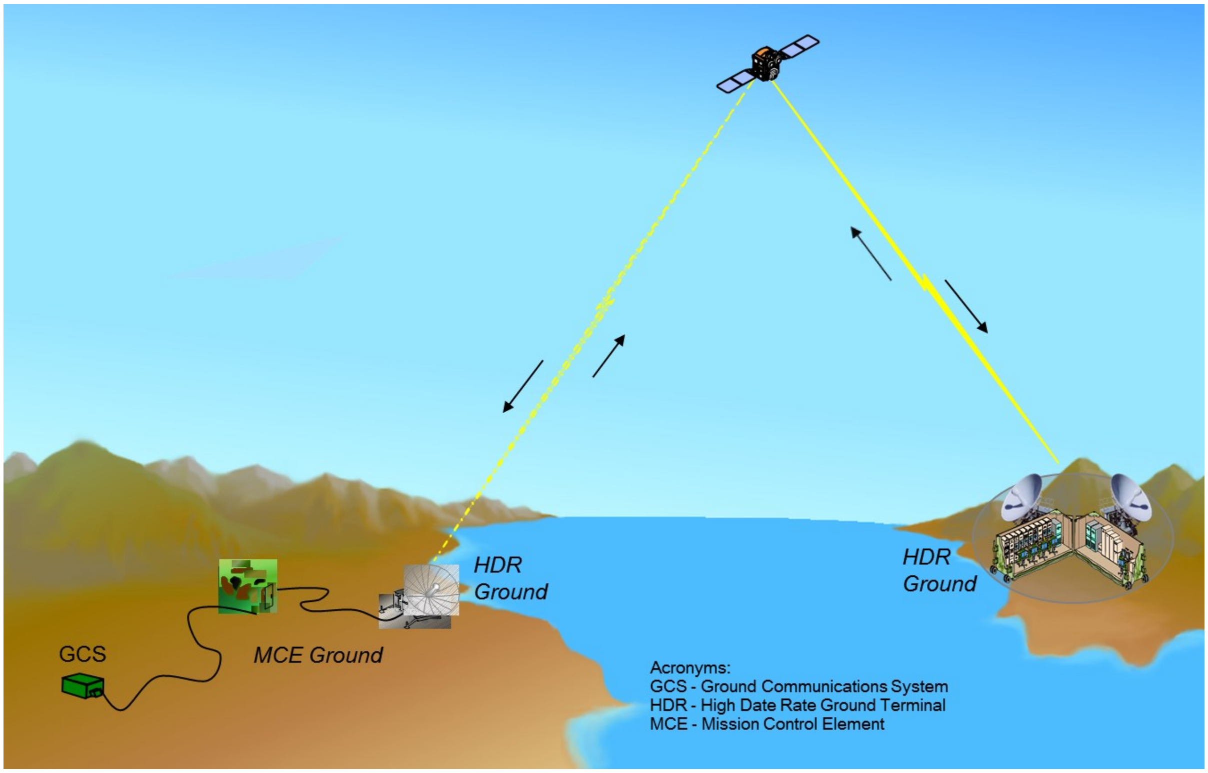

HDR
Ground
HDR
Ground
GCS
MCE Ground
Acronyms:
GCS - Ground Communications System
HDR - High Date Rate Ground Terminal
MCE - Mission Control Element

Figure 1: The Application - Using DVB-S2 to push high-data-rate application

One of the leaders in digital communication standards is the Digital Video Broadcasting – Satellite Second Generation (DVB-S2) developed by the ETSI in 2005. Figure 1 shows the DVB-S2 application with the communication between HDR ground stations. Compared to previous standards this new standard offers more flexibility and more quality levels. The most significant characteristic of the DVB-S2 standard is the ability to select the best coding-modulation scheme to maximize the system throughput [2]. The DVB-S2 standard is often used for high-data-rate applications and therefore the implementation of a LLR soft decision demapper is a key component for a HDR modem.

1.2 Problem Statement

The MITRE Corporation sponsored this project to replace the third party intellectual property (IP) of the LLR demapper implementation. The goal is to achieve an improvement in performance when designing the LLR demapper as well as adding support for higher-order modulation in HDR systems.

There are several challenges for creating the demapper in order to meet all of the project requirements. The proposed design should have better performance than the 3rd party IP core. The hardware complexity increases when applied for higher order modulations. The core must be able to generate the real-time waveform and work seamlessly with other components. The output of a demapper is quantized with a limited number of bits. Also, the core must operate above a specific clock speed. Lastly, there are limitations on the amount of DSP resources available on the FPGA.

The current design of the MITRE LLR core is limited to 8PSK and does not have the flexibility for future expansion. Replacing this core with a new implementation allows for higher modulation schemes like 16APSK and 32APSK included in the DVB-S2 standard. Currently MITRE is using another IP core to handle the higher modulation schemes. This secondary core is from the 3rd party. The baseline requirement is to produce an implementation that provides the same performance as the 3rd party core.

2 Requirements and Specifications

2.1 General Goals

The existing MITRE LLR demapper core lacks expandability for reusable IP. As a result, MITRE has to use a 3rd party IP core for the LLR core in order to implement the other design. This project strives to create a LLR demapper core that can meet MITRE's requirements as well as the specifications defined in the DVB-S2 standard.

The higher order modulation provides a challenge because of the complexity of 16APSK, 32APSK, and higher PSK constellations in the future. By creating the structure for higher-order modulations it allows the design to have both flexibility and expandability.

2.2 DVB-S2 Standard Specifications

The DVB-S2 Standard has several requirements and specifications that are specific to satellite communication [8]. Due to the great distance that a signal needs to travel when being transmitted, high power amplifiers are needed in order to keep a good Signal to Noise Ratio (SNR). The use of these high powered amplifiers causes distortions in the amplified signal because they are non-linear [8]. These distortions impact communication by causing errors in demodulating the signal which results in misinterpreting the received symbols and therefore increases the Bit-Error-Rate (BER) [8].

The following are the terms often used in the DVB-S2 standard:

- *Noise* – It is assumed that the noise distribution follows an Additive White Gaussian Noise (AWGN) distribution.
- *Constellation Design* – DVB-S2 uses Phase Shift Key (PSK) constellation design.

- *Variable Coding and Modulation (VCM)* – This functionality allows DVB-S2 to use and change to different modulations and error protection levels on a frame-by-frame basis thus optimizing transmission parameters for minimizing BER [13].
- *Flexibility* – This standard is not limited to Motion Picture Expert Group (MPEG-2) video and audio coding but able to handle a variety of data formats [13].
- *Code Rates* – The code rates include 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, and 9/10.
- *FECFRAME Length* – The normal frame size is 64,800 bits and the small frame size is 16,200 bits.

2.3 MITRE Requirements

The MITRE Corporation has several requirements for the LLR demapper in order to integrate the new core with their existing waveform design.

- *Modulation* – The ability to demodulate QPSK, 8PSK, 16APSK, and 32APSK modulations to LLR values using a soft decision demapper [2].
- *Performance* – Maintain Quasi-Error Free (QEF) performance with a BER less than 1E-8 at an E_b/N_o close to the current LLR demapper design.
- *Clock* – The clock needs to run at 150MHz with some additional headroom in order for the core to meet timing on a congested FPGA.
- *Flow Control* – The core needs to support flow control on input and output interfaces to handle variable dataflow rates and profiles.
- *Resources* – Minimize resource utilization while maintaining performance.

- *Expandability* – Develop parameterized implementation to maximize reusability for future development.
- *Code* – Follow MITRE's coding guidelines when developing the VHDL core.
- *Testing* – Run a variety of verification tests that stress the core and expose any bugs and issues before deployment on the FPGA.

3 Methodology

3.1 DVB-S2

Digital Video Broadcasting – Satellite – Second Generation is the successor for the popular DVB-S system. This standard defines the second generation modulation and channel coding system. The system is characterized by a flexible input stream adapter which allows for various input formats. This includes a FEC system based on Low-Density Parity Check (LDPC) codes concentrated with BCH codes. This design creates the Quasi-Error-Free operation around 0.7 to 1 dB from the Shannon limit.

The DVB-S2 standard includes several advance techniques in extended range of spectral efficiencies supported and near Shannon power and spectral efficiency [8]. The system includes a wide range of code rates from 1/4 up to 9/10, four modulations and their respecting constellations that includes a spectrum efficiency range from 2 bits/Hz to 5 bits/Hz. The DVB-S2 standard includes Adaptive Coding and Modulation (ACM) which optimizes channel coding and modulation on a frame-by-frame basis. The system has been targeted for broadband satellite applications including HDTV [14].

The DVB-S2 system outlines the specifications and requirements for the LLR demapper design and specifically the bit mapping into constellation. The FECFRAME is converted from serial-to-parallel based on the number of bits per symbol which includes 64,800 bits for normal frame and 16,200 for a short frame. The Most Significant Bit (MSB) of the FECFRAME is shifted into the MSB of the first parallel sequence and therefore each parallel sequence is then mapped to a constellation. This generates a (I, Q) sequence with a length depending on the modulation. The I/Q data represents a modulation symbol of the complex vector where "I" being the in-phase component and "Q" for the quadrature component.

A simplistic graphical explanation of satellite communication can be shown in Figure 2 below. All information that needs to be transmitted starts off as bit data (red box). This block represents the sequence that will be transmitted at the transmitter end and later received at the receiver end. In order for the bit data to be transmitted, it needs to be FEC encoded by adding parity bits. The parity bits allow for error correction by giving additional information to the receiver. The bit data is then mapped into symbols based on the modulation scheme with its associated I/Q values. This complex value is then used to create the signal transmitted through the satellite link.

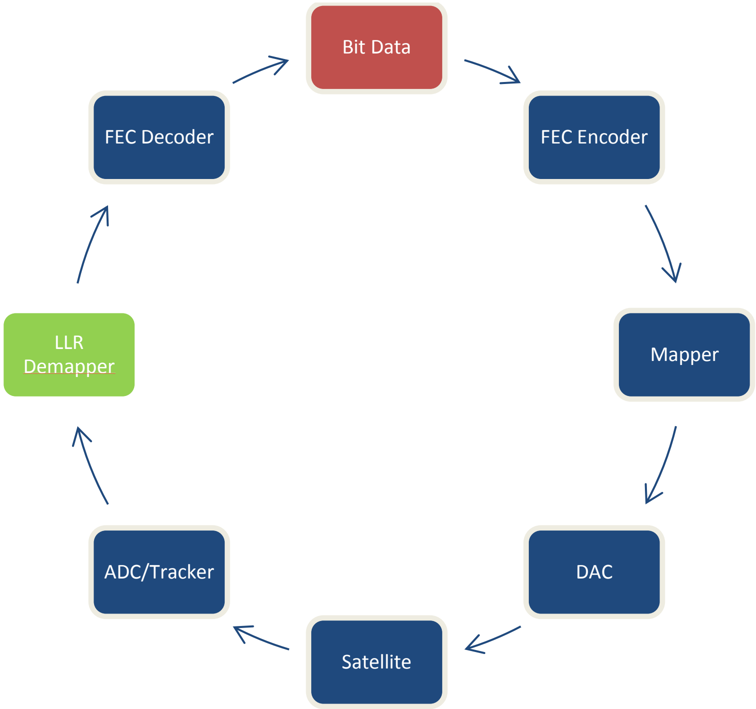


Figure 2: Bit Data Flow Diagram

Once information is transmitted, the satellite is used as the middleman for communications by re-directing the signal to the correct receiving location. On the receiver side the signal is converted back to digital information and then tracked to align the incoming data with known sequences. The complex I/Q values are now received and the LLR demapper determines the probability of each bit in the symbol. The parity bits are then used by the FEC decoder to correct any bit errors caused by the channel and pass the decoder information to the receiver. Ideally, the received information bit data should be exactly the same as the original bit data sent by the transmitter.

3.1.1 Modulation/Demodulation Schemes

In the DVB-S2 standard there are four modulation schemes which include QPSK, 8PSK, 16APSK, and 32APSK. As shown in Table 1, the modulation schemes are named for number of bits per symbol used in the modulation. The QPSK and 8PSK modulations are typically used for broadcast applications because they are constant envelope modulations that can be used in non-linear satellite applications. 16APSK and 32APSK modulation schemes are used for HDR applications and professional broadcasting. These four modulation modes are not the most power efficient as other modes such as Quadrature Amplitude Modulation (QAM) but offer greater spectrum efficiency. Gray mapping of constellations are used for QPSK and 8PSK respectively. Each point on the constellation represents a symbol comprised of the actual sequence of bits [15].

DVB-S2 Demodulation Schemes	Bits Per Symbol (BPS)
QPSK	2
8PSK	3

16APSK	4
32APSK	5

Table 1: DVB-S2 Demodulation Schemes

3.1.1.1 QPSK Modulation

The QPSK modulation is the lowest modulation scheme used in the DVB-S2 standard and has been included in the current implementation of the LLR demapper core. This system uses conventional Gray-coding with absolute mapping. The corresponding bit mapping for the QPSK constellation is shown in Figure 3.

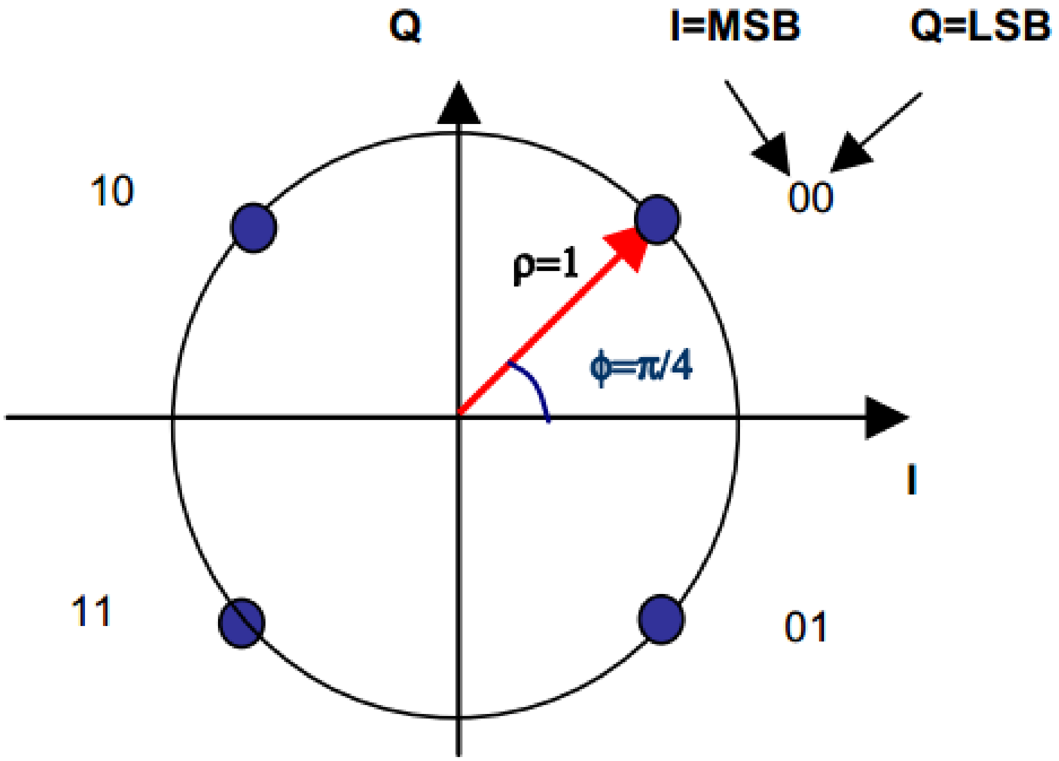


Figure 3: Bit mapping for the QPSK Constellation [14]

For a modulation of QPSK there are two bits per symbol and therefore there are four distinct points on the constellation as shown below in Table 2.

PSK Number	Binary Value

0	00
1	01
2	10
3	11

Table 2: QPSK Symbol Values

3.1.1.2 8PSK Modulation

The 8PSK modulation is the next level modulation scheme above QPSK modulation used in the DVB-S2 standard and was also included in the current implementation of the LLR demapper core. This system, similar to QPSK, also uses conventional Gray-coding with absolute mapping. Unlike the QPSK system which implements a rigid grid-line constellation design, the 8PSK system moves toward a circular design with symbols mapped around the circle at uniform distances. The corresponding bit mapping for the 8PSK constellation is shown in Figure 4.

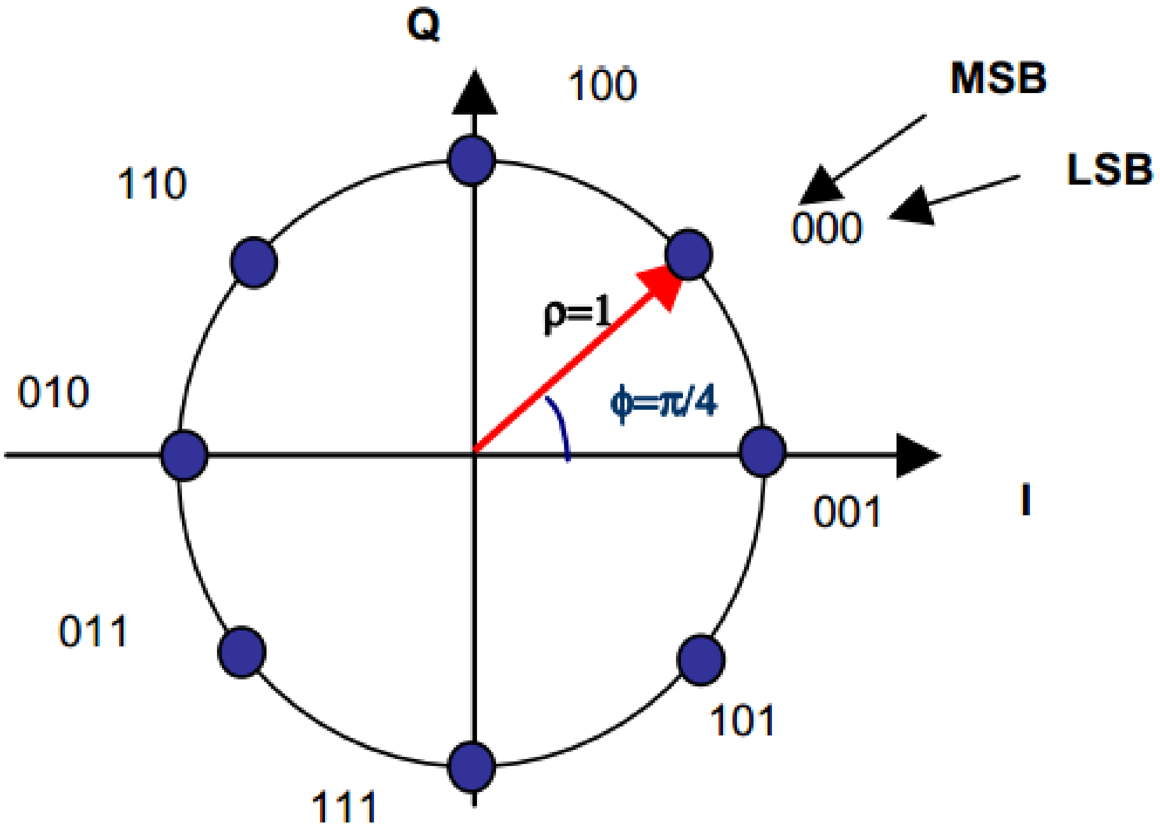


Figure 4: Bit mapping for the 8PSK Constellation[14]

For a modulation of 8PSK there are three bits per symbol and therefore there are eight distinct points on the constellation as shown below in Table 3.

PSK Number	Binary Value
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 3: 8PSK Symbol Values

3.1.1.3 16APSK Modulation

The 16APSK modulation is the next level modulation scheme above 8PSK modulation used in the DVB-S2 standard and has not been included in the current implementation of the LLR demapper core. This system builds off the 8PSK design and includes two concentric rings with the PSK points uniformly spaced around each of the rings. There are two radiuses (R_1 and R_2) which describe the relationship of the power levels between the rings. The corresponding bit mapping for the 16APSK constellation is shown in Figure 5.

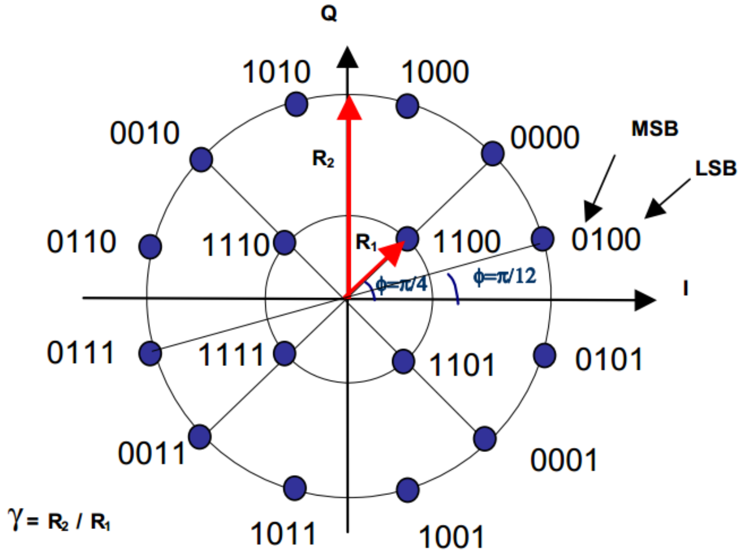


Figure 5: Bit mapping for the 16APSK Constellation [14]

For a modulation of 16APSK there are four bits per symbol and therefore there are sixteen distinct points on the constellation as shown below in Table 4.

PSK Number	Binary Value
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 4: 16APSK Symbol Values

3.1.1.4 32APSK Modulation

The 32APSK modulation is the highest modulation scheme used in the DVB-S2 standard and has not been included in the current implementation of the LLR demapper core. This system builds off the 8PSK and 16APSK designs and includes three concentric rings with the PSK points uniformly spaced around each of the rings. There are three radiuses (R_1, R_2, and R_3) which describe the relationship of the power levels between the rings. The corresponding bit mapping for the 32APSK constellation is shown in Figure 6.

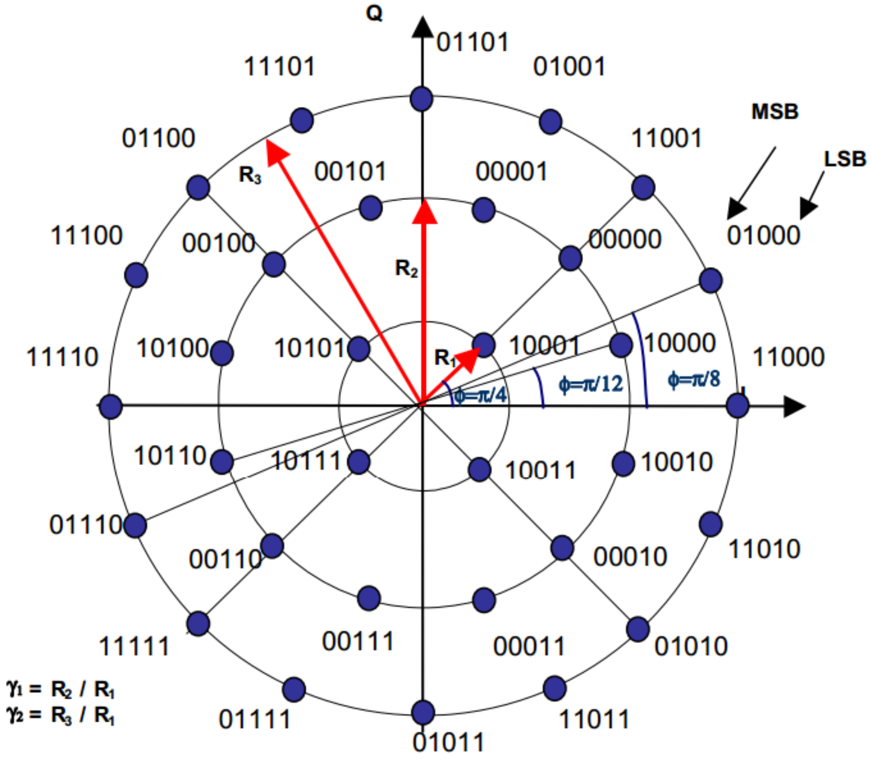


Figure 6: Bit mapping for the 32APSK Constellation

For a modulation of 32APSK there are five bits per symbol and therefore there are thirty-two distinct points on the constellation as shown below in Table 5.

PSK Number	Binary Value	PSK Number	Binary Value
0	00000	16	10000
1	00001	17	10001
2	00010	18	10010
3	00011	19	10011
4	00100	20	10100
5	00101	21	10101
6	00110	22	10110
7	00111	23	10111
8	01000	24	11000
9	01001	25	11001
10	01010	26	11010
11	01011	27	11011
12	01100	28	11100
13	01101	29	11101
14	01110	30	11110

15	01111	31	11111

Table 5: 32APSK Symbol Values

As modulation increases there will be concentric circles added to the design with higher power levels. Power levels are shown by the Figure 7 below on the Q-axis and I-axis. The points that are mapped to the constellation have a power level associated with their bit value which is described by I/Q magnitudes. The outer ring of this constellation has a power level of approximately 90. This outer ring's power level defines the rest of the other inner ring's power levels by using a ratio standard described by DVB-S2.

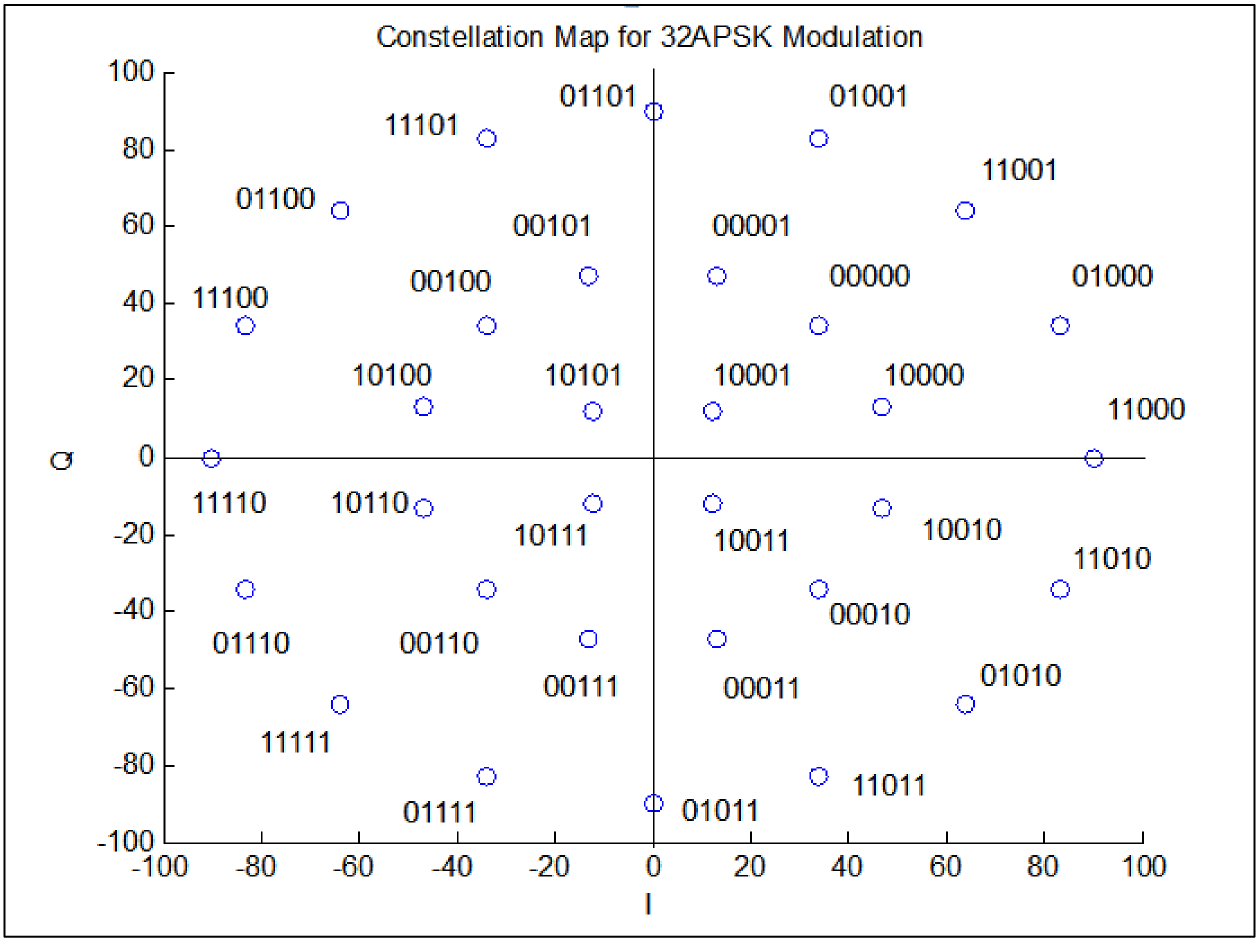


Figure 7: Constellation Map for 32APSK with DVB-S2 Power Levels

3.1.2 FEC Encoding

FEC is a technique in which digital information is transmitted with additional parity bits to limit the number of errors when decoding a noisy communication signal [16]. The redundancy of digital information allows the decoder on the receiver end to detect and fix errors in the received information. The result of using FEC with satellite communication is an increased channel usage and therefore FEC is used because retransmissions are costly or impossible [16]. FEC is also used in satellite communications to reduce the high packet error rate. The DVB-S2 standard uses a specific type of FEC encoding called LDPC codes because of their reliability and high efficiency. The FEC LDPC coding rates used in the DVB-S2 are shown below in Table 6. The FEC coding rates of 1/4, 1/3, and 2/5 are used with QPSK under extremely poor link conditions where the signal level is below the noise level of the SNR [13]. Using smaller order FEC rates while keeping the same modulation has shown to have better performance than decreasing the modulation and increasing the FEC coding rate [13].

FEC LDPC Coding Rates										
1/4	1/3	2/5	1/2	3/5	2/3	3/4	4/5	5/6	8/9	9/10

Table 6: FEC LDPC Coding Rates

3.2 Log Likelihood Ratio

The LLR method is a soft decision technique where, given a set of parameters and possible outcomes, there is a corresponding probability for each of those results. This method is used in soft decision to determine the likelihood of a received symbol based on its mapping to a constellation based on the modulation. The DVB-S2 uses LLR calculations are based on the APSK constellation standard.

3.2.1 Simplified LLR Explanation

Each of these points (blue circles) are located around the I/Q plane in the four quadrants to maximize distinction between the points. The demodulation process involves determining the correct symbol from the symbol received. The symbols are transmitted with that particular I/Q value therefore it can be determined which symbol was transmitted. However, when a symbol is transmitted there is an amount of noise that is added to the signal therefore altering that received I/Q point. For this demodulation it is assumed that the noise is AWGN and therefore the received points will be clustered around those QPSK symbol points.

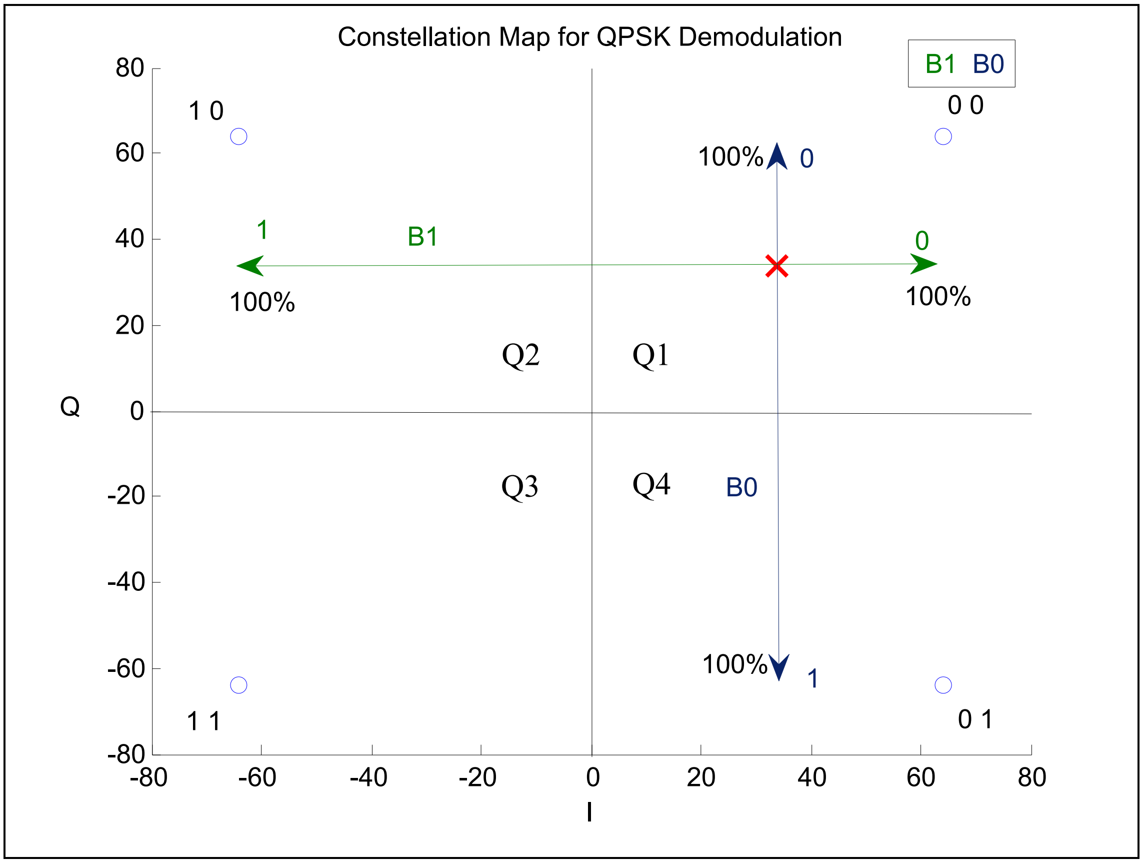


Figure 8: Constellation Map for QPSK Demodulation Example

In order to identify what QPSK symbol was received, an individual LLR calculation is done on the individual bits of the symbol. For the QPSK demodulation shown in Figure 8, there are only two bits, resulting in two comparisons to determine the QPSK symbol. The I/Q plane is split into four different quadrants Q1, Q2, Q3, and Q4. The four QPSK symbols are mapped to these quadrants.

The Least Significant Bit's (LSB) bit value (B0) is 0 for the symbols mapped to quadrant Q1 and Q2. Quadrants Q3 and Q4 have a bit value of 1. The probability of B0's bit value is split vertically across the Q-axis where B0 is expected to have a bit value of 0 above the Q axis and a bit value of 1 below the Q-axis. The received symbol is above the Q-axis and therefore has a higher probability of having a bit value of 0.

The MSB's bit value (B1) is 0 for the symbols mapped to quadrants Q1 and Q4. Quadrants Q2 and Q3 have a bit value of 1. The probability of B1's bit value is split horizontally across the I-axis where B1 is expected to have a bit value of 0 to the right of the I-axis and a bit value of to the left of the I-axis. The received symbol is to the right of the I-axis and therefore has a higher probability of having a bit value of 0.

The result of this analysis is a LLR probability for each particular bit in a symbol for the decoder to interpret. For this example shown in Figure 8, B0 and B1 have a high probability of having a bit value of 0 therefore the received symbol is expected to be the symbol that is mapped in quadrant Q1 (00).

Each PSK point on the I/Q plane has a Gaussian noise distribution associated with it resulting in a function shown in Figure 9 below. The figure illustrates two different PSK points and their overlapping Gaussian noise distribution. The signal level is described by the power level of those PSK points and the LLR probability of the likelihood of a received point at any power level is shown by the Gaussian distribution curves. For this particular example for a power level of zero of a received symbol, there is about an equal probability that the value is at point one or point zero. This is just a simulation and in reality the area under each Gaussian curve should add up to a probability of one.

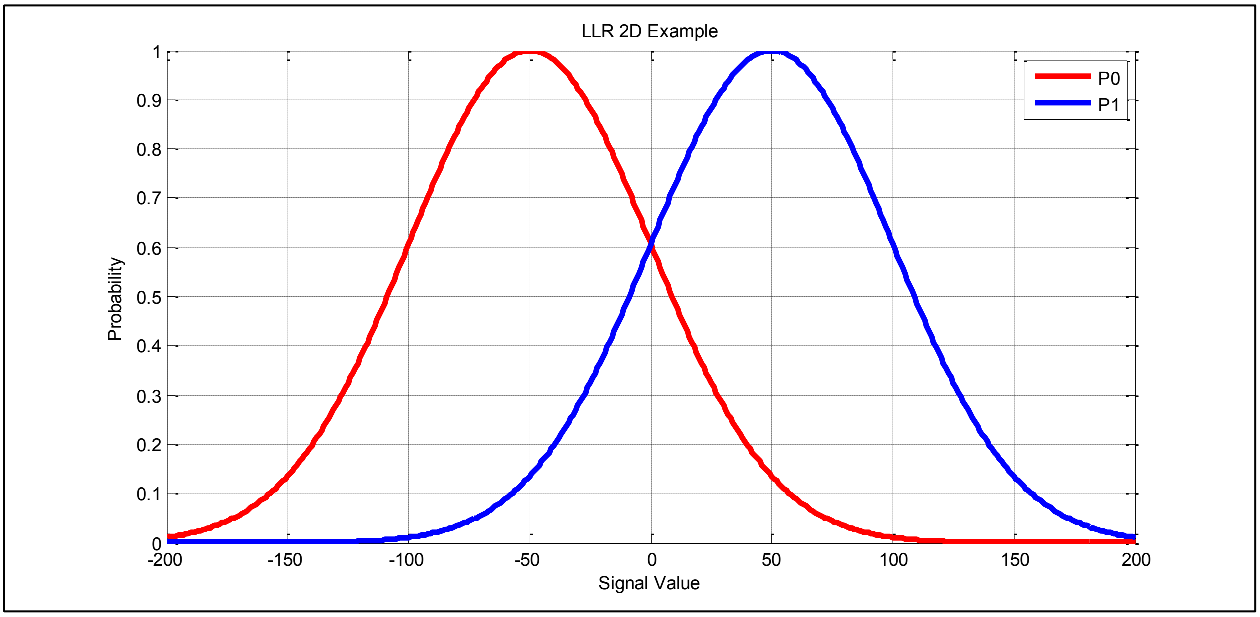


Figure 9: Example of 2D LLR Values

3.2.2 LLR Bitwise Comparison

Based on the 32APSK constellation shown in Figure 7 prior, there are several patterns or tendencies that can be used in used to calculate the LLR value at each data point. The following five figures (Figures 10-14) are 3D plots of the LLR values for their respective I/Q points. Since each bit has an associated LLR value there is a figure for each bit in the five bit 32APSK symbol constellation. The plots are color coded based on the LLR value with a scale to show the transition between LLR values. The darker red represents data points having the highest LLR value which associates to having a higher probability that that particular bit has a value of one. The darker blue represents data points having the lowest LLR value which translates to having a higher probability that that particular bit has a value of zero.

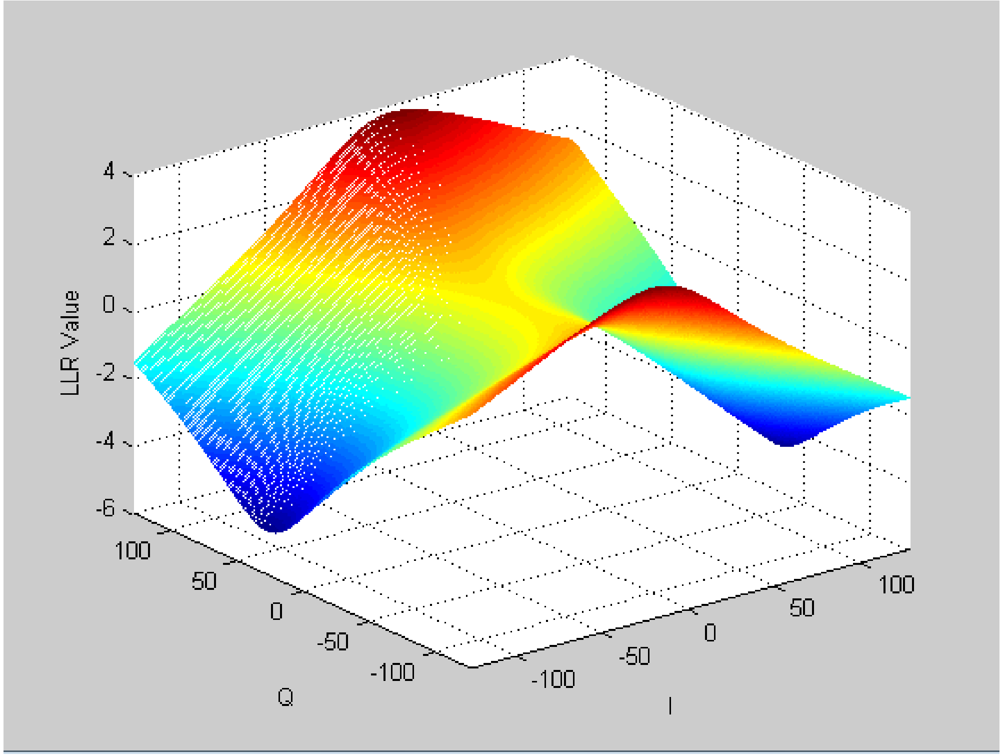


Figure 10: LLR Values of Bit 0 for 32APSK

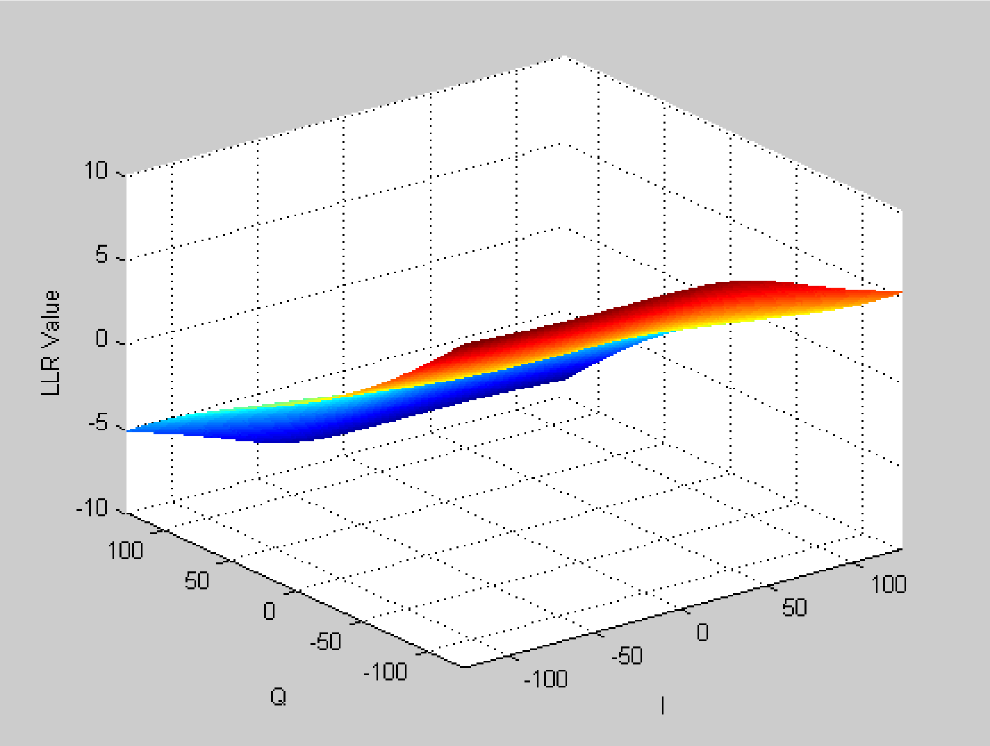


Figure 11: LLR Values of Bit 1 for 32APSK

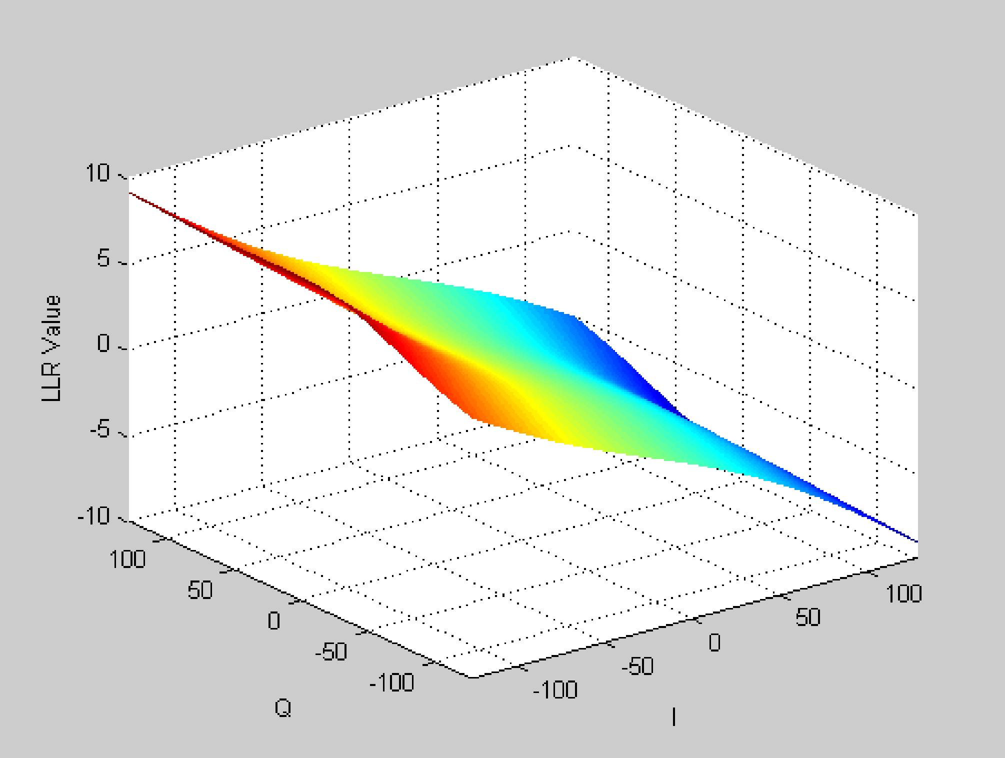


Figure 12: LLR Values of Bit 2 for 32APSK

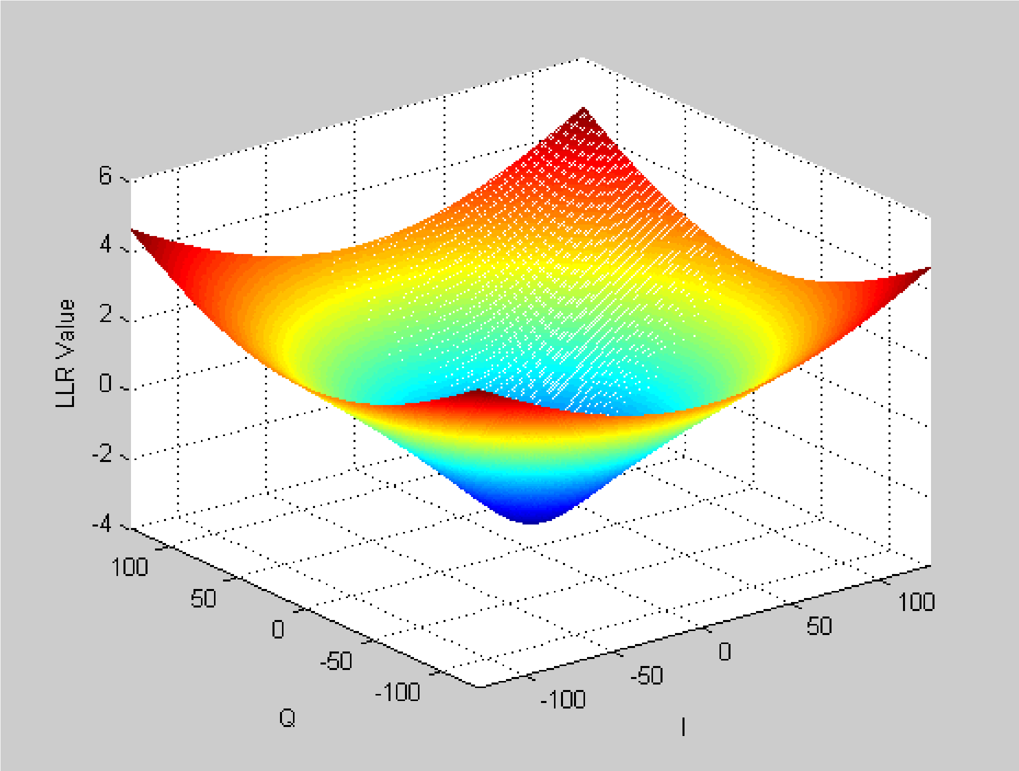


Figure 13: LLR Values of Bit 3 for 32APSK

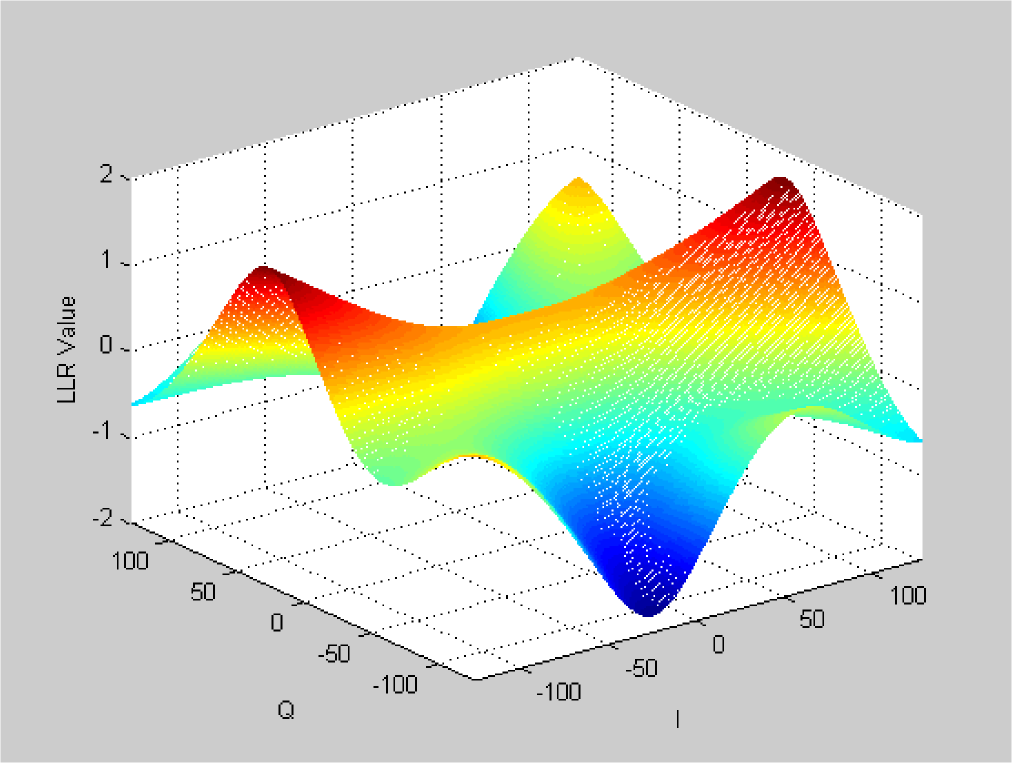


Figure 14: LLR Values of Bit 4 for 32APSK

3.2.3 True LLR Algorithm

The True LLR Algorithm is the closest approximation for the LLR calculation and serves as the basis for all other LLR algorithms. The True LLR Algorithm's accuracy results in a combination of complex hardware and large power consumption due to the complicated mathematical operations. In order to explain the logic behind this algorithm consider how the transmission of a sequence of complex modulation symbols over AWGN is effected by uncompensated frequency error, f_e, and a time-varying phase, Θ, the symbol duration, T, and the sequence of complex noise, n(k), with variance, σ^2. Therefore the discrete-time baseband signal for the receiver can be represented by the following Equation (1). [2]

$$r(k) = e^{j(2*\pi k f_e T+\theta)} + n(k) \tag{1}$$

For an 8-PSK modulation, each symbol consists of 3 bits (b2, b1, b0). PSK Points are located at $\pi/4$ positions around the ring as shown below in figure 15.

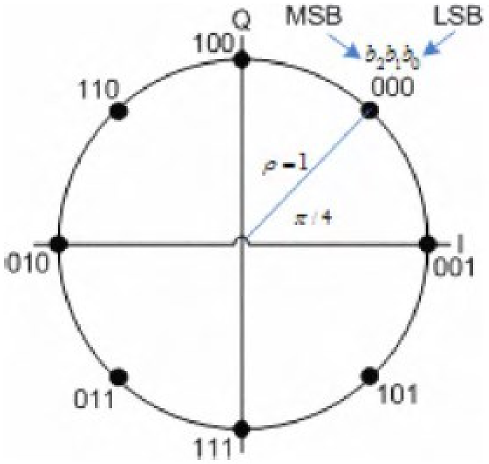


Figure 15: 8PSK Constellation for LLR Calculation

The Probability Density Function (PDF) of the received signals given a transmission at constellation point I is described by Equation (2) where s_i is the symbols in 8PSK.

$$P_i = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} \cdot e^{\frac{-|r-s_i|^2}{2 \cdot \sigma^2}} \tag{2}$$

$$LLR(b_2) = \log \frac{P_0 + P_1 + P_2 + P_3}{P_4 + P_5 + P_6 + P_7}$$

$$LLR(b_1) = \log \frac{P_0 + P_1 + P_4 + P_5}{P_2 + P_3 + P_6 + P_7}$$

$$LLR(b_0) = \log \frac{P_0 + P_2 + P_4 + P_6}{P_1 + P_3 + P_5 + P_7}$$

Figure 16: The True LLR Algorithm's LLR bit-wise Calculation Equations

3.2.4 MAX Algorithm

The MAX Algorithm is an LLR approximation using the two most likely constellation points [2]. The two most likely constellation points are determined using the PDF of each point on the constellation in relation to the received point. The number of constellation points and the bits per symbol depend on the modulation scheme. The calculation of the LLR is determined by the LLR of each bit in the symbol. For a specific bit there is a list of points on the constellation where that particular bit has the value of one or zero. In order to calculate the LLR for each bit, the points where that bit has the value of one or zero are separated into two lists – a list of points where the bit value is zero and a list where the bit value is one. The probability of the bit value of the received point is determined by the difference between the maximum of the zero list when subtracted from the maximum of the one list. The result is a value indicating whether that bit is more likely to be a one or zero. The larger the PDF from the list of ones or zeros will dominate the other and indicate by the magnitude of the result the relative probability.

The probability density function of the received signal given a transmission at constellation points, P_i, is described by Equations (3) and (4) [2]. The PDF equation shown in Equation (3) is equivalent to the exponential term of the True LLR Algorithm PDF calculation. The exponent is not necessary because the MAX Algorithm does not use a direct log calculation. The difference between received point, r, from the expected point on the constellation, s, is divided by the variance of the noise of the received points. That Equation (4) is equivalent to Equation (3) illustrates the fact that the point is comprised of both real and imaginary components and therefore a complex distance calculation is used to get the real result.

$$P_i = \frac{-|r - s_i|^2}{2 \cdot \sigma^2} \tag{3}$$

$$P_i = -\left[(r_{real} - s_{real})^2 + \left(r_{imag} - s_{imag}\right)^2\right] \cdot \frac{1}{2 \cdot \sigma^2} \tag{4}$$

Soft decision de-mapping definitions for 8PSK modulation:

$$LLR(b_2) = \{\max(P_0, P_1, P_2, P_3) - \max(P_4, P_5, P_6, P_7)\}$$
$$LLR(b_1) = \{\max(P_0, P_1, P_4, P_5) - \max(P_2, P_3, P_6, P_7)\}$$
$$LLR(b_0) = \{\max(P_0, P_2, P_4, P_6) - \max(P_1, P_3, P_5, P_7)\}$$

Figure 17: The MAX Algorithm's LLR bit-wise Calculation Equations [2]

3.2.5 Euclidean Algorithm

The Euclidean Algorithm is a LLR approximation that determines the PDF by calculating the distance between the closest point equal to zero and closest point equal to one on the constellation. This is the most basic algorithm implementation because of the limited number of parameters taken into consideration. The calculation of the LLR is determined by the LLR of each bit in the symbol. For a specific bit there is a list of points on the constellation where that particular bit has the value of one or zero. In order to calculate the LLR for each bit, the points where that bit has the value of one or zero are separated into two lists – producing a list of points where the bit value is zero and a list where the bit value is one. The probability of the bit value of the received point is determined by the distance between the minimum of the list of points where that bit is zero is subtracted from the minimum of the list of points where that bit is one. The result is a value indicating whether that bit is more likely to be a one or zero. The smaller distance from the list of ones or zeros will dominate the other and indicate by the magnitude distance of the result the relative probability.

The PDF equation shown in Equation (5) and (6) are the Euclidean distance equation in a simplified and expanded format. Because the algorithm does not use the noise variance in the PDF calculation it is a distance, d, not a PDF that is calculated. The distance is calculated by the difference between the received point and the expected constellation point. Just as in the MAX Algorithm the received point is complex and therefore the real and imaginary components need to be used in order to determine the actual distance.

$$d_i = |x - s_i| \tag{5}$$

$$d_i = \sqrt{(x_{real} - s_{real})^2 + \left(x_{imag} - s_{imag}\right)^2} \tag{6}$$

Soft decision de-mapping definition for an 8-PSK:

$$LLR(b_2) = \{\min(d_{0,}d_{1,}d_{2,}d_3) - \min(d_{4,}d_{5,}d_{6,}d_7)\}$$
$$LLR(b_1) = \{\min(d_{0,}d_{1,}d_{4,}d_5) - \min(d_{2,}d_{3,}d_{6,}d_7)\}$$
$$LLR(b_0) = \{\min(d_{0,}d_{2,}d_{4,}d_6) - \min(d_{1,}d_{3,}d_{5,}d_7)\}$$

Figure 18: The Euclid Algorithm's LLR bit-wise Calculation Equations [2]

3.2.6 3rd party IP LLR Algorithm

In order to accommodate higher order modulations in their waveform, MITRE uses a 3rd Party LLR Implementation which is compiled executable code that is then implemented on the FPGA. This design may not be the best implementation and lacks flexibility by relying on that 3rd party to continue to implement future designs.

3.3 Algorithm Comparison

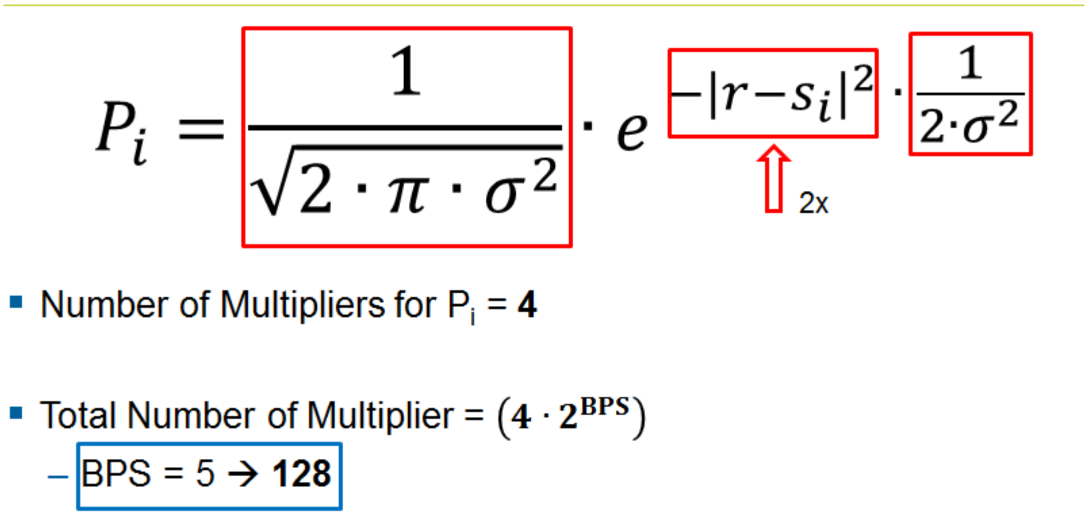
In most systems the True LLR algorithm has been used as the preferred soft decision implementation. However, the True LLR Algorithm has drawbacks that include hardware complexity and power consumption due to complicated operations. Therefore the other algorithms proposed in this document try to reduce these disadvantages to the True LLR Algorithm. The MAX Algorithm reduces the exponential and logarithm functions of the True LLR Algorithm and therefore lowers the hardware complexity compared to the True LLR Algorithm. The Euclidean Algorithm can reduce the number of multiplications but requires the square and root square operations that also lead to higher hardware complexity compared to the MAX Algorithm. The Euclidean Algorithm also has decreased BER performance compared to the MAX and True LLR Algorithms. In conclusion the MAX Algorithm is determined to be the best soft decision demapper for the DVB-S2 system. However in order to implement the MAX Algorithm in hardware, the software requires more complexity to support the four modulation modes in DVB-S2. This complexity is what ultimately changed the design of the core from the LLR Algorithm implementation to the LUT design.

3.3.1 Complexity

The LLR algorithms are compared based on complexity in terms of the number of multipliers required to implement the design on an FPGA.

3.3.1.1 True LLR Algorithm

The True LLR Algorithm is described by the equation below where P_i is the LLR approximation for a given PSK point on the constellation. The equation involves two multiplication operations for each bit because each real and imaginary component requires an individual multiplication calculation. Therefore for 32APSK (a BPS value of 5) there are a total of 128 multipliers for one symbol.



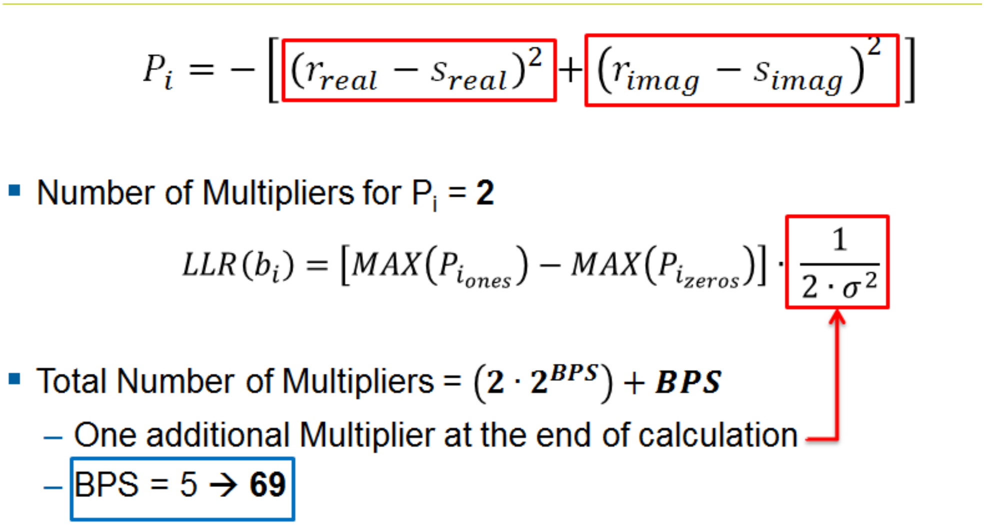
$$P_i = \boxed{\frac{1}{\sqrt{2\cdot\pi\cdot\sigma^2}}}\cdot e^{\underset{\Uparrow\ 2x}{\boxed{-|r-s_i|^2}}\cdot\boxed{\frac{1}{2\cdot\sigma^2}}}$$

- Number of Multipliers for P_i = **4**
- Total Number of Multiplier = $\left(\mathbf{4}\cdot\mathbf{2}^{\mathbf{BPS}}\right)$
  - BPS = 5 → **128**

Figure 19: True LLR Algorithm Complexity Breakdown

3.3.1.2 MAX Algorithm

The MAX Algorithm is described by the equation below where P_i is the LLR approximation for a given PSK point on the constellation The equation involves two multiplication operations for each bit because each real and imaginary component requires an individual multiplication calculation. There is an additional multiplication included in the calculation for each bit in the symbol. Therefore for 32APSK (a BPS value of 5) there are a total of 69 multipliers for one symbol.



$$P_i = -\left[\boxed{(r_{real} - s_{real})^2} + \boxed{\left(r_{imag} - s_{imag}\right)^2}\right]$$

- Number of Multipliers for P_i = **2**

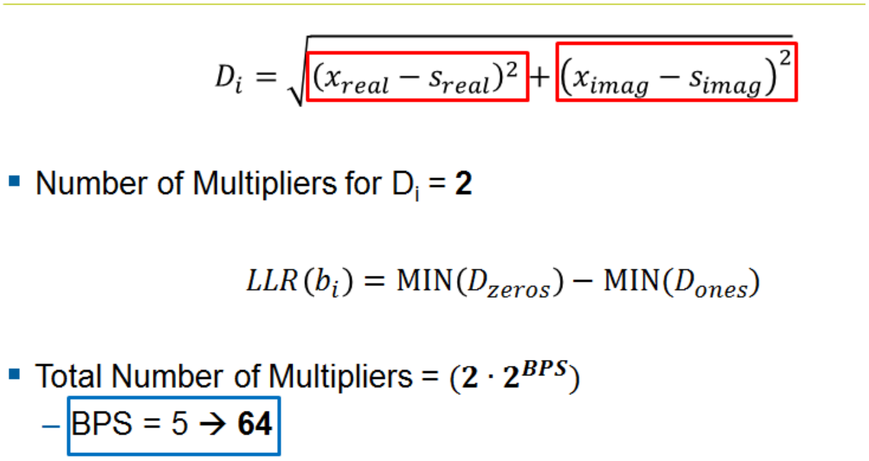
$$LLR(b_i) = \left[MAX\left(P_{i_{ones}}\right) - MAX\left(P_{i_{zeros}}\right)\right] \cdot \boxed{\frac{1}{2 \cdot \sigma^2}}$$

- Total Number of Multipliers = $(\mathbf{2 \cdot 2^{BPS}}) + \boldsymbol{BPS}$
  - One additional Multiplier at the end of calculation
  - BPS = 5 → **69**

Figure 20: MAX Algorithm Complexity Breakdown

3.3.1.3 Euclidean Algorithm

The Euclidean Algorithm is described by the equation below where D_i is the LLR approximation for a given PSK point on the constellation. The equation involves two multiplication operations for each bit because each real and imaginary component requires an individual multiplication calculation. Therefore for 32APSK (a BPS value of 5) there are a total of 64 multipliers for one symbol.



$$D_i = \sqrt{\boxed{(x_{real} - s_{real})^2} + \boxed{\left(x_{imag} - s_{imag}\right)^2}}$$

- Number of Multipliers for D_i = **2**

$$LLR(b_i) = \mathrm{MIN}(D_{zeros}) - \mathrm{MIN}(D_{ones})$$

- Total Number of Multipliers = $(\mathbf{2 \cdot 2^{BPS}})$
  - BPS = 5 → **64**

Figure 21: Euclidean Algorithm Complexity Breakdown

3.3.2 Algorithm Multiplier Comparison

There is a unique PDF, $\mathbf{P_i}$, for each point on the constellation. The number of P_i is equal to 2^{BPS} as shown by the Figure 22 and Table 7 below.

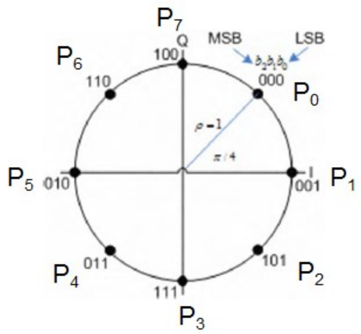


Figure 22: 8PSK Modulation Constellation Diagram with PSK Points

Modulation (BPS)	Number of PSK Points
QPSK (2)	4
8PSK (3)	8
16APSK (4)	16
32APSK (5)	32

Table 7: Number of PSK Points in relationship to Modulation

The results of the complexity analysis showed the MAX and Euclidean Algorithms both having similar complexities and the True LLR Algorithm having a significantly larger complexity in terms of the number of multiplications necessary to calculate the LLR. The analysis was done for one PDF calculation where the number of PDF calculations for one LLR calculation is related to modulation scheme. Table 8 below illustrates the results for 32APSK modulation LLR calculation and is based on the following assumptions:

- 32APSK Modulation (5 BPS)
- Number of constellation points (P_i) = 32
- Number of bit LLR calculations (B_i) = 5

	Number of Multipliers		
	True LLR	MAX	Euclid
Each P_i	4	2	2
Each B_i	0	1	0
Total	128	69	64

Table 8: Multiplier Complexity for LLR Algorithms

3.3.3 Complexity of Operations used in Calculation

In order for one of these algorithms to be implemented on the FPGA, the complexity of the mathematic calculations needs to be taken into consideration. Mathematic calculations such as a logarithm, exponential, or square root require additional hardware components and therefore pose a challenge in the design. Each algorithm was specifically investigated in order to determine the complexity of the mathematical operations used in the LLR calculation. The True LLR Algorithm includes several high order calculations and therefore would make the design on hardware nearly impossible with limited resources. The two approximation methods that were considered do not include those calculations and therefore allow for straightforward design implementation.

- **True LLR Algorithm**
  - Multiplication, Addition, Subtraction
  - <u>Exponential</u>
  - <u>Square root</u>
  - <u>Log</u>
  - Variance
- **MAX Algorithm**
  - Multiplication, Addition, Subtraction
  - Maximum
  - Variance
- **Euclidean Algorithm**
  - Multiplication, Addition, Subtraction
  - <u>Square Root</u>
  - Minimum

The MAX Algorithm was chosen for a limited hardware complexity because it did not include any mathematical calculations that would involve other hardware designs or components. The Variance calculation would pose an immediate challenge but solutions to the overall design were introduced before a decision was made on how to include this component of the calculation.

3.3.2 Performance from MATLAB Models

One of the main project goals is to improve the overall performance of the current LLR core. The new core was implemented based on the performance of the algorithm selection considered. In order to determine the design with the best overall performance, the algorithms were compared by generating BER curves in MATLAB. The results were separated based on modulation. In order to correctly mimic the Automatic Gain Control (AGC) scaling use on hardware, a sweep of scaling values was used to determine the best power level. The best case performance of each algorithm was compared to determine the best algorithm to use for the new LLR core design. The original comparisons were done with limited configuration including a smaller FECFRAME length, limited number of interactions, and a general scaling across all modulations to get a basis for the performance results. The tests were then updated to include the normal FECFRAME length, variation of iterations, and specific scaling based on the modulation.

MATLAB Performance Configurations	
Original	**Updated**
Small FECFRAME Length (16,200 bits)	Normal FECFRAME Length (64,800 bits)
Limited Iterations	Variation of Iterations
General Scaling	Specific Scaling

Table 9: MATLAB Performance Configurations

The results of the performance testing are compared to the hardware results of the current LLR core shown below in Figure 23.

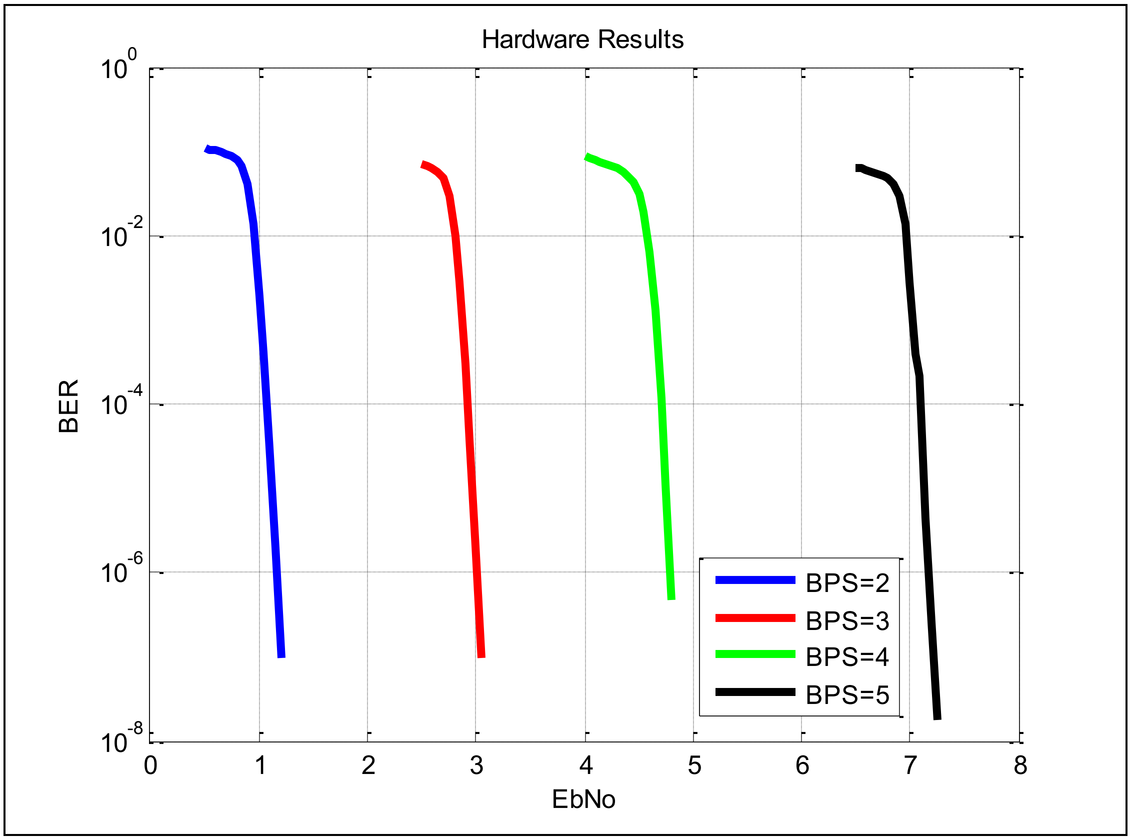


Figure 23: Hardware BER Results for QPSK, 8PSK, 16APSK, and 32APSK

3.3.3.1 Original BER Curve Performance

The original performance results are separated based on modulation and are shown below in Figures 24-27. The results for the original LLR algorithm comparisons show the MAX and True LLR Algorithms have a higher performance than the Euclidean or the 3rd party IP Algorithms. This trend increases in severity as modulation increases. However, this original testing was done with specific scaling and therefore values being sent to the decoder for one algorithm compared to another were not on the same scale. The decoder is expecting a certain range and magnitude for LLR values and might be tuned more towards expecting one of the algorithm's outputs rather than the other. Also the lack of iterations and FECFRAME length also contributed to the inaccuracy of these original test results. However the results showed that the basic algorithm implementation displayed similar results to the current LLR 3rd party IP implementation.

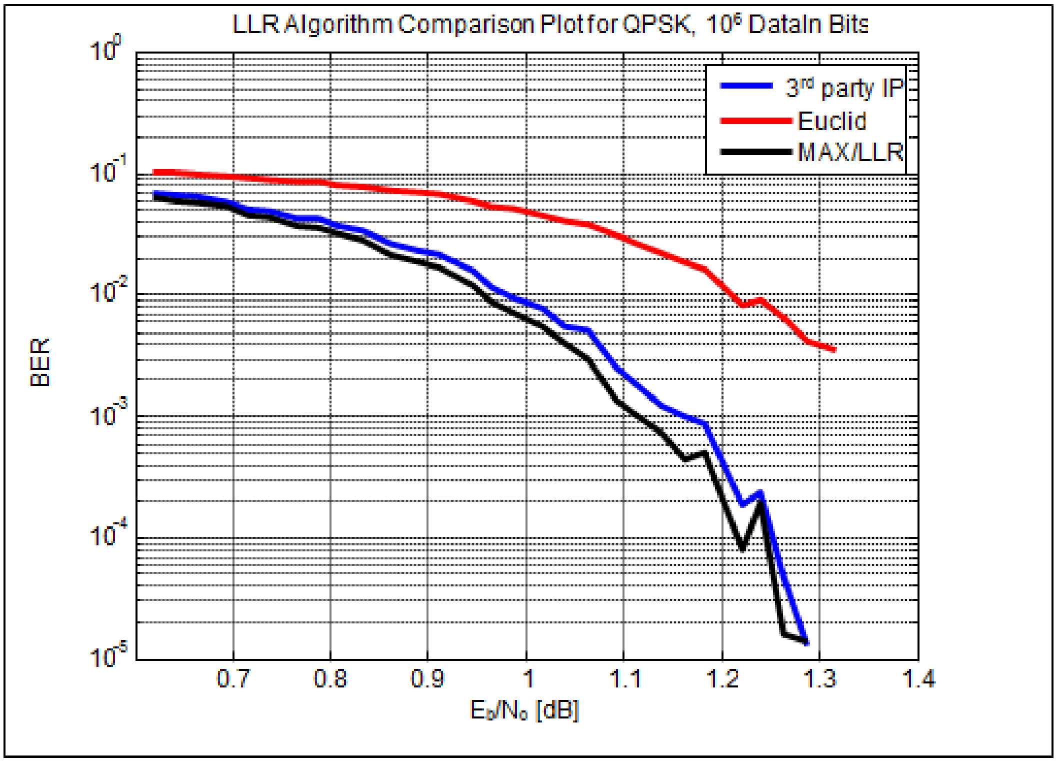


Figure 24: Original LLR Algorithm Comparison Plot for QPSK

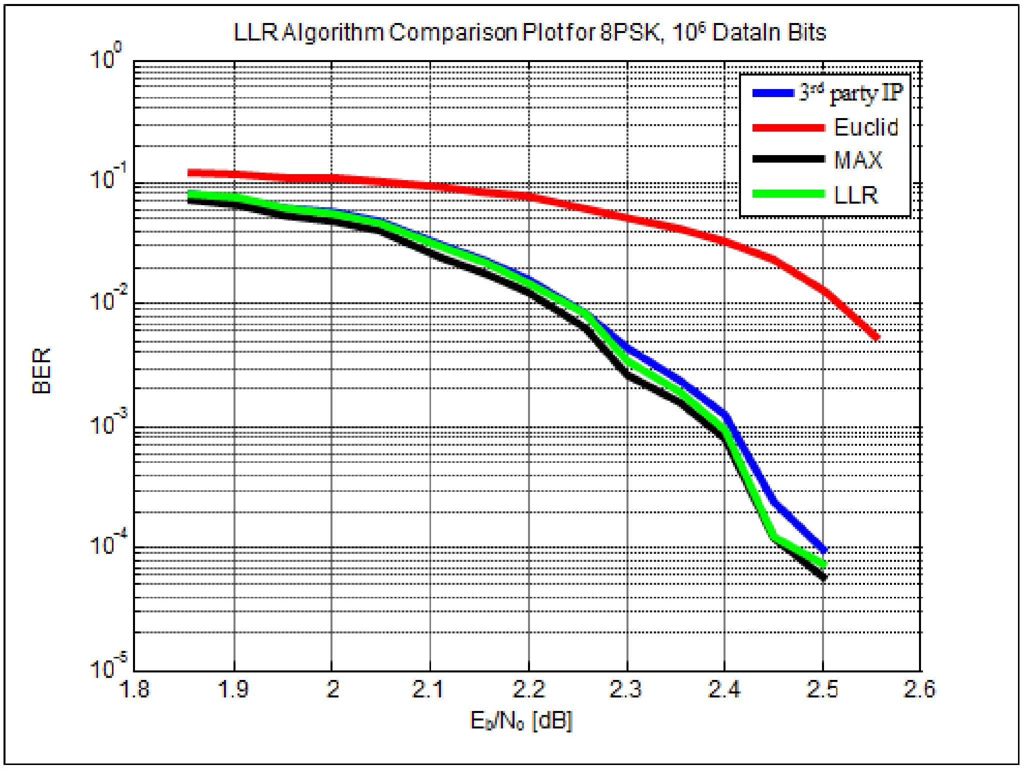


Figure 25: Original LLR Algorithm Comparison Plot for 8PSK *

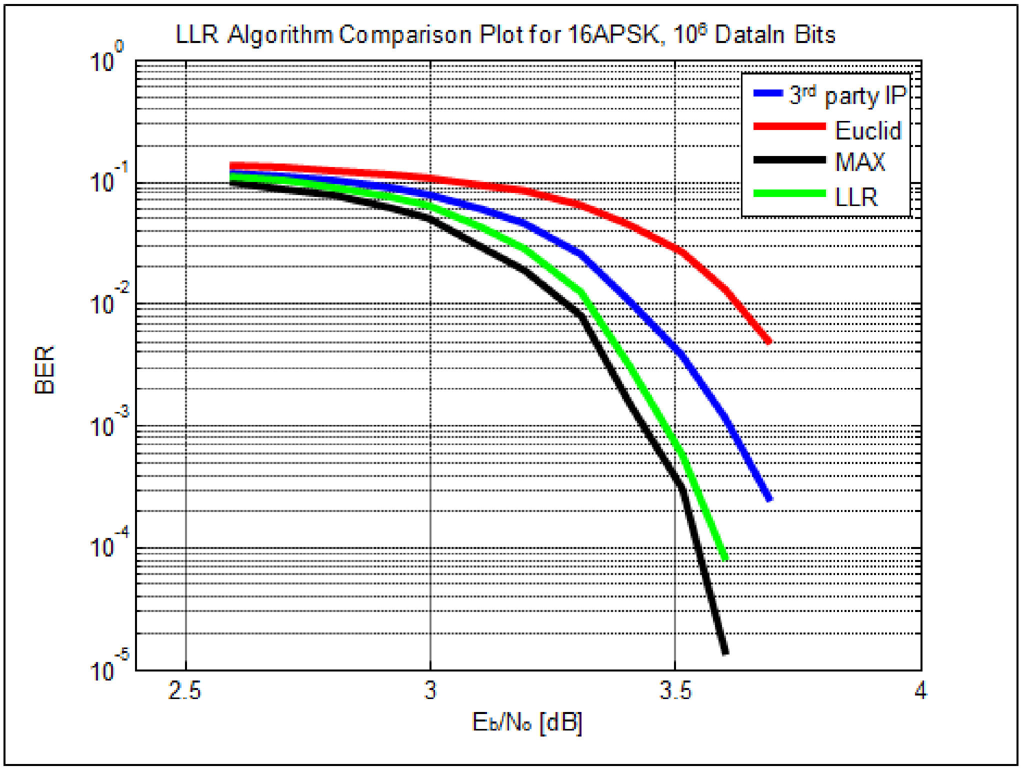


Figure 26: Original LLR Algorithm Comparison Plot for 16APSK *

* The MAX algorithm performing better than the LLR algorithm in these tests is recognized as a discrepancy that needs further investigation.

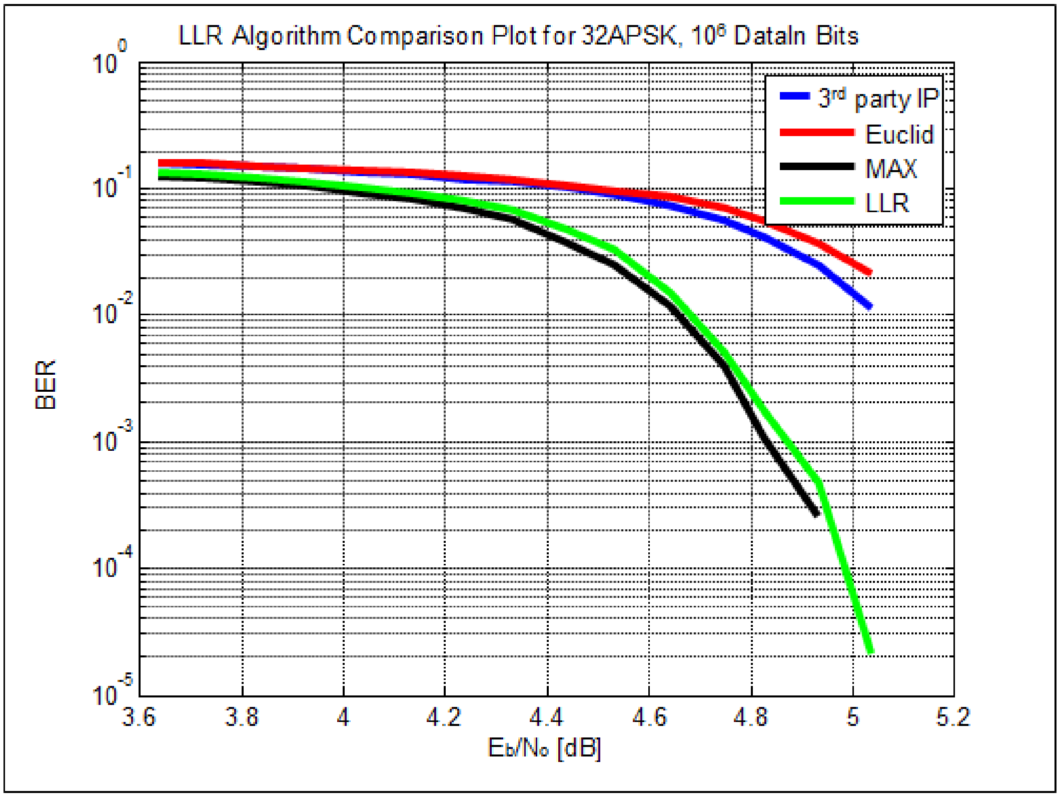


Figure 27: Original LLR Algorithm Comparison Plot for 32APSK

3.3.3.2 Updated BER Curve Performance

The original LLR algorithm comparison BER curves were repeated with an updated test plan that included the correct scaling for each algorithm at every modulation. The FECFRAME length and number of iterations were also changed to mimic the hardware results for a direct comparison. The results shown in Figures 27-31 show that the original data obtained showed the correct trend with the MAX Algorithm having the best overall BER curve performance. The updated LLR comparisons correctly illustrate the BER curve performance of all algorithms on the same scale. Unfortunately with the MATLAB simulation, limitations on the number of data input bits is approximately 10^6 where on hardware the number of bits used for performance testing is closer to 10^{11}. This produces a much smaller sample size and a loss in the number of sample points on the graph to draw the curve. The hardware results are also included for prospective and to give validity to the MATLAB simulation.

The 8PSK results are not accurate; however they are the closest approximation results that could be used. The other three modulations show correct trends where the 3rd party IP simulation results match the hardware results and the other algorithm performance results are close in comparison. The best example is shown in Figure 30 where the 3rd party IP hardware results match up directly with the 3rd party IP MATLAB simulation results. The MAX Algorithm results still have a slightly better performance which confirms the results seen on hardware when the MAX Algorithm is implemented later on hardware.

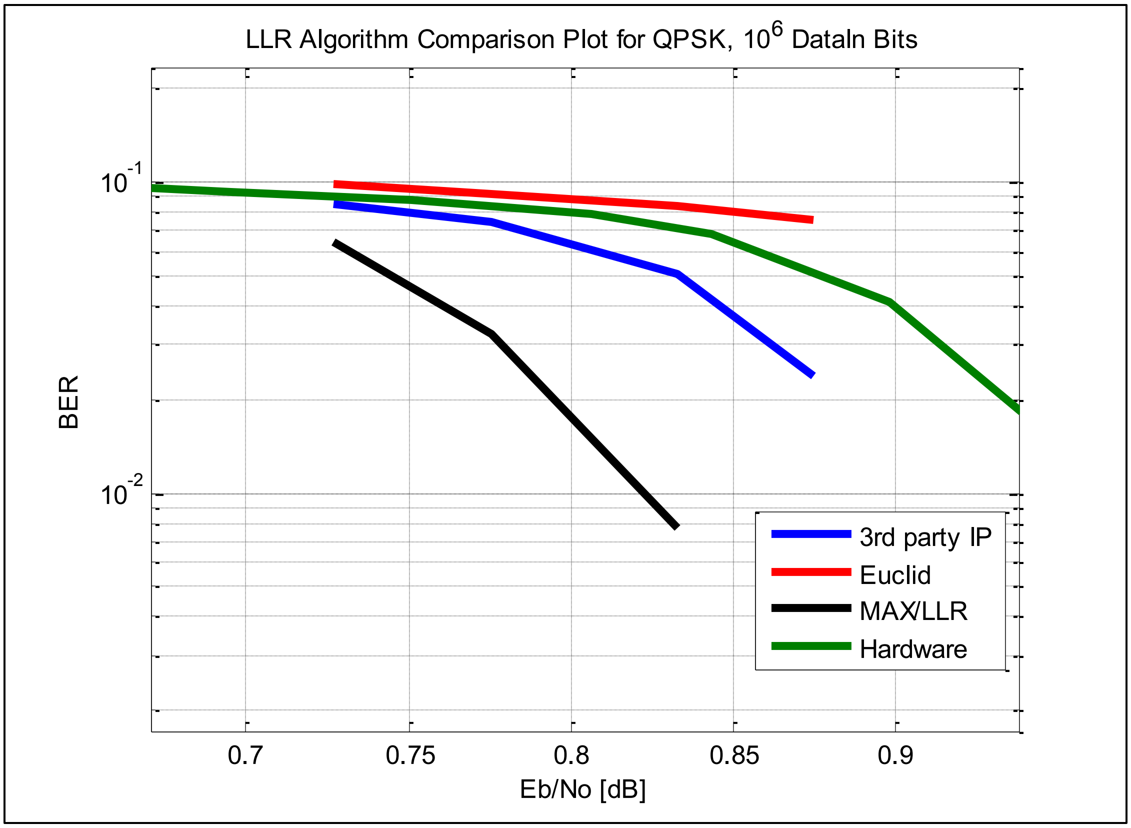


Figure 28: Updated LLR Algorithm Comparison Plot for QPSK

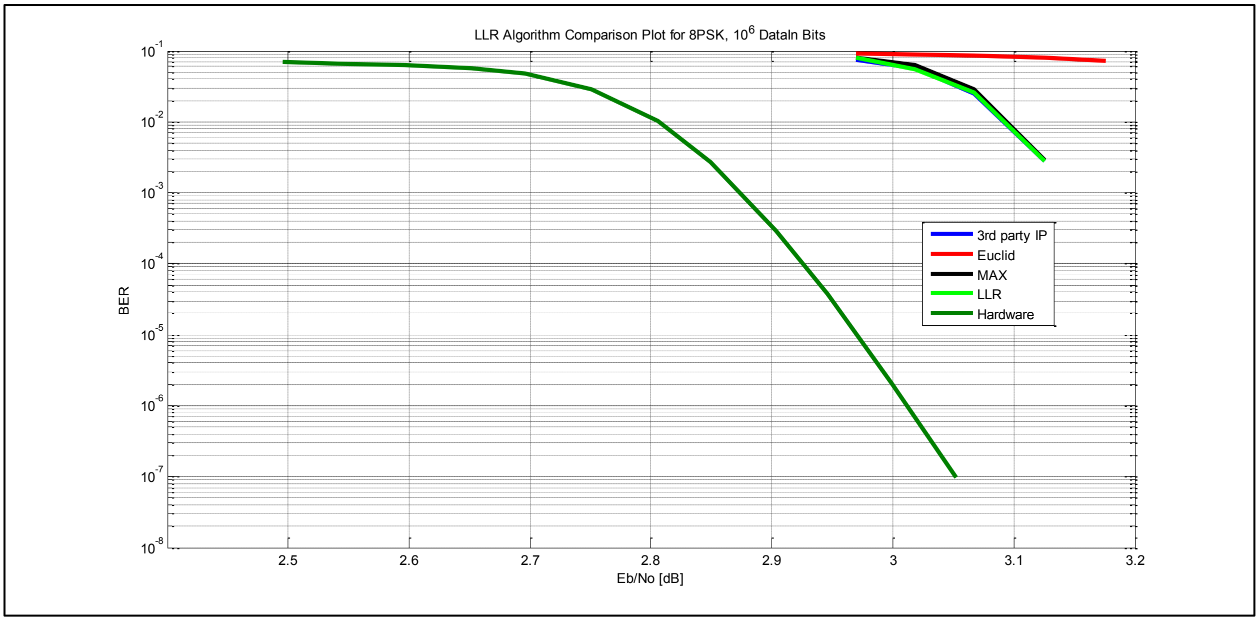


Figure 29: Updated LLR Algorithm Comparison Plot for 8PSK

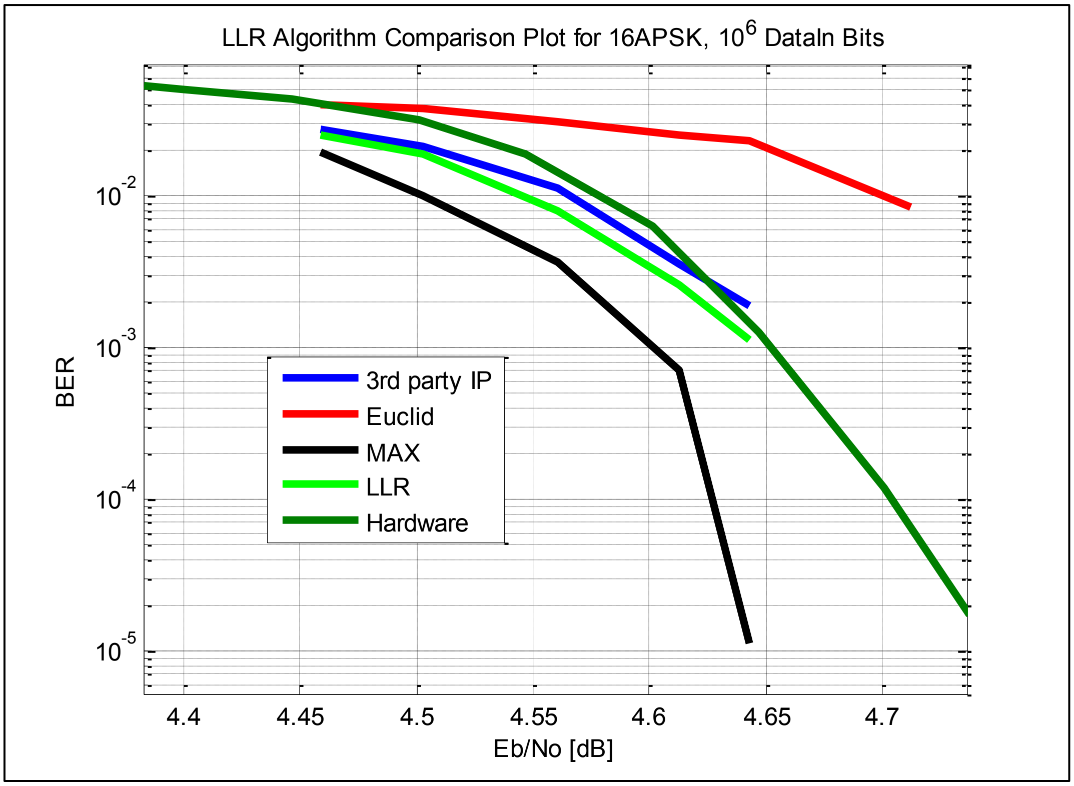


Figure 30: Updated LLR Algorithm Comparison Plot for 16APSK

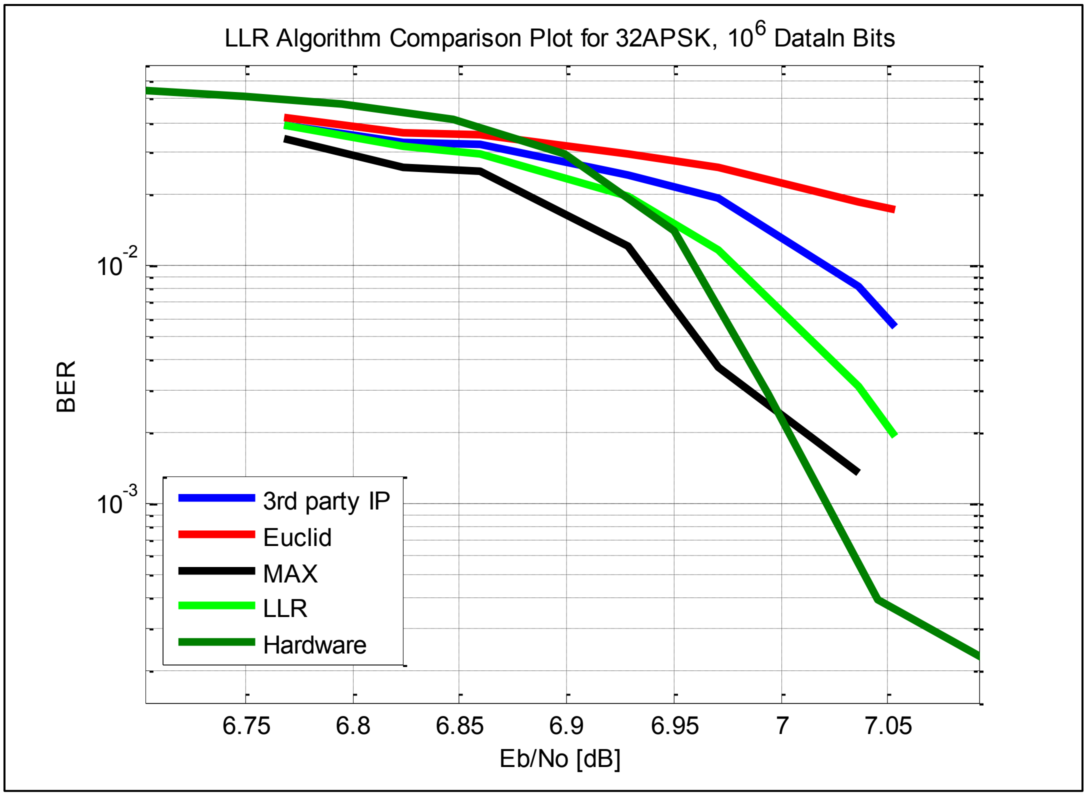


Figure 31: Updated LLR Algorithm Comparison Plot for 32APSK

4 Design and Implementation

4.1 The Chosen Algorithm

The algorithm chosen for implementation was the MAX Algorithm. The MAX Algorithm has a low complexity compared to other methods with a low number of multiplications, no exponentials or square roots, and the best BER Performance. However with the limited number of DSP resources on the FPGA, the design of the actual method could not be implemented. Instead a proposed solution included the use of LUTs. This method would allow the LLR calculations to be done in software and then loaded into LUTs. This would allow for simplification of the VHDL code on the core and allow for complex math to be done. Instead of the LLR values being calculated every time which would also create additional timing delays, the LUTs provide an efficient and expandability solution. This would also allow for any LLR algorithm to be implemented in the design because the complexity is separated for the hardware design.

4.1.1 Max Algorithm Block Diagram

The MAX Algorithm follows the block diagrams shown in Figure 32 and 33. The I/Q data are received and subtracted from the expected I/Q point “S”. The result is squared and the results added together. The result is the PDF for that point on the constellation. The block diagram shown in Figure 32 is just an example of one PDF calculation. This process would need to be repeated for the number points on the constellation which depends on the modulation scheme.

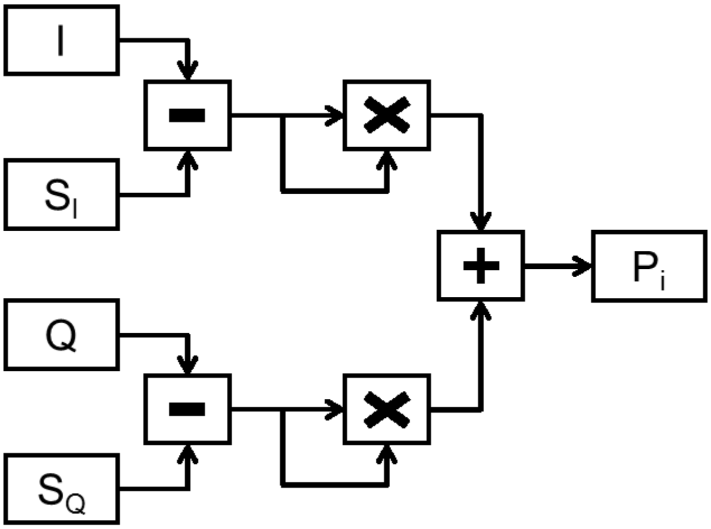


Figure 32: Block Diagram One for MAX Algorithm

The second block diagram shown in Figure 33 illustrates the process taken after the PDF's are calculated. Once a PDF is calculated it is separated into a set of points where that particular bit in the symbol should be a one or a zero on the constellation. Once all the PDF's are calculated the maximum value, which is where this algorithm's MAX name is derived from, is then used to calculate the difference between the PDF of the bit being one or zero. Normally the variance is included for each PDF calculation but because this is a constant term that is distributed it can be taken out and applied at the end. The result is the LLR value for that bit in the symbol. This process occurs for the number of bits in each symbol and for each symbol the demapper receives.

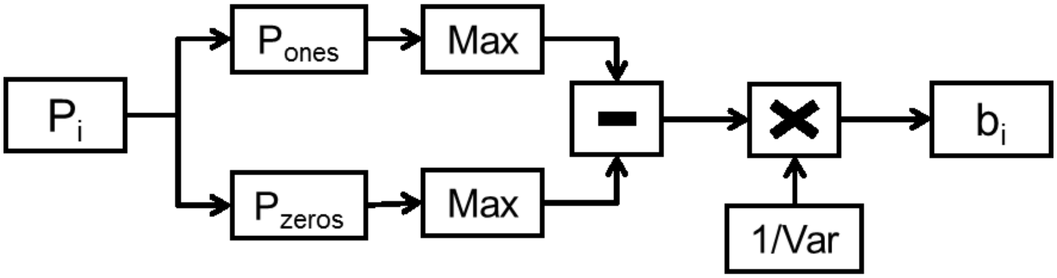


Figure 33: Block Diagram Two for MAX Algorithm

4.2 VHDL Design

4.2.1 Integration into Current LLR core Architecture

The current LLR demapper cores are located inside a wrapper that allows the cores to communicate to other components and acts as the interface between them. In order to include the new MITRE core into this architecture, several changes needed to be made to allow for the new core to work correctly with the interface.

The current MITRE core and the 3rd party IP core are currently the two LLR cores in the LLR slice. The proposed MITRE core just replaces the current MITRE core and therefore a whole new hierarchy design was not necessary. However the configuration of the current MITRE core in the LLR slice needed to be modified in order to work with the new proposed core. The proposed core requires certain data and configuration buses with different lengths than the current core. The generics and ports of the proposed MITRE core are filtered up through the demapper core hierarchy. The only major modification to the current MITRE core's entity was the addition of the LUT width size to allow for expandability and flexibility with the size of the lookup table.

4.2.2 Integration of MAX Algorithm

Originally the MAX method was going to be implemented on hardware but later in the design process. After the algorithm comparison was completed, it was decided to use a LUT for the LLR values instead of calculating them directly on hardware. The MAX Algorithm was used to generate the LLR values to store in the LUT. The LLR values are packaged and loaded into the parameter files and loaded into RAM for the core to use in the LLR calculation. The result is having all the complex mathematic calculations done in software in MATLAB and having the results stored in memory. However in order to match to the incoming received symbol with the correct LLR values, bi-linear interpolation is used to determine the actual value based on the LUT values.

4.2.3 Lookup Table

In order to save space in RAM not every single possible LLR value was calculated for all possible I/Q data points. The symbol data received is 16 bits in length where the upper 8 bits represent the quadrature value, Q, and the bottom 8 bits represent the phase value, I. Therefore the I/Q plane where the received symbol is located is on a grid that is 256x256. There would be a total of 65,536 possible I/Q values to store in the LUT. The result would be a large amount of information stored and defeat the purpose of storing the LLR values because doing the calculation would be more efficient.

Instead of storing all possible values in LUTs, a solution is to only store certain points on the I/Q plane and do bi-linear interpolation between the known points in order to figure out the LLR value at that point. By only using a certain amount of upper bits of I/Q to calculate the result is a grid where known LLR value locations are separated by the size of the bottom bits. This relationship is shown in Figure 34 with a LUT width of five. The LUT width allows the core to have flexibility by including the tradeoff between size and precision. The larger the LUT width the more space in RAM the data takes up but the more accurate the LLR results. Consequently the smaller the LUT width the less space in RAM the LUT data takes up but the LLR values will be less accurate.

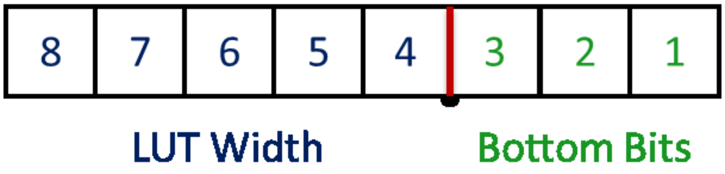


Figure 34: LUT Width Diagram

Given that the LUT has a width of five, the grid reduces from 256x256 (2^8x2^8) to 32x32 (2^5x2^5). This reduces the number of LUT values from 65,536 to 1,024. Since the LUT width is five, the three extra bits in the I/Q data are used as the remainder and used in the bi-linear interpolation to approximate the known LLR value based on the five bits. The length between known LLR points with a LUT width of five is eight. The magnitudes of the bottom bits are very important in the linear interpolation proportionality calculation. The LLR at an I/Q point will be between one and eight between the known values and therefore the bottom bits act as the scaling of the known LLR values to the LLR value at that point based on the magnitude of the bottom bits of that particular I/Q symbol.

The LLR value at a particular I/Q value changes depending on which bit in the symbol the LLR calculation is for as well as the overall modulation. Therefore there needs to be four different LUT depending on the modulation. For a given modulation the LLR, each bit in the symbol has a different LLR calculation and therefore the LUT for a specific I/Q value needs to include the LLR value based on which bit in the symbol is currently being determined. The values returned from the LUT are 6 bits in magnitude and represent the LLR for the modulation, the specific bit in the symbol, and for the I/Q value. Since each LLR value is 6 bits and the highest modulation currently implemented has a 5 bit symbol size there needs to be a total of 30 (5x6) bits stored in each LUT location.

In order for the hardware to compute the bi-linear interpolation there are four unique lookups necessary into the LUT. In order to accomplish the four simultaneous lookups there are four individual RAM allocations to increase speed and allow all four known values to be accessed at the same time rather than consecutively. This effectively results in four copies of the LUT which is why the size of the LUT has to be considered. The larger the LUT width, the move values are stored, increasing the space needed in RAM to store these LUT. However if not a large enough LUT width is used, the LLR values will not be as accurate. Simulation of the MAX Algorithm for 32APSK modulation with a 12.1 E_S/N_0 with different LUT widths is shown below in Table 10. A LUT width was chosen for the default implementation of the core because of its low BER performance which results in having high accuracy LLR calculations from the bi-linear interpolation and a relatively small amount of memory needed for the LUTs.

LUT Width	Errors	Data Input (bits)	BER
1	37,386	96,816	3.862E-1
2	14,439	96,816	1.491E-1
3	7,356	96,816	7.60E-2
4	7,422	96,816	7.67E-2
5	7,136	96,816	7.37E-2
6	7,353	96,816	7.59E-2
7	7,358	96,816	7.60E-2
8	7,341	96,816	7.58E-2

Table 10: LUT Width BER Comparison

4.2.4 Bi-linear Interpolation

Bi-linear Interpolation is linear interpolation but with two variables on two dimensions. Linear interpolation is a mathematic technique that is used to approximate a value between two points with known values. A linear interpolation assumes a constant slope between the two known points to create the curve. If the function between the two points is not linear this introduces error into the approximation.

The LLR values stored in RAM are separated by an amount specified by the LUT width and therefore creating a grid-like pattern of known LLR values across the I/Q plane. The goal is to determine the LLR value at point "P" and coordinates (I, Q) shown in Figure 35. The x-coordinate represents the phase (I) component and the y-coordinate represents the quadrature (Q) component of the received signal for the LLR calculation. There are only certain I/Q values that have known LLR values and therefore creating a grid-like or box pattern across the I/Q plane. Assuming that the I/Q point is not a known value, there are four I/Q points that form a box around that unknown point. The four red points (X_{11}, X_{12}, X_{21}, and X_{22}) in Figure 35 represent the four known LLR values at specific I/Q values relating to their sub-index. Bi-linear interpolation consists of three separate linear interpolation calculations from X_{11} to X_{21}, X_{12} to X_{22}, and R_2 to R_1. The result of this process will give the LLR approximation at point P based on the LLR four known points. The closer those known points are to the point P the more accurate the LLR value will be based on the other four values. Conversely, the farther away the four known points are from point P results in a less accurate the LLR value.

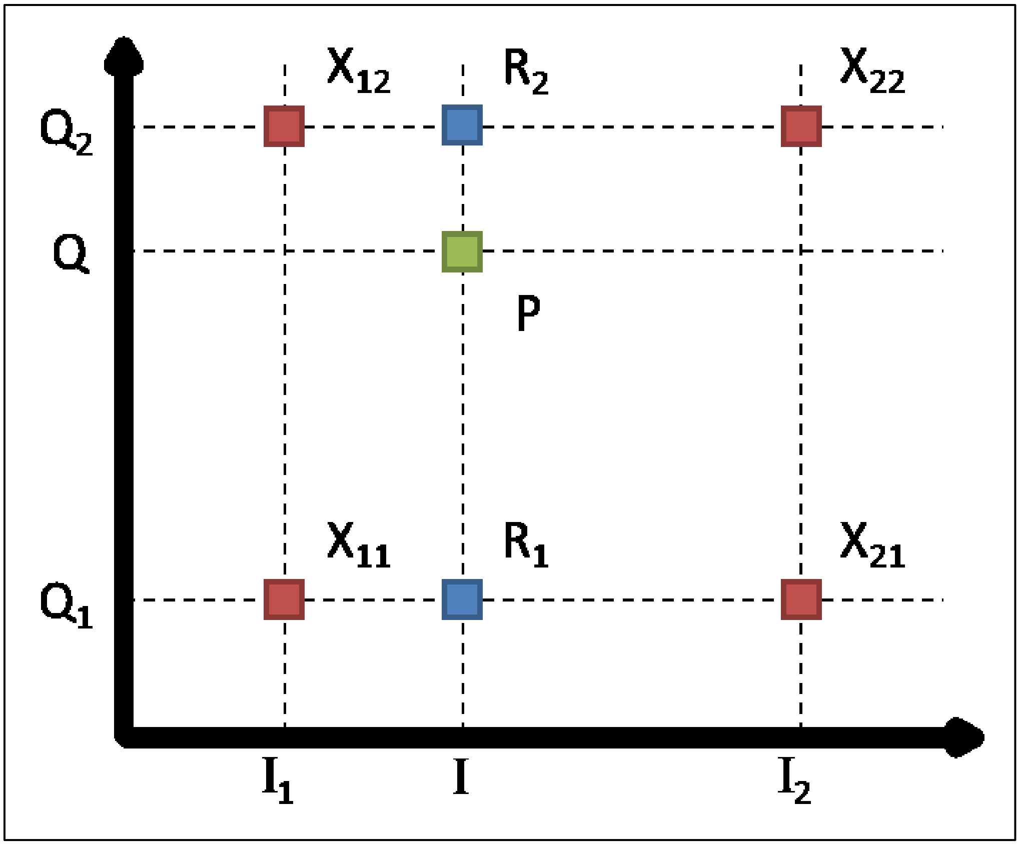


Figure 35: Bi-Linear Interpolation Diagram

The first linear interpolation in the phase direction is calculated across the bottom of the box shown in Figure 36. This linear interpolation will determine the LLR value for point R1 using proportionality based on the distances from I_1, I_2, I, and the known LLR values at X_{11} and X_{21}. This relationship is represented below in Equation (7).

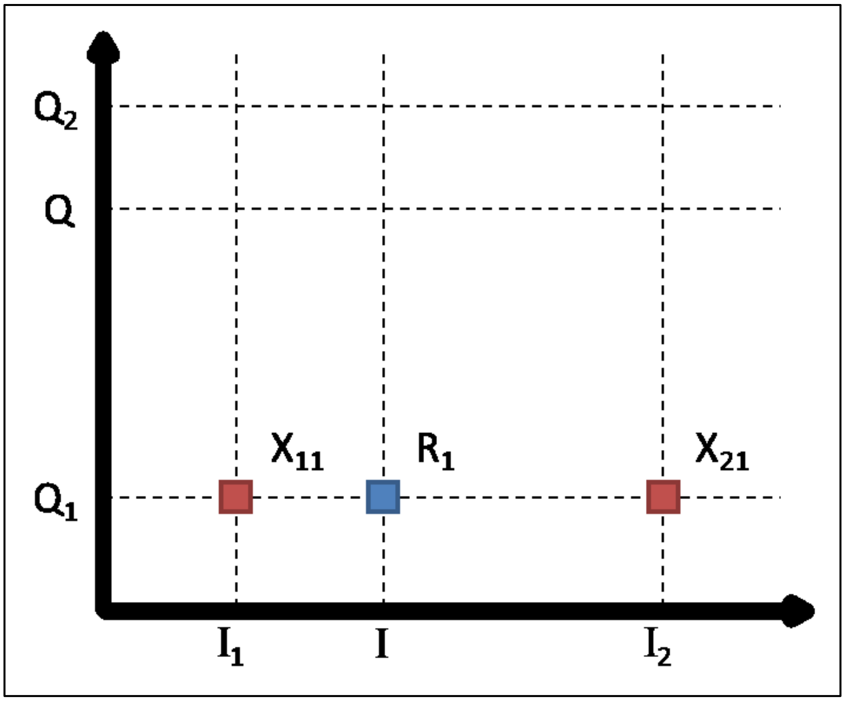


Figure 36: Linear Interpolation for Point R1

$$LLR(R_1) \approx \frac{I_2 - I}{I_2 - I_1} \cdot LLR(X_{11}) + \frac{I - I_1}{I_2 - I_1} \cdot LLR(X_{21})$$

(7)

The second linear interpolation in the phase direction is similar to the first except across the top of the box in order to determine the LLR value at point R2 shown in Figure 37. This linear interpolation will determine the LLR value for point R1 using proportionality based on the distances from I_1, I_2, I, and the known LLR values at X_{12} and X_{22}. This relationship is represented below in Equation (8).

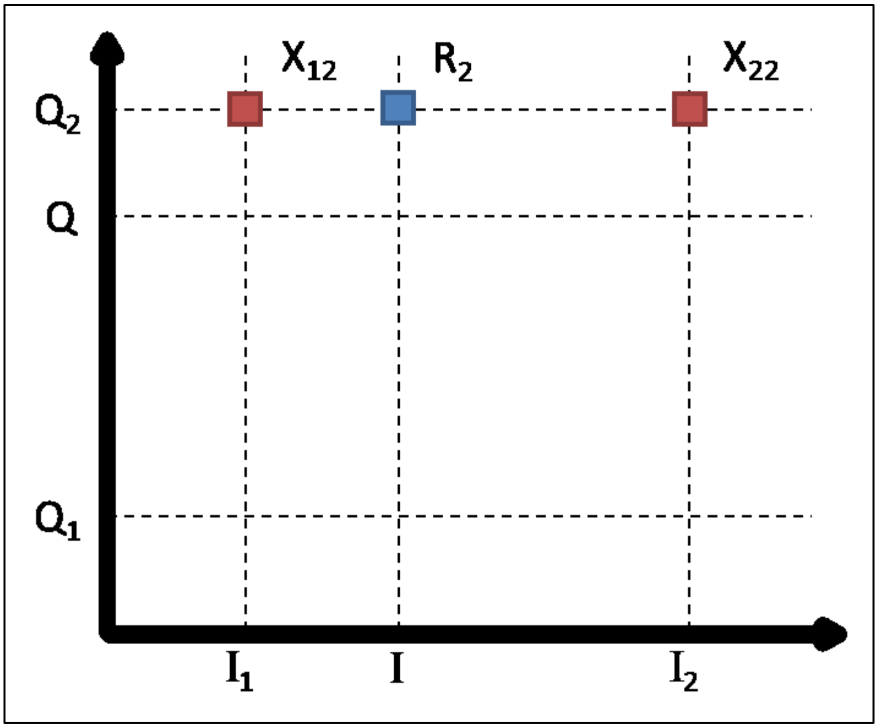


Figure 37: Linear Interpolation for Point R2

$$LLR(R_2) \approx \frac{I_2 - I}{I_2 - I_1} \cdot LLR(X_{12}) + \frac{I - I_1}{I_2 - I_1} \cdot LLR(X_{22}) \tag{8}$$

The final linear interpolation is in quadrature direction between the first two linear interpolation results shown in Figure 38. This linear interpolation will determine the LLR value for point P using proportionality based on the distances from Q_1, Q_2, Q, and the known LLR values at R_1 and R_2. This relationship is represented below in Equation 9.

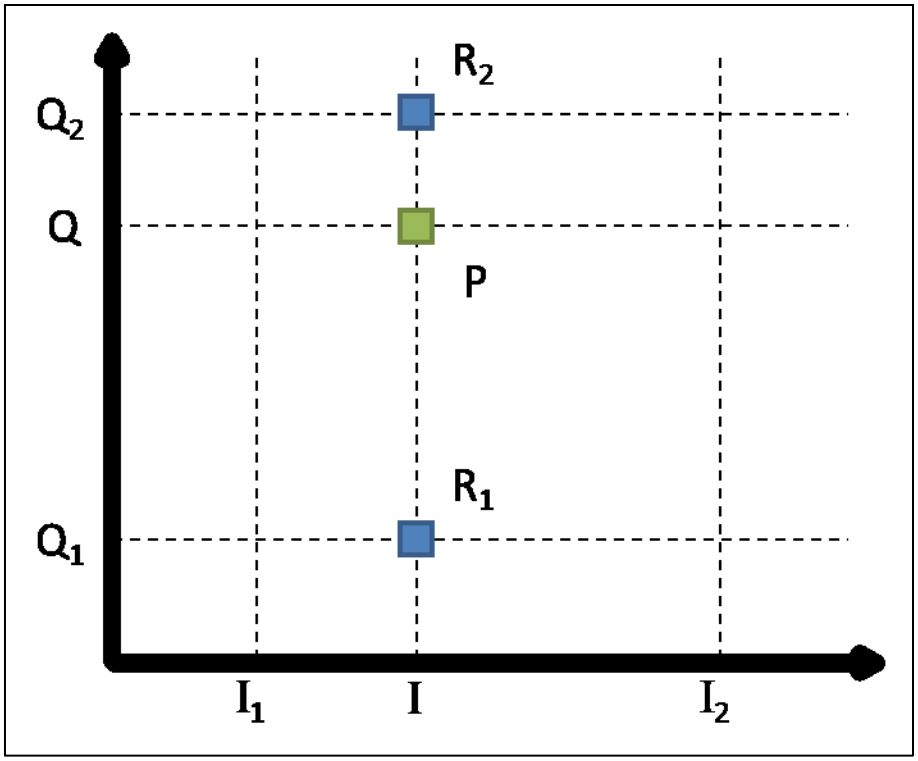


Figure 38: Linear Interpolation for Point P

$$LLR(P) \approx \frac{Q_2 - Q}{Q_2 - Q_1} \cdot LLR(R_1) + \frac{Q - Q_1}{Q_2 - Q_1} \cdot LLR(R_2) \qquad (9)$$

4.2.5 Data Flow

Data flow is extremely important in the core design because it a part of a series of receiver components that connect together. The other components such as the decoder along with the LLR core act as black boxes where the importance is put on accepting and outputting the correct parameters and to meet timing constraints. In order to correctly meet these conditions the information inside the LLR core needs to be handled correctly.

There are several scenarios where data could be lost or corrupted. A specific scenario could include the LLR core is not ready to accept new data and it receives new data, similarly if the core sends out data to a component that is not ready to accept that data. Since the core is run on a clock edge it is constantly pushing data through the pipeline when the next component is ready to accept that data. However sometimes that data is not valid and therefore the data being pushed should not be used until that data becomes valid.

In order to account for these scenarios the core uses a "handshake" between the components it directly interfaces with. This handshake, shown in Figure 39, allows the core to correctly process the data based on the two input and output "handshake" ports. The input data in valid port tells the core whether or not the incoming data is valid from the previous component. The output data out valid port displays to the next component whether or not the LLR core's data is valid. The input data out accept displays to the next component whether or not the LLR core is ready to accept new data. The output data in accept displays the next component's accept status. By correctly checking these ports the LLR core can process data correctly and be able to interact with components before and after the "handshake".

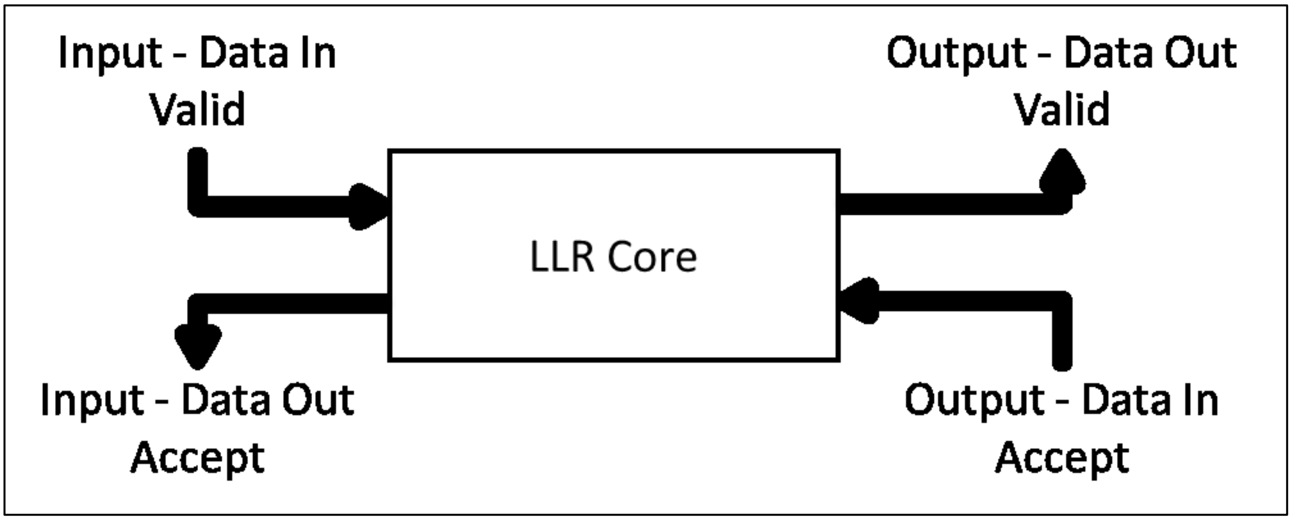


Figure 39: LLR Core Data Flow Handshake

Inside the LLR core there are two pipelines that store the status of the validity of the data and the EOB flag. The end of the block flag is related the length of the FECFRAME and although it is not needed for the LLR core it still needs to be passed on to future components. The core checks to make sure that the next component is ready to accept data by sampling the output data in accept port. If this port is high that means the LLR core can start computing the incoming data.

The core starts processing the received symbols even if the data is not valid because it keeps track of the valid data in the valid data pipeline array. The LLR core is a sequence of clocked registers where information is taken out of a register sent through logic and then stored back into a register. This relationship allows the data to be tracked as it goes through the LLR core. On every clock edge the input data in valid port is loaded into a pipeline which allows for the data and its corresponding validity to progress through the core at the same rate. The output data out valid port is connected to the last index of the data valid pipeline and therefore when the data is actually being output by the LLR core the corresponding validity is also outputted. The LLR core is coded to reset these pipelines on a standard reset instruction. Without this "handshake" the core would not be able to communicate to the other components and therefore even if the received symbols were being correctly decoded the result would not show up correctly.

4.2.6 Rounding

The original design of the LLR core's bi-linear interpolation included an implementation of division by dropping of least significant bits. The result was correct in theory because the division was correct but the implementation effectively floored the result instead of rounding.

For example of the LLR value was 22.894 which after dropping the least significant bits, the result would be 22 instead of the correct result of 23. In order to solve this, a rounding entity was added to the core to process the LLR result after the bi-linear interpolation and therefore obtain the correct rounded result.

4.2.7 Timing

The LLR core and other components run on a common clock and therefore it is important that the proposed core meets timing. The goal for the project design was the ability to run the core at 150+ MHz. After synthesis testing the core in parallel by two ran at 199MHz, which is way above the 150 MHz goal. The core was also tested in parallel by four and the result from that testing showed that the core passed for 142 MHz which is slightly below the 150 MHz goal. After further investigation the timing issues are related to the lookup time from RAM. Future work could be done to improve the maximum clock speed by looking into the register timing.

5 Testing and Verification

5.1 MATLAB Testing

The testing and verification process included software and hardware testing. MATLAB was used extensively throughout the design process specifically in generating BER performance curves for the different LLR algorithms. The BER curve performances were instrumental in understanding the core concept of the project. The MATLAB script written to run all the tests consists of a sequence of parameters the user can choose from in order to specify details such as the modulation scheme, the number of input data input bits, and which type of BER curve performance plot to execute.

5.2 C++ Model Testing

Instead of creating a C++ model for the new design the third party IP model was used instead for comparison. The proposed MITRE core VHDL was run alongside the C++ model and compared for another reference of comparison. The results from this testing produced zero errors for all four modulations when the C++ model was compared with the proposed MITRE core.

5.3 VHDL Data Flow Testing

The VDHL simulation testing was done on MITRE's servers and was done in several steps. The original test for the VHDL code included a LUT width of five, QPSK modulation, and T0. By holding these configurations constant the VHDL code could be tested and modified to add additional features while maintaining a baseline for what was known to be currently working.

There are three different test types used in this test sequence labeled as T0, T1, and T2. The three tests alter the "handshake" ports referenced in section 4.2.5 Data Flow. Test 0 (T0) configures the core to always accept data and that the data is always valid. Since this is the baseline test it was use for the majority of the modulation testing. Test 1 (T1) configures the core to toggle the input valid data port. Test 2 (T2) configures the core to toggle the output accept data port. Test 1 and 2 were used to test the data flow by testing the valid pipeline and the "handshake" control ports on the core. These three tests were tested on all test configurations used in the final test plan.

The final test plan and the corresponding results are shown in Appendix B. The tests were broken down into configuring the LLR core to run in parallel by two or four. Parallel by two and four are the two configurations that will be used for the LLR demapper. In order to properly test the expandability and flexibility of the design the LUT widths were changed to make sure the configurations changed properly. The LUT widths tested ranged from 1 to 8 in which, for a given LUT width, all four modulations were tested and for each modulation all three test types were run. This test plan shows the ability of the core to have flexibility for different LUT widths.

5.4 Hardware Results

In order to test the LLR demapper using the Digital LDPC Mini waveform, several tests were run with different modulations and FEC rates. The types of tests that were originally part of the test plan included the modulations listed below in Table 11.

LLR Demapper Modulation Tests
QPSK ½ Short
QPSK ½ Long
QPSK 7/8 Short
QPSK 7/8 Long
8PSK 7/8 Long
16APSK 7/8 Long
32APSK 7/8 Long

Table 11: LLR Demapper Modulation Test Plan

5.4.1 FPGA Slice Logic Utilization

The addition of the new 64APSK modulation core did not allow for the proposed core and the 3rd party IP core with the new configurations to be deployed at the same time. Instead, the proposed core was just deployed so some testing could be made since similar hardware results were already obtained. The slice logic utilization for the proposed MITRE core is shown in Table 12 below. The current core and the 3rd party IP 32APSK core slice logic utilization is shown in Table 13. The current core and the 3rd party IP 64APSK core slice logic utilization is shown in Table 14.

Proposed MITRE Core	
Name	**Amount**
Slice Registers	11,713
Slice LUTs	9,901
RAMB36E1	68
FIFO36E1	0
DSP48E1	136

Table 12: FPGA Slice Logic Utilization for the Proposed MITRE Core

Current MITRE Core and the 3RD party IP 32APSK Core	
Name	**Amount**
Slice Registers	25,177
Slice LUTs	43,009
RAMB36E1	28
FIFO36E1	0
RAMB18E1	132
FIFO18E1	0
DSP48E1	368

Table 13: Slice Logic Utilization for Current MITRE Core and 3rd party IP 32APSK Core

Current MITRE Core and the 3RD party IP 64APSK Core	
Name	**Amount**
Slice Registers	82,708
Slice LUTs	72,660
RAMB36E1	56
FIFO36E1	0
RAMB18E1	8
FIFO18E1	0
DSP48E1	616

Table 14: Slice Logic Utilization for Current MITRE Core and 3rd party IP 64APSK Core

5.4.2 Digital LDPC Mini Testing

After the proposed MITRE core was deployed to the LDPC Mini waveform, testing on the hardware could begin. The first step in confirming the validity of the new core on hardware included testing with no noise added to the symbols. The results indicated a 0 BER for all modulation, FEC coding rate, and FECFRAME length combinations. When symbols are transmitted with no noise added it means that they are received with the same I/Q coordinates as the expected symbol constellation points on the I/Q plane. This basic test confirms that the LLR core correctly interoperates all possible received symbols for all modulations.

The next step involved the addition of noise to the test process which revealed mixed results. Poor BER performance prompted a confirmation of the configurations used including the power level used for mapping the constellation values, the variance values determined through simulation, and the internal scaling done within the LUT generation. The mapping of the constellations originally used a power level of 90 for all modulations. However the power level of 90 is only used for 16APSK and 32APSK for the DVB-S2 standard and a lower power level is used for QPSK and 8PSK. The power level used when mapping the symbols with the lower modulations was altered to reflect the correct values provided by the DVB-S2 standard. A simulation to confirm the sampled variance values revealed that the variance values were not accurate anymore to the new design and therefore were also updated to reflect these changes. The last plausible cause of the poor BER performance involved the scaling internally to the core and the AGC scaling before and after the LLR core. The corrections made to the design improved the overall performance of the core for the QPSK modulation but not for higher modulations.

5.4.3 AGC Scaling

The AGC component scales the digital power to account for power loss. The decoder is looking for LLR values with a certain power level and the AGC is used to scale the LLR values to the right power level in order for the decoder to interpret the LLR values accurately. In order for the AGC to scale the LLR values correctly the values must be mapped to the correct power level on the constellation.

The AGC scaling before the LLR core was the focus of improving performance. A range of AGC scaling values were tested in order to determine the best scale to use that would result in the best BER performance for a given E_b/N_o. QPSK 7/8 normal AGC scaling test results shown below in Table 15 illustrate the change in BER when the AGC scaling is altered. Based on these results an AGC scale value the BER improved from originally being 9.638E-03 to 7.016E-07 E_b/N_o of 3.7 by changing the AGC scale from 7694 to 7510. The 3rd party IP core has a BER of 7.739E-07 for the same E_b/N_o which is an improvement in the BER by 7.23E-08.

Test Number	E_b/N_o	AGC Scale Value	BER
1	3.8	7500	4E-10
2	3.8	7450	0
3	3.8	7495	0
4	3.8	7499	0
5	3.8	7501	4E-10
6	3.7	7694	9.638E-03
7	3.7	7500	7.126E-07
8	3.7	7495	9.431E-07
9	3.7	7499	9.431E-07
10	3.7	7505	7.126E-07
11	3.7	7510	7.016E-07
12	3.7	7515	9.647E-03
13	3.7	7450	9.431E-07
14	3.7	7507	7.126E-07
15	3.7	7509	7.126E-07
16	3.7	7508	7.126E-07
17	3.7	7511	7.126E-07
18	3.7	7512	7.126E-07
20	3.7	7513	7.126E-07
21	3.7	7514	7.126E-07

Table 15: QPSK 7/8 Normal AGC Scaling Test Results

The best AGC scaling value was chosen based on the results collected. However the best AGC scaling value altered depending on several conditions including modulation, FEC coding rate, FECFRAME length, and current E_b/N_o. Future work could be done to improve the AGC scaling values used in these various situations in order to improve the overall performance of the LLR core.

5.4.4 Final Hardware Results

The poor BER performance limited the ability to obtain the results for the higher order modulations. The AGC scaling that was implemented to QPSK to improve performance did not affect the higher order modulations which suggest that there are other scaling issues. The limited time of the project did not allow for additional investigation of the performance issues. As a result only the QPSK modulation results were obtained and are shown below in Figures 41-44. The current MITRE core (Old core) is compared with the proposed MITRE core (New core) for the QPSK modulation with varying FEC coding rates and FECFRAME lengths. The current MITRE core's hardware results for all modulations are shown in Figure 40 for reference.

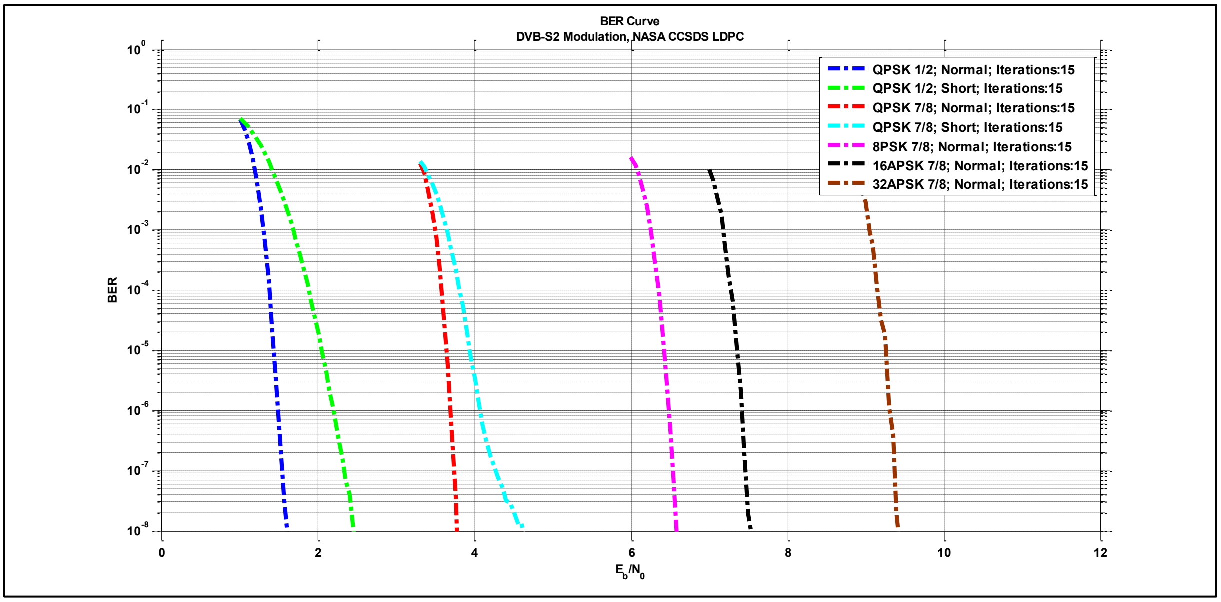


Figure 40: 3rd party IP Hardware Results for all Modes and Modulations

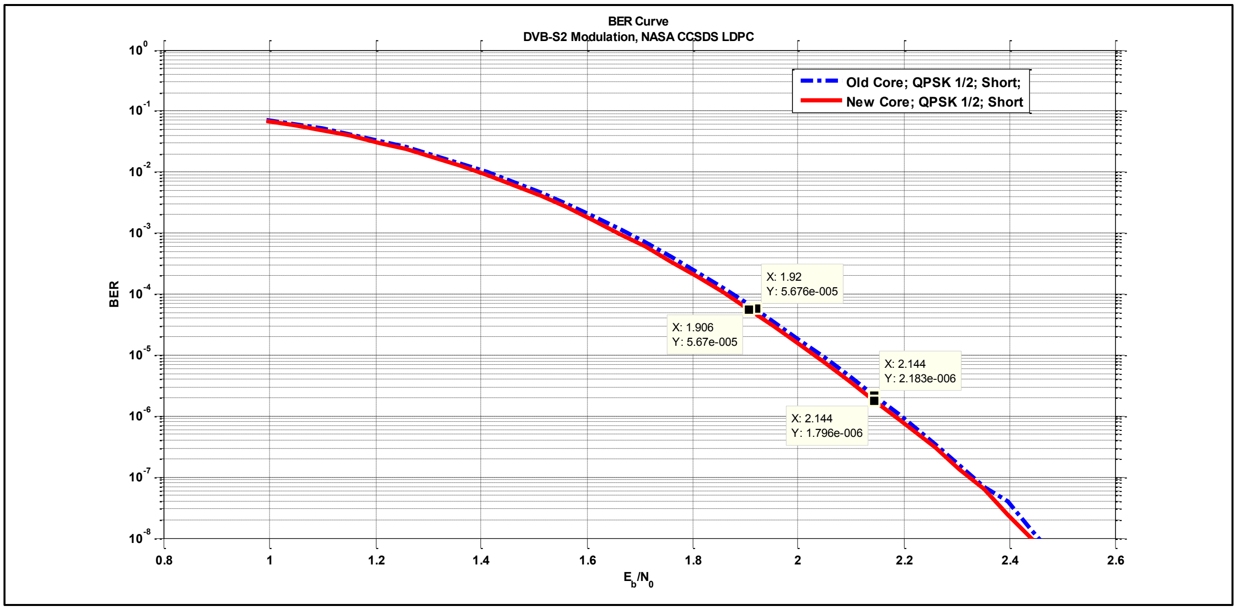


Figure 41: QPSK 1/2 Short BER Performance Comparison

Exp_E_b/N_o	Ns_gain	Din_gain	Act_E_b/N_o	BER	Errors	Samples
1.00	3261	325	0.993702	6.95E-02	100000000	1438868264
1.05	3261	327	1.04699	5.86E-02	100000001	1706219160
1.10	3261	329	1.099952	4.85E-02	100000001	2061855568
1.15	3261	331	1.152594	3.93E-02	100000000	2541889728
1.20	3261	333	1.204919	3.08E-02	100000000	3252402456
1.25	3261	335	1.256931	2.35E-02	100000000	4248589704
1.30	3261	337	1.308632	1.75E-02	100000000	5717475216
1.35	3261	339	1.360028	1.27E-02	100000003	7848669680
1.40	3261	341	1.411122	9.00E-03	100000001	11115407024
1.45	3261	343	1.461917	6.09E-03	100000000	16433180088
1.50	3261	345	1.512416	4.05E-03	100000000	24713839288
1.55	3261	347	1.562624	2.62E-03	100000002	38162780336
1.60	3261	349	1.612543	1.62E-03	100000000	61846201552
1.65	3261	351	1.662177	9.85E-04	98510677	1E+11
1.70	3261	353	1.711529	5.98E-04	59838512	1E+11
1.75	3261	355	1.760602	3.38E-04	33833407	1E+11
1.80	3261	357	1.809399	1.92E-04	19168760	1E+11
1.85	3261	359	1.857923	1.08E-04	10750710	1E+11
1.90	3261	361	1.906179	5.67E-05	5669698	1E+11
1.95	3261	363	1.954167	2.94E-05	2935200	1E+11
2.00	3261	365	2.001892	1.52E-05	1523931	1E+11
2.05	3261	367	2.049356	7.71E-06	770686	1E+11
2.10	3261	369	2.096562	3.76E-06	376346	1E+11

2.15	3261	371	2.143513	1.80E-06	179589	1E+11
2.20	3261	373	2.190211	8.66E-07	86569	1E+11
2.25	3261	376	2.259791	3.05E-07	30521	1E+11
2.30	3261	378	2.30587	1.32E-07	13231	1E+11
2.35	3261	380	2.351706	6.38E-08	6380	1E+11
2.40	3261	382	2.397302	2.39E-08	2392	1E+11
2.45	3261	384	2.442659	9.75E-09	975	1E+11
2.50	3261	387	2.510254	1.67E-09	167	1E+11
2.55	3261	389	2.555026	1.23E-09	123	1E+11
2.60	3261	391	2.59957	5.70E-10	57	1E+11

Table 16: QPSK 1/2 Short Full Hardware Results

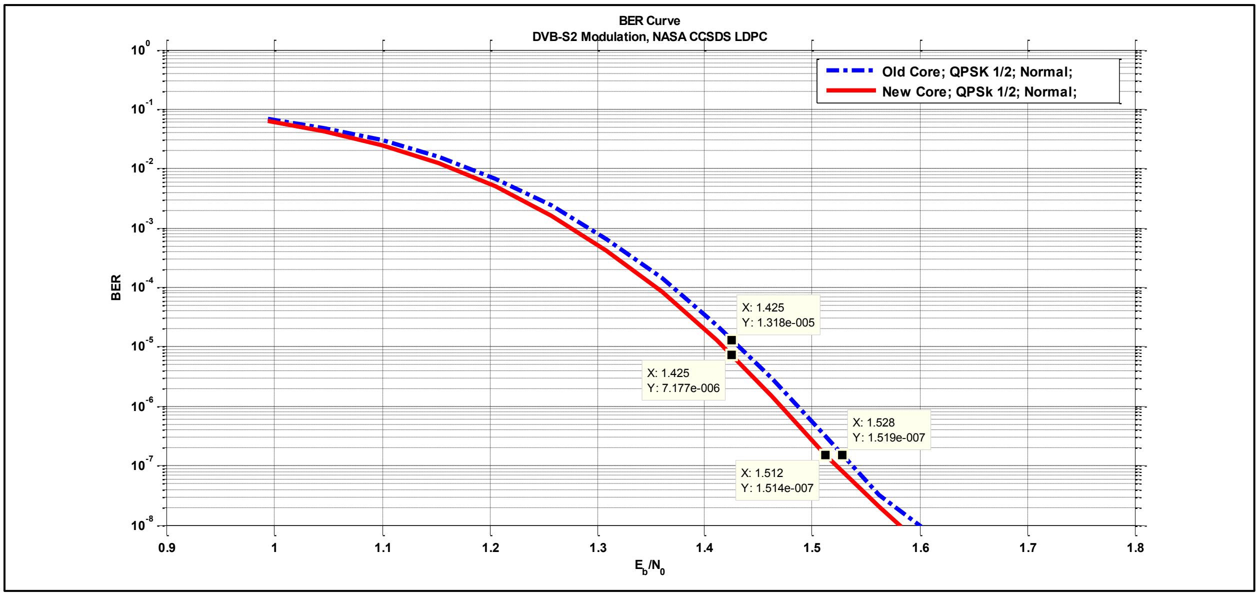


Figure 42: QPSK 1/2 Normal BER Performance Comparison

Exp_E_b/N_o	Ns_gain	Din_gain	Act_E_b/N_o	BER	Errors	Samples
1.00	3261	325	0.993702	6.26E-02	1E+08	1.6E+09
1.05	3261	327	1.04699	4.24E-02	1E+08	2.36E+09
1.10	3261	329	1.099952	2.51E-02	1E+08	3.99E+09
1.15	3261	331	1.152594	1.26E-02	1E+08	7.96E+09
1.20	3261	333	1.204919	5.00E-03	1E+08	2E+10
1.25	3261	335	1.256931	1.62E-03	1E+08	6.17E+10
1.30	3261	337	1.308632	4.11E-04	41065329	1E+11
1.35	3261	339	1.360028	8.37E-05	8366183	1E+11
1.40	3261	341	1.411122	1.32E-05	1316979	1E+11
1.45	3261	343	1.461917	1.50E-06	149469	1E+11

1.50	3261	345	1.512416	1.51E-07	15139	1E+11
1.55	3261	347	1.562624	2.08E-08	2081	1E+11
1.60	3261	349	1.612543	3.15E-09	315	1E+11
1.65	3261	351	1.662177	1.80E-10	18	1E+11

Table 17: QPSK 1/2 Normal Full Hardware Results

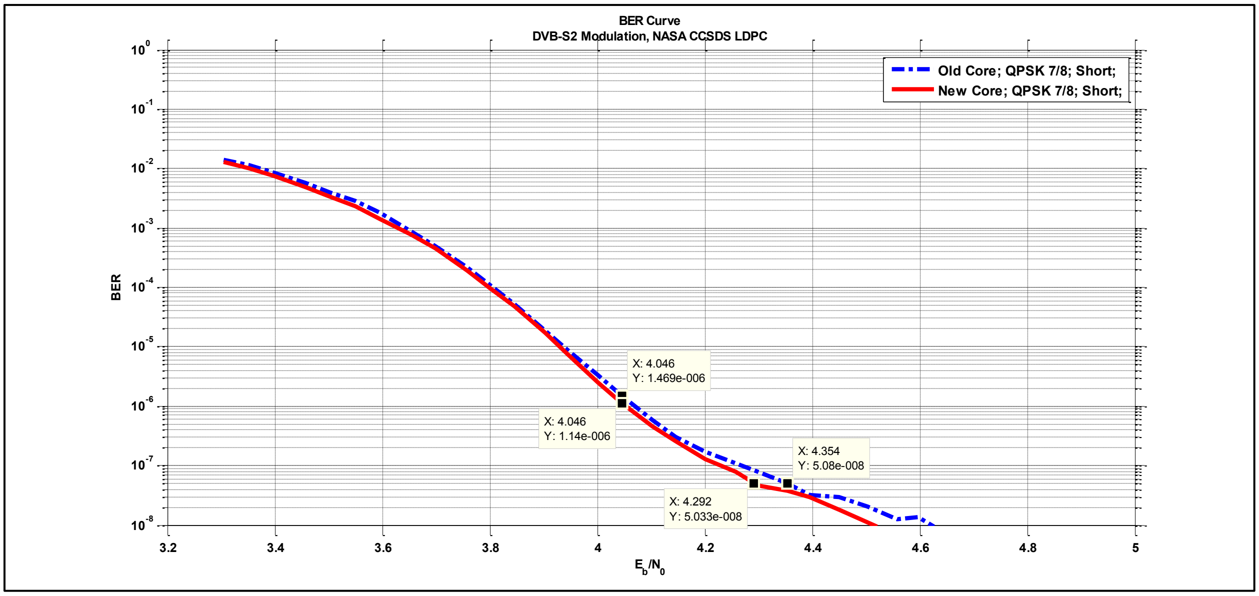


Figure 43: QPSK 7/8 Short BER Performance Comparison

Exp_E$_b$/N$_o$	Ns_gain	Din_gain	Act_E$_b$/N$_o$	BER	Errors	Samples
3.30	3261	561	3.304911	1.30E-02	100000001	7717147624
3.35	3261	564	3.351236	1.03E-02	100000000	9702544608
3.40	3261	567	3.397315	7.58E-03	100000000	13190359184
3.45	3261	571	3.458376	4.91E-03	100000000	20387019864
3.50	3261	574	3.503892	3.35E-03	100000000	29853038080
3.55	3261	577	3.54917	2.31E-03	100000000	43363082904
3.60	3261	580	3.594214	1.43E-03	100000000	69770168216
3.65	3261	584	3.653911	7.60E-04	75986434	1E+11
3.70	3261	587	3.698416	4.58E-04	45818487	1E+11

3.75	3261	591	3.757404	1.94E-04	19427144	1E+11
3.80	3261	594	3.801383	9.63E-05	9634207	1E+11
3.85	3261	597	3.845141	4.78E-05	4781245	1E+11
3.90	3261	601	3.903143	1.71E-05	1709055	1E+11
3.95	3261	604	3.946393	7.42E-06	742315	1E+11
4.00	3261	608	4.003726	2.48E-06	248085	1E+11
4.05	3261	611	4.046478	1.14E-06	113977	1E+11
4.10	3261	615	4.103156	4.52E-07	45243	1E+11
4.15	3261	618	4.145423	2.59E-07	25902	1E+11
4.20	3261	622	4.201462	1.28E-07	12834	1E+11
4.25	3261	626	4.257141	7.91E-08	7912	1E+11
4.30	3261	629	4.298667	4.59E-08	4587	1E+11
4.35	3261	633	4.353728	3.72E-08	3717	1E+11
4.40	3261	636	4.394796	2.90E-08	2898	1E+11
4.45	3261	640	4.449253	1.81E-08	1805	1E+11
4.50	3261	644	4.503371	1.12E-08	1121	1E+11
4.55	3261	648	4.557154	7.06E-09	706	1E+11
4.60	3261	651	4.597274	6.15E-09	615	1E+11
4.65	3261	655	4.65048	6.12E-09	612	1E+11
4.70	3261	659	4.703362	2.36E-09	236	1E+11
4.75	3261	663	4.755925	2.35E-09	235	1E+11
4.80	3261	666	4.795139	3.42E-09	342	1E+11
4.85	3261	670	4.84715	0.00E+00	0	1E+11

Table 18: QPSK 7/8 Short Full Hardware Results

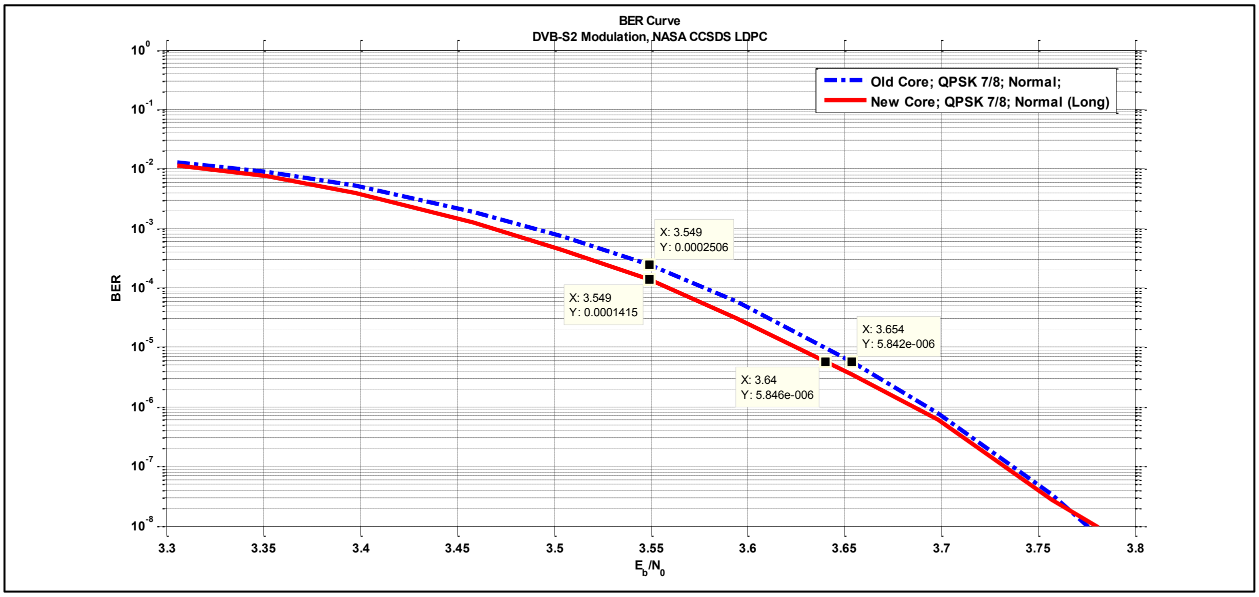


Figure 44: QPSK 7/8 Normal BER Performance Comparison

Exp_E_b/N_o	Ns_gain	Din_gain	Act_E_b/N_o	BER	Errors	Samples
3.30	3261	561	3.304911	1.15E-02	100000000	8682549536
3.35	3261	564	3.351236	7.67E-03	100000000	13031312616
3.40	3261	567	3.397315	3.89E-03	100000000	25733664176
3.45	3261	571	3.458376	1.28E-03	100000000	78255690520
3.50	3261	574	3.503892	4.35E-04	43537054	1E+11
3.55	3261	577	3.54917	1.42E-04	14150347	1E+11
3.60	3261	580	3.594214	3.03E-05	3032942	1E+11
3.65	3261	584	3.653911	3.55E-06	354832	1E+11
3.70	3261	587	3.698416	6.09E-07	60903	1E+11
3.75	3261	591	3.757404	2.69E-08	2689	1E+11
3.80	3261	594	3.801383	4.05E-09	405	1E+11
3.85	3261	597	3.845141	2.00E-11	2	1E+11

Table 19: QPSK 7/8 Normal Full Hardware Results

6 Conclusion

This project set out to investigate, design, and implement a LLR soft decision demapper for a HDR modem to replace the current core used by MITRE's HDR design group. An emphasis was placed on expandability and flexibility of the design to allow for future implementation. The major goal of the project is to improve the performance of current system and provide recommendations for future designs of the soft decision demapper for DVB-S2. The investigation of the algorithms for DVB-S2 soft-decision demapping allowed for a correct implementation of the algorithms in MATLAB to analyze the performance. Finally an improved core design was implemented in VHDL, deployed on an FPGA, and compared against the current core implementation.

This project provides MITRE with a library of MATLAB scripts and functions for LLR Algorithm implementation with the DVB-S2 standard. This directory includes algorithm simulation, BER performance, LUT parameter file generation, test vector creation scripts and all the documentation in order to repeat the project design. The directory of scripts can be found in Appendix D. The new MITRE core found in Appendix E is also included in the project contributions along with the performance results.

The proposed MITRE LLR core is planned to be integrated into the LLR slice after the conclusion of this project. The addition of the 64APSK modulation will also be included in the future along with future higher order modulations. Additional work needs to be done to obtain the 8PSK, 16APSK, and 32APSK hardware results for comparison to verify the conclusion that the proposed core meets the goal of improved performance and the addition of higher order modulation. The comparison between the MAX Algorithm LUTs and the True LLR Algorithm could be made to verify the simulation results. This would also prove the algorithm flexibility the LUT design has compared to the straight algorithm implantation. Additional performance work could be done to improve the BER performance of each modulation, FEC coding rate, and E_b/N_o values.

In conclusion, the project was able to meet the requirements and specifications of the MQP, the DVB-S2 standard, and The MITRE corporation.

References

[1] D. J. Bem, T. W. Wieckowski, and R. J. Zielinski, "Broadband satellite systems," *Communications Surveys & Tutorials, IEEE,* vol. 3, pp. 2-15, 2000.

[2] P. Jang Woong, S. Myung Hoon, K. Pan Soo, and C. Dae-Ig, "Multi-level modulation soft-decision demapper for DVB-S2," in *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, 2009, pp. 013-017.

[3] R. Aggarwal and P. Moore, "Digital communications for protection. I. General principles," *Power Engineering Journal,* vol. 7, pp. 281-287, 1993.

[4] C. J. Birkmaier, "An open architecture for digital communication systems.2. Creating enabling standards for a digital communication infrastructure," *MultiMedia, IEEE,* vol. 1, pp. 79-84, 1994.

[5] B. Sklar, "Defining, designing, and evaluating digital communication systems," *Communications Magazine, IEEE,* vol. 31, pp. 91-101, 1993.

[6] G. Kolumban, M. P. Kennedy, and L. O. Chua, "The role of synchronization in digital communications using chaos. I . Fundamentals of digital communications," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on,* vol. 44, pp. 927-936, 1997.

[7] A. B. Salberg and A. Hanssen, "Secure digital communications by means of stochastic process shift keying," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, 1999, pp. 1523-1527 vol.2.

[8] E. Casini, R. D. Gaudenzi, and A. Ginesi, "DVB-S2 modem algorithms design and performance over typical satellite channels," *International Journal of Satellite Communications and Networking,* vol. 22, pp. 281-318, 2004.

[9] R. L. Nkumbwa, "Emerging next generation communication technology: Unveiling the Ubiquitous Society," in *Education and Management Technology (ICEMT), 2010 International Conference on*, 2010, pp. 1-5.

[10] K. M. Price and R. G. Leamon, "Definition of a commercial mobile satellite services network to meet DoD communications needs," in *Military Communications Conference,*

1993. MILCOM '93. Conference record. Communications on the Move., IEEE, 1993, pp. 821-825 vol.3.

[11] E. Lutz, H. Bischl, H. Ernst, D. Giggenbach, M. Holzbock, A. Jahn*, et al.*, "Development and future applications of satellite communications," in *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, 2004, pp. 2342-2346 Vol.4.

[12] T. E. Mangir, "The future of public satellite communications," in *Aerospace Applications Conference, 1995. Proceedings., 1995 IEEE*, 1995, pp. 393-410 vol.1.

[13] A. Morello and U. Reimers, "DVB-S2, the second generation standard for satellite broadcasting and unicasting," *International Journal of Satellite Communications and Networking,* vol. 22, pp. 249-268, 2004.

[14] E. T. S. Institute, "ETSI EN 302 307 V1.2.1 European Standard (Telecommunications series)," in *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2)*, ed, 2009.

[15] C. Seung-Hyun, O. Cheon-In, O. Doeck Gil, and C. Dae-Ig, "The Mapping and demmaping algorithms for high order modulation of DVB-S2 systems," in *Communications, 2006. APCC '06. Asia-Pacific Conference on*, 2006, pp. 1-5.

[16] T. Ming-Fong, S. Ce-Kuen, H. Tzu-Chi, and D. Der-Jiunn, "Forward-Looking Forward Error Correction Mechanism for Video Streaming Over Wireless Networks," *Systems Journal, IEEE,* vol. 5, pp. 460-473, 2011.

Appendix A

Appendix A.1 – QPSK Modulation (2 BPS) – 10^5 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
```

```
Encoding 14 Blocks of 16k at rate ½
# BPS = 2   #
```

```
Es/No = 0
Euclid   = 11061 (errors) | 98448 (bits)
MAX      =  7185 (errors) | 98448 (bits)
llr2     =  7185 (errors) | 98448 (bits)
IP_llr   =  7544 (errors) | 98448 (bits)
```

```
Es/No = 5.000000e-02
Euclid   = 10246 (errors) | 98448 (bits)
MAX      =  6234 (errors) | 98448 (bits)
llr2     =  6234 (errors) | 98448 (bits)
IP_llr   =  6453 (errors) | 98448 (bits)
```

```
Es/No = 1.000000e-01
Euclid   = 9635 (errors) | 98448 (bits)
MAX      = 4705 (errors) | 98448 (bits)
llr2     = 4705 (errors) | 98448 (bits)
IP_llr   = 5286 (errors) | 98448 (bits)
```

```
Es/No = 1.500000e-01
Euclid   = 8606 (errors) | 98448 (bits)
MAX      = 3257 (errors) | 98448 (bits)
llr2     = 3257 (errors) | 98448 (bits)
IP_llr   = 3867 (errors) | 98448 (bits)
```

```
Es/No = 2.000000e-01
Euclid   = 8093 (errors) | 98448 (bits)
MAX      = 2907 (errors) | 98448 (bits)
llr2     = 2907 (errors) | 98448 (bits)
IP_llr   = 3485 (errors) | 98448 (bits)
```

```
Es/No = 2.500000e-01
Euclid   = 6866 (errors) | 98448 (bits)
MAX      = 1576 (errors) | 98448 (bits)
llr2     = 1576 (errors) | 98448 (bits)
IP_llr   = 1821 (errors) | 98448 (bits)
```

```
Es/No = 3.000000e-01
Euclid   = 6344 (errors) | 98448 (bits)
MAX      = 1516 (errors) | 98448 (bits)
llr2     = 1516 (errors) | 98448 (bits)
```

```
IP_llr    = 2054 (errors) | 98448 (bits)
------------------------------------------
Es/No = 3.500000e-01
Euclid   = 5669 (errors) | 98448 (bits)
MAX      = 1104 (errors) | 98448 (bits)
llr2     = 1104 (errors) | 98448 (bits)
IP_llr   = 1555 (errors) | 98448 (bits)
------------------------------------------
Es/No = 4.000000e-01
Euclid   = 4825 (errors) | 98448 (bits)
MAX      =  440 (errors) | 98448 (bits)
llr2     =  440 (errors) | 98448 (bits)
IP_llr   =  626 (errors) | 98448 (bits)
------------------------------------------
Es/No = 4.500000e-01
Euclid   = 2390 (errors) | 98448 (bits)
MAX      =  159 (errors) | 98448 (bits)
llr2     =  159 (errors) | 98448 (bits)
IP_llr   =  302 (errors) | 98448 (bits)
------------------------------------------
Es/No = 5.000000e-01
Euclid   = 2795 (errors) | 98448 (bits)
MAX      =  105 (errors) | 98448 (bits)
llr2     =  105 (errors) | 98448 (bits)
IP_llr   =  166 (errors) | 98448 (bits)
------------------------------------------
Es/No = 5.500000e-01
Euclid   = 2332 (errors) | 98448 (bits)
MAX      =  117 (errors) | 98448 (bits)
llr2     =  117 (errors) | 98448 (bits)
IP_llr   =  144 (errors) | 98448 (bits)
------------------------------------------
Es/No = 6.000000e-01
Euclid   = 1464 (errors) | 98448 (bits)
MAX      =    0 (errors) | 98448 (bits)
llr2     =    0 (errors) | 98448 (bits)
IP_llr   =    0 (errors) | 98448 (bits)
```

Appendix A.2 – QPSK Modulation (2 BPS) – 10^6 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:"16200":The coded size (read-only)
   usize:UNSIGNED:"7032":The input size (read-only)
   enc_config:STRING:"dvb_12_16.encbin":Encoder configuration file
   interleave_depth:INT:"1":Encode Interleave Depth
   backwards:INT:"0":Backwards symbol interleave (for 8psk 3/5 S2 code)
------------------------------------------
Encoding 142 Blocks of 16k at rate ½
# BPS = 2 #
==========================================
Es/No = 0
Euclid   = 104733 (errors) | 998544 (bits)
MAX      =  64433 (errors) | 998544 (bits)
llr2     =  64433 (errors) | 998544 (bits)
```

```
IP_llr    =   68706 (errors) | 998544 (bits)
Es/No = 5.000000e-02
Euclid   = 99396 (errors) | 998544 (bits)
MAX      = 56708 (errors) | 998544 (bits)
llr2     = 56708 (errors) | 998544 (bits)
IP_llr   = 62266 (errors) | 998544 (bits)
Es/No = 1.000000e-01
Euclid   = 94388 (errors) | 998544 (bits)
MAX      = 47455 (errors) | 998544 (bits)
llr2     = 47455 (errors) | 998544 (bits)
IP_llr   = 52434 (errors) | 998544 (bits)
Es/No = 1.500000e-01
Euclid   = 88294 (errors) | 998544 (bits)
MAX      = 38853 (errors) | 998544 (bits)
llr2     = 38853 (errors) | 998544 (bits)
IP_llr   = 44192 (errors) | 998544 (bits)
Es/No = 2.000000e-01
Euclid   = 80545 (errors) | 998544 (bits)
MAX      = 28153 (errors) | 998544 (bits)
llr2     = 28153 (errors) | 998544 (bits)
IP_llr   = 33689 (errors) | 998544 (bits)
Es/No = 2.500000e-01
Euclid   = 76331 (errors) | 998544 (bits)
MAX      = 25047 (errors) | 998544 (bits)
llr2     = 25047 (errors) | 998544 (bits)
IP_llr   = 30226 (errors) | 998544 (bits)
Es/No = 3.000000e-01
Euclid   = 67430 (errors) | 998544 (bits)
MAX      = 16904 (errors) | 998544 (bits)
llr2     = 16904 (errors) | 998544 (bits)
IP_llr   = 21287 (errors) | 998544 (bits)
Es/No = 3.500000e-01
Euclid   = 57493 (errors) | 998544 (bits)
MAX      = 11789 (errors) | 998544 (bits)
llr2     = 11789 (errors) | 998544 (bits)
IP_llr   = 15160 (errors) | 998544 (bits)
Es/No = 4.000000e-01
Euclid   = 44143 (errors) | 998544 (bits)
MAX      =   5427 (errors) | 998544 (bits)
llr2     =   5427 (errors) | 998544 (bits)
IP_llr   =   7475 (errors) | 998544 (bits)
Es/No = 4.500000e-01
Euclid   = 37088 (errors) | 998544 (bits)
MAX      =   2087 (errors) | 998544 (bits)
llr2     =   2087 (errors) | 998544 (bits)
IP_llr   =   3426 (errors) | 998544 (bits)
Es/No = 5.000000e-01
Euclid   = 27794 (errors) | 998544 (bits)
MAX      =   1257 (errors) | 998544 (bits)
llr2     =   1257 (errors) | 998544 (bits)
IP_llr   =   2024 (errors) | 998544 (bits)
Es/No = 5.500000e-01
Euclid   = 15363 (errors) | 998544 (bits)
```

```
MAX     =   415 (errors) | 998544 (bits)
llr2    =   415 (errors) | 998544 (bits)
IP_llr  =   640 (errors) | 998544 (bits)
```

```
Es/No = 6.000000e-01
Euclid  = 13348 (errors) | 998544 (bits)
MAX     =   131 (errors) | 998544 (bits)
llr2    =   131 (errors) | 998544 (bits)
IP_llr  =   300 (errors) | 998544 (bits)
```

```
Es/No = 6.500000e-01
Euclid  = 4436 (errors) | 998544 (bits)
MAX     =    0 (errors) | 998544 (bits)
llr2    =    0 (errors) | 998544 (bits)
IP_llr  =   72 (errors) | 998544 (bits)
```

Appendix A.3 – 8PSK Modulation (3 BPS) – 10^5 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
Encoding 14 Blocks of 16k at rate ½
# BPS = 3 #
Es/No = 3
Euclid   = 11990 (errors) | 98448 (bits)
MAX      =   7360 (errors) | 98448 (bits)
llr2     =   8084 (errors) | 98448 (bits)
IP_llr   =   8055 (errors) | 98448 (bits)
Es/No = 3.050000e+00
Euclid   = 10590 (errors) | 98448 (bits)
MAX      =   5430 (errors) | 98448 (bits)
llr2     =   6086 (errors) | 98448 (bits)
IP_llr   =   6078 (errors) | 98448 (bits)
Es/No = 3.100000e+00
Euclid   = 10238 (errors) | 98448 (bits)
MAX      =   4519 (errors) | 98448 (bits)
llr2     =   5328 (errors) | 98448 (bits)
IP_llr   =   5511 (errors) | 98448 (bits)
Es/No = 3.150000e+00
Euclid   = 10368 (errors) | 98448 (bits)
MAX      =   5528 (errors) | 98448 (bits)
llr2     =   6051 (errors) | 98448 (bits)
IP_llr   =   6150 (errors) | 98448 (bits)
Es/No = 3.200000e+00
Euclid   = 9013 (errors) | 98448 (bits)
MAX      = 3516 (errors) | 98448 (bits)
llr2     = 4070 (errors) | 98448 (bits)
IP_llr   = 4452 (errors) | 98448 (bits)
Es/No = 3.250000e+00
Euclid   = 9350 (errors) | 98448 (bits)
MAX      = 2543 (errors) | 98448 (bits)
llr2     = 3031 (errors) | 98448 (bits)
IP_llr   = 3414 (errors) | 98448 (bits)
Es/No = 3.300000e+00
Euclid   = 8441 (errors) | 98448 (bits)
MAX      = 1819 (errors) | 98448 (bits)
llr2     = 2133 (errors) | 98448 (bits)
IP_llr   = 2399 (errors) | 98448 (bits)
Es/No = 3.350000e+00
Euclid   = 7529 (errors) | 98448 (bits)
MAX      = 1152 (errors) | 98448 (bits)
llr2     = 1564 (errors) | 98448 (bits)
IP_llr   = 1533 (errors) | 98448 (bits)
```

```
Es/No = 3.400000e+00
Euclid   = 5950 (errors) | 98448 (bits)
MAX      =   256 (errors) | 98448 (bits)
llr2     =   349 (errors) | 98448 (bits)
IP_llr   =   438 (errors) | 98448 (bits)
Es/No = 3.450000e+00
Euclid   = 5046 (errors) | 98448 (bits)
MAX      =   236 (errors) | 98448 (bits)
llr2     =   382 (errors) | 98448 (bits)
IP_llr   =   461 (errors) | 98448 (bits)
Es/No = 3.500000e+00
Euclid   = 4782 (errors) | 98448 (bits)
MAX      =   136 (errors) | 98448 (bits)
llr2     =     83 (errors) | 98448 (bits)
IP_llr   =   258 (errors) | 98448 (bits)
Es/No = 3.550000e+00
Euclid   = 2081 (errors) | 98448 (bits)
MAX      =     19 (errors) | 98448 (bits)
llr2     =     37 (errors) | 98448 (bits)
IP_llr   =     34 (errors) | 98448 (bits)
Es/No = 3.600000e+00
Euclid   = 1032 (errors) | 98448 (bits)
MAX      =       0 (errors) | 98448 (bits)
llr2     =       0 (errors) | 98448 (bits)
IP_llr   =       0 (errors) | 98448 (bits)
```

Appendix A.4 – 8PSK Modulation (3 BPS) – 10^6 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
```

```
Encoding 142 Blocks of 16k at rate ½
# BPS = 3 #
```

```
Es/No = 3
Euclid   = 118244 (errors) | 998544 (bits)
MAX      =   73755 (errors) | 998544 (bits)
llr2     =   80947 (errors) | 998544 (bits)
IP_llr   =   81263 (errors) | 998544 (bits)
Es/No = 3.050000e+00
Euclid   = 114535 (errors) | 998544 (bits)
MAX      =   65922 (errors) | 998544 (bits)
llr2     =   72592 (errors) | 998544 (bits)
IP_llr   =   72323 (errors) | 998544 (bits)
Es/No = 3.100000e+00
Euclid   = 110125 (errors) | 998544 (bits)
MAX      =   58144 (errors) | 998544 (bits)
llr2     =   65087 (errors) | 998544 (bits)
IP_llr   =   65399 (errors) | 998544 (bits)
Es/No = 3.150000e+00
Euclid   = 102975 (errors) | 998544 (bits)
MAX      =   45361 (errors) | 998544 (bits)
llr2     =   50866 (errors) | 998544 (bits)
IP_llr   =   51381 (errors) | 998544 (bits)
Es/No = 3.200000e+00
Euclid   = 97509 (errors) | 998544 (bits)
MAX      = 32395 (errors) | 998544 (bits)
llr2     = 38883 (errors) | 998544 (bits)
IP_llr   = 39188 (errors) | 998544 (bits)
Es/No = 3.250000e+00
Euclid   = 90388 (errors) | 998544 (bits)
MAX      = 26102 (errors) | 998544 (bits)
llr2     = 32745 (errors) | 998544 (bits)
IP_llr   = 32200 (errors) | 998544 (bits)
Es/No = 3.300000e+00
Euclid   = 83637 (errors) | 998544 (bits)
MAX      = 18390 (errors) | 998544 (bits)
llr2     = 22748 (errors) | 998544 (bits)
IP_llr   = 22491 (errors) | 998544 (bits)
Es/No = 3.350000e+00
Euclid   = 72726 (errors) | 998544 (bits)
MAX      = 10810 (errors) | 998544 (bits)
llr2     = 13792 (errors) | 998544 (bits)
```

```
IP_llr   = 14825 (errors) | 998544 (bits)
Es/No = 3.400000e+00
Euclid   = 65065 (errors) | 998544 (bits)
MAX      =  5753 (errors) | 998544 (bits)
llr2     =  7537 (errors) | 998544 (bits)
IP_llr   =  8426 (errors) | 998544 (bits)
Es/No = 3.450000e+00
Euclid   = 48892 (errors) | 998544 (bits)
MAX      =  2326 (errors) | 998544 (bits)
llr2     =  3167 (errors) | 998544 (bits)
IP_llr   =  3071 (errors) | 998544 (bits)
Es/No = 3.500000e+00
Euclid   = 38161 (errors) | 998544 (bits)
MAX      =  1201 (errors) | 998544 (bits)
llr2     =  1553 (errors) | 998544 (bits)
IP_llr   =  1737 (errors) | 998544 (bits)
Es/No = 3.550000e+00
Euclid   = 31043 (errors) | 998544 (bits)
MAX      =   528 (errors) | 998544 (bits)
llr2     =   630 (errors) | 998544 (bits)
IP_llr   =   860 (errors) | 998544 (bits)
Es/No = 3.600000e+00
Euclid   = 20234 (errors) | 998544 (bits)
MAX      =   139 (errors) | 998544 (bits)
llr2     =   233 (errors) | 998544 (bits)
IP_llr   =   500 (errors) | 998544 (bits)
Es/No = 3.650000e+00
Euclid   = 12041 (errors) | 998544 (bits)
MAX      =    66 (errors) | 998544 (bits)
llr2     =   195 (errors) | 998544 (bits)
IP_llr   =   116 (errors) | 998544 (bits)
Es/No = 3.700000e+00
Euclid   = 7057 (errors) | 998544 (bits)
MAX      =    0 (errors) | 998544 (bits)
llr2     =   12 (errors) | 998544 (bits)
IP_llr   =   13 (errors) | 998544 (bits)
```

Appendix A.5 – 16APSK Modulation (4 BPS) – 10^5 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
Encoding 14 Blocks of 16k at rate ½
# BPS = 4 #
Es/No = 5
Euclid   = 13540 (errors) | 98448 (bits)
MAX      = 10024 (errors) | 98448 (bits)
llr2     = 10757 (errors) | 98448 (bits)
IP_llr   = 11740 (errors) | 98448 (bits)
Es/No = 5.100000e+00
Euclid   = 12737 (errors) | 98448 (bits)
MAX      =  8481 (errors) | 98448 (bits)
llr2     =  9797 (errors) | 98448 (bits)
IP_llr   = 10696 (errors) | 98448 (bits)
Es/No = 5.200000e+00
Euclid   = 12155 (errors) | 98448 (bits)
MAX      =  7698 (errors) | 98448 (bits)
llr2     =  8888 (errors) | 98448 (bits)
IP_llr   = 10177 (errors) | 98448 (bits)
Es/No = 5.300000e+00
Euclid   = 11191 (errors) | 98448 (bits)
MAX      =  5872 (errors) | 98448 (bits)
llr2     =  7478 (errors) | 98448 (bits)
IP_llr   =  8491 (errors) | 98448 (bits)
Es/No = 5.400000e+00
Euclid   = 10303 (errors) | 98448 (bits)
MAX      =  4889 (errors) | 98448 (bits)
llr2     =  6803 (errors) | 98448 (bits)
IP_llr   =  8302 (errors) | 98448 (bits)
Es/No = 5.500000e+00
Euclid   = 9177 (errors) | 98448 (bits)
MAX      = 2875 (errors) | 98448 (bits)
llr2     = 3991 (errors) | 98448 (bits)
IP_llr   = 6157 (errors) | 98448 (bits)
Es/No = 5.600000e+00
Euclid   = 7705 (errors) | 98448 (bits)
MAX      = 718 (errors) | 98448 (bits)
llr2     = 1843 (errors) | 98448 (bits)
IP_llr   = 3276 (errors) | 98448 (bits)
Es/No = 5.700000e+00
Euclid   = 6453 (errors) | 98448 (bits)
MAX      =  362 (errors) | 98448 (bits)
llr2     =  626 (errors) | 98448 (bits)
```

```
IP_llr    = 1838 (errors) | 98448 (bits)
Es/No = 5.800000e+00
Euclid   = 4323 (errors) | 98448 (bits)
MAX      =   215 (errors) | 98448 (bits)
llr2       =   468 (errors) | 98448 (bits)
IP_llr    =   997 (errors) | 98448 (bits)
Es/No = 5.900000e+00
Euclid   = 2827 (errors) | 98448 (bits)
MAX      =     52 (errors) | 98448 (bits)
llr2        =    39 (errors) | 98448 (bits)
IP_llr    =   397 (errors) | 98448 (bits)
Es/No = 6
Euclid   = 987 (errors) | 98448 (bits)
MAX      =     0 (errors) | 98448 (bits)
llr2       =     0 (errors) | 98448 (bits)
IP_llr    =   85 (errors) | 98448 (bits)
```

Appendix A.6 – 16APSK Modulation (4 BPS) – 10^6 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:"16200":The coded size (read-only)
   usize:UNSIGNED:"7032":The input size (read-only)
   enc_config:STRING:"dvb_12_16.encbin":Encoder configuration file
   interleave_depth:INT:"1":Encode Interleave Depth
   backwards:INT:"0":Backwards symbol interleave (for 8psk 3/5 S2 code)
Encoding 142 Blocks of 16k at rate ½
# BPS = 4 #
Es/No = 5
Euclid   = 134840 (errors) | 998544 (bits)
MAX      =   98437 (errors) | 998544 (bits)
llr2     = 109776 (errors) | 998544 (bits)
IP_llr   = 118562 (errors) | 998544 (bits)
Es/No = 5.100000e+00
Euclid   = 130117 (errors) | 998544 (bits)
MAX      =   90050 (errors) | 998544 (bits)
llr2     = 101884 (errors) | 998544 (bits)
IP_llr   = 111370 (errors) | 998544 (bits)
Es/No = 5.200000e+00
Euclid   = 120735 (errors) | 998544 (bits)
MAX      =   75148 (errors) | 998544 (bits)
llr2     =   89174 (errors) | 998544 (bits)
IP_llr   =   99785 (errors) | 998544 (bits)
Es/No = 5.300000e+00
Euclid   = 114022 (errors) | 998544 (bits)
MAX      =   61597 (errors) | 998544 (bits)
llr2     =   76895 (errors) | 998544 (bits)
IP_llr   =   90103 (errors) | 998544 (bits)
Es/No = 5.400000e+00
Euclid   = 103272 (errors) | 998544 (bits)
MAX      =   47436 (errors) | 998544 (bits)
llr2     =   63363 (errors) | 998544 (bits)
IP_llr   =   76751 (errors) | 998544 (bits)
Es/No = 5.500000e+00
Euclid   = 91289 (errors) | 998544 (bits)
MAX      = 29898 (errors) | 998544 (bits)
llr2     = 43145 (errors) | 998544 (bits)
IP_llr   = 59596 (errors) | 998544 (bits)
Es/No = 5.600000e+00
Euclid   = 79605 (errors) | 998544 (bits)
MAX      = 17891 (errors) | 998544 (bits)
llr2     = 27080 (errors) | 998544 (bits)
IP_llr   = 42884 (errors) | 998544 (bits)
Es/No = 5.700000e+00
Euclid   = 65054 (errors) | 998544 (bits)
MAX      =   6988 (errors) | 998544 (bits)
llr2     = 10760 (errors) | 998544 (bits)
```

```
IP_llr    = 25751 (errors) | 998544 (bits)
Es/No = 5.800000e+00
Euclid   = 44445 (errors) | 998544 (bits)
MAX     =   1958 (errors) | 998544 (bits)
llr2      =   4438 (errors) | 998544 (bits)
IP_llr    = 11221 (errors) | 998544 (bits)
Es/No = 5.900000e+00
Euclid   = 25484 (errors) | 998544 (bits)
MAX     =     299 (errors) | 998544 (bits)
llr2      =     903 (errors) | 998544 (bits)
IP_llr    =   3565 (errors) | 998544 (bits)
Es/No = 6
Euclid   = 11666 (errors) | 998544 (bits)
MAX     =         0 (errors) | 998544 (bits)
llr2      =       52 (errors) | 998544 (bits)
IP_llr    =     616 (errors) | 998544 (bits)
```

Appendix A.7 – 32APSK Modulation (5 BPS) – 10^5 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
Encoding 14 Blocks of 16k at rate ½
# BPS = 5 #
Es/No = 7
Euclid   = 16057 (errors) | 98448 (bits)
MAX      = 12211 (errors) | 98448 (bits)
llr2     = 13006 (errors) | 98448 (bits)
IP_llr   = 15896 (errors) | 98448 (bits)
Es/No = 7.100000e+00
Euclid   = 15259 (errors) | 98448 (bits)
MAX      = 11309 (errors) | 98448 (bits)
llr2     = 12024 (errors) | 98448 (bits)
IP_llr   = 14988 (errors) | 98448 (bits)
Es/No = 7.200000e+00
Euclid   = 15068 (errors) | 98448 (bits)
MAX      = 10744 (errors) | 98448 (bits)
llr2     = 11551 (errors) | 98448 (bits)
IP_llr   = 14979 (errors) | 98448 (bits)
Es/No = 7.300000e+00
Euclid   = 14623 (errors) | 98448 (bits)
MAX      = 10037 (errors) | 98448 (bits)
llr2     = 10669 (errors) | 98448 (bits)
IP_llr   = 14080 (errors) | 98448 (bits)
Es/No = 7.400000e+00
Euclid   = 13640 (errors) | 98448 (bits)
MAX      =  8939 (errors) | 98448 (bits)
llr2     =  9636 (errors) | 98448 (bits)
IP_llr   = 12839 (errors) | 98448 (bits)
Es/No = 7.500000e+00
Euclid   = 12981 (errors) | 98448 (bits)
MAX      =  7859 (errors) | 98448 (bits)
llr2     =  8761 (errors) | 98448 (bits)
IP_llr   = 12611 (errors) | 98448 (bits)
Es/No = 7.600000e+00
Euclid   = 12230 (errors) | 98448 (bits)
MAX      =  7011 (errors) | 98448 (bits)
llr2     =  8189 (errors) | 98448 (bits)
IP_llr   = 12145 (errors) | 98448 (bits)
Es/No = 7.700000e+00
Euclid   = 11708 (errors) | 98448 (bits)
MAX      =  5517 (errors) | 98448 (bits)
llr2     =  6164 (errors) | 98448 (bits)
```

```
IP_llr    = 11091 (errors) | 98448 (bits)
Es/No = 7.800000e+00
Euclid   = 10583 (errors) | 98448 (bits)
MAX      =   3923 (errors) | 98448 (bits)
llr2       =   4877 (errors) | 98448 (bits)
IP_llr    =   9981 (errors) | 98448 (bits)
Es/No = 7.900000e+00
Euclid   = 9664 (errors) | 98448 (bits)
MAX      = 1878 (errors) | 98448 (bits)
llr2       = 2642 (errors) | 98448 (bits)
IP_llr    = 8253 (errors) | 98448 (bits)
Es/No = 8
Euclid   = 7998 (errors) | 98448 (bits)
MAX      = 1338 (errors) | 98448 (bits)
llr2       = 1832 (errors) | 98448 (bits)
IP_llr    = 7161 (errors) | 98448 (bits)
Es/No = 8.100000e+00
Euclid   = 6553 (errors) | 98448 (bits)
MAX      =   462 (errors) | 98448 (bits)
llr2       =   818 (errors) | 98448 (bits)
IP_llr    = 4739 (errors) | 98448 (bits)
Es/No = 8.200000e+00
Euclid   = 4511 (errors) | 98448 (bits)
MAX      =     23 (errors) | 98448 (bits)
llr2       =     72 (errors) | 98448 (bits)
IP_llr    = 3479 (errors) | 98448 (bits)
Es/No = 8.300000e+00
Euclid   = 4204 (errors) | 98448 (bits)
MAX      =       0 (errors) | 98448 (bits)
llr2       =       0 (errors) | 98448 (bits)
IP_llr    = 2335 (errors) | 98448 (bits
```

Appendix A.8 – 32APSK Modulation (5 BPS) – 10^6 Input Bits

```
BLOCK Configuration:
   Input Type: BLOCK_BIT_TYPE
   Output Type: BLOCK_BIT_TYPE
   esize:UNSIGNED:”16200”:The coded size (read-only)
   usize:UNSIGNED:”7032”:The input size (read-only)
   enc_config:STRING:”dvb_12_16.encbin”:Encoder configuration file
   interleave_depth:INT:”1”:Encode Interleave Depth
   backwards:INT:”0”:Backwards symbol interleave (for 8psk 3/5 S2 code)
```

```
Encoding 142 Blocks of 16k at rate ½
# BPS = 5 #
```

```
Es/No = 7
Euclid   = 163231 (errors) | 998544 (bits)
MAX      = 123049 (errors) | 998544 (bits)
llr2     = 131034 (errors) | 998544 (bits)
IP_llr   = 159464 (errors) | 998544 (bits)
```

```
Es/No = 7.100000e+00
Euclid   = 158591 (errors) | 998544 (bits)
MAX      = 117233 (errors) | 998544 (bits)
llr2     = 125375 (errors) | 998544 (bits)
IP_llr   = 155207 (errors) | 998544 (bits)
```

```
Es/No = 7.200000e+00
Euclid   = 152257 (errors) | 998544 (bits)
MAX      = 109154 (errors) | 998544 (bits)
llr2     = 119132 (errors) | 998544 (bits)
IP_llr   = 149193 (errors) | 998544 (bits)
```

```
Es/No = 7.300000e+00
Euclid   = 146780 (errors) | 998544 (bits)
MAX      = 100222 (errors) | 998544 (bits)
llr2     = 108765 (errors) | 998544 (bits)
IP_llr   = 141909 (errors) | 998544 (bits)
```

```
Es/No = 7.400000e+00
Euclid   = 138845 (errors) | 998544 (bits)
MAX      =   89204 (errors) | 998544 (bits)
llr2     =   97912 (errors) | 998544 (bits)
IP_llr   = 135471 (errors) | 998544 (bits)
```

```
Es/No = 7.500000e+00
Euclid   = 130809 (errors) | 998544 (bits)
MAX      =   77470 (errors) | 998544 (bits)
llr2     =   87771 (errors) | 998544 (bits)
IP_llr   = 126217 (errors) | 998544 (bits)
```

```
Es/No = 7.600000e+00
Euclid   = 125279 (errors) | 998544 (bits)
MAX      =   69434 (errors) | 998544 (bits)
llr2     =   76897 (errors) | 998544 (bits)
IP_llr   = 120698 (errors) | 998544 (bits)
```

```
Es/No = 7.700000e+00
Euclid   = 117005 (errors) | 998544 (bits)
MAX      =   53618 (errors) | 998544 (bits)
llr2     =   62860 (errors) | 998544 (bits)
```

```
IP_llr   = 110894 (errors) | 998544 (bits)
Es/No = 7.800000e+00
Euclid   = 107893 (errors) | 998544 (bits)
MAX      =   38822 (errors) | 998544 (bits)
llr2     =   46845 (errors) | 998544 (bits)
IP_llr   = 101450 (errors) | 998544 (bits)
Es/No = 7.900000e+00
Euclid   = 98508 (errors) | 998544 (bits)
MAX      = 23119 (errors) | 998544 (bits)
llr2     = 28121 (errors) | 998544 (bits)
IP_llr   = 90302 (errors) | 998544 (bits)
Es/No = 8
Euclid   = 82159 (errors) | 998544 (bits)
MAX      =   9864 (errors) | 998544 (bits)
llr2     = 12047 (errors) | 998544 (bits)
IP_llr   = 71357 (errors) | 998544 (bits)
Es/No = 8.100000e+00
Euclid   = 71505 (errors) | 998544 (bits)
MAX      =   4151 (errors) | 998544 (bits)
llr2     =   6406 (errors) | 998544 (bits)
IP_llr   = 60160 (errors) | 998544 (bits)
Es/No = 8.200000e+00
Euclid   = 55020 (errors) | 998544 (bits)
MAX      =   1120 (errors) | 998544 (bits)
llr2     =   1375 (errors) | 998544 (bits)
IP_llr   = 39869 (errors) | 998544 (bits)
Es/No = 8.300000e+00
Euclid   = 36237 (errors) | 998544 (bits)
MAX      =     220 (errors) | 998544 (bits)
llr2     =     210 (errors) | 998544 (bits)
IP_llr   = 23106 (errors) | 998544 (bits)
Es/No = 8.400000e+00
Euclid   = 21175 (errors) | 998544 (bits)
MAX      =       40 (errors) | 998544 (bits)
llr2     =       32 (errors) | 998544 (bits)
IP_llr   = 10213 (errors) | 998544 (bits)
Es/No = 8.500000e+00
Euclid   = 9834 (errors) | 998544 (bits)
MAX      =       0 (errors) | 998544 (bits)
llr2     =     23 (errors) | 998544 (bits)
IP_llr   = 3990 (errors) | 998544 (bits)
```

Appendix B VHDL Testing Results

Appendix B.1 Parallel by Two Testing Results

P2					
LUT_W	**Modulation**	**T0,T1,T2**	**Pass or Fail**	**Errors**	**Samples**
1	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
2	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
3	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984

		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
4	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
5	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
6	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984

		T2	PASS	0	159984
7	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
8	QPSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	8PSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	16APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984
	32APSK	T0	PASS	0	159984
		T1	PASS	0	159984
		T2	PASS	0	159984

Appendix B.2 Parallel by Four Testing Results

P4					
LUT_W	Modulation	T0,T1,T2	Pass or Fail	Errors	Samples
1	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
2	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
3	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968

		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
4	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
5	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
6	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968

7	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
8	QPSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	8PSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	16APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968
	32APSK	T0	PASS	0	19968
		T1	PASS	0	19968
		T2	PASS	0	19968

Appendix C Terms and Abbreviations

Appendix C.1 Field Terminology

Abbreviation	Definition
ACM	Adaptive Coding and Modulation
AGC	Automatic Gain Control
AWGN	Additive White Gaussian Noise
BPS	Bits Per Symbol
CCM	Constant Coding and Modulation
DOD	Department of Defense
IP	Intellectual Property
HDL	Hardware Description Language
HDR	High Data Rate
HDTV	High Definition Television
LDPC	Low-Density Parity Check
LLR	Log Likelihood Ratio
LSB	Least Significant Bit
LUT	Lookup Table
MPEG-2	Motion Picture Expert Group
MSB	Most Significant Bit
PDF	Probability Density Function
PSK	Phase Shift Key
QAM	Quadrature Amplitude Modulation
QEF	Quasi-Error Free
QPSK	Quadrature Phase Shift Keying
SNR	Signal to Noise Ratio
VCM	Variable Coding and Modulation
VHDL	VHSIC Hardware Description Language
8PSK	8-ary Phase Shift Keying
16APSK	16-ary Amplitude and Phase Shift Keying
32APSK	32-ary Amplitude and Phase Shift Keying

Appendix D MATLAB Scripts and Functions

Appendix D.1 demapper_euclid_algorithm.m

```
% ---------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: demapper_euclid_algorithm.m
% -- Author   : Brian Leslie
% --
% --------------------------------------------------------------------------

%% Description

% This function is used to generate the LLR values using the Euclid
% algorithm given a vector of I/Q Data Points based on the BPS and the
% variance of the input vector.

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% mapper.m

%% Code

function demodulated=demapper_euclid_algorithm(vector, bps, hard_decode)

% Hard decode Setup
if nargin<3
    hard_decode=0;
end

% Vector Setup
demodulated = zeros(1,length(vector)*bps);
d = zeros(2^bps,length(vector)*bps);
b = zeros(bps,length(vector));
is0 = cell(1,bps);
```

```
is1 = cell(1,bps);
P = zeros(1,2^bps);

% Ratio of the concentric circles of LLR points in the Constellation
ratio={0, 1, 1, 3.15, [2.84 5.27]};

% Map Expected Constellation points (P)
for i=0:2^bps-1
    P(i+1)=mapper(fliplr(dec2bin(i,bps)-48),bps,90,ratio{bps});
end

binary=fliplr(dec2bin(0:2^bps-1)-48);

for c=0:bps-1       %check bit 0 or 1
    is0{c+1}=find(binary(:,c+1)==0);
    is1{c+1}=find(binary(:,c+1)==1);
end

% is0 - set of vectors of indices in a binary number that have a value of 0
% is1 - set of vectors of indices in a binary number that have a value of 1

% The minimum distances that would make the bit a value of 0 are subtracted
% by the minimum distance that would make the bit a value of 1.  If the
% resulting subtraction is negative the bit has a value of 0, otherwise if
% the result is positive the value is 1.
%
%              is0          is1
%  LSB = min( d1,d3 )-min( d2,d4 );
%  MSB = min( d1,d2 )-min( d3,d4 );

for i=1:length(vector)
    for j=1:2^bps
        d(j,i)=abs(vector(i)-P(j));  % pi = r - si
    end

    for y=1:bps

        % Reset vectors
        closeToOne = zeros(1,2^(bps-1));
        closeToZero = zeros(1,2^(bps-1));

        for x=1:2^(bps-1)

            closeToOne(x) = d(is1{y}(x),i);
            closeToZero(x) = d(is0{y}(x),i);
        end
        % Determine the bits of the symbol from MSB to LSB
        % b1 - MSB
        % b2 - LSB
        b(y,i) = min( closeToZero ) - min( closeToOne ) ;
    end

    % Store the bits into the demodulated vector
    for j=bps:-1:1
        demodulated(bps*i-j+1) = b(bps-j+1,i);
    end

end
% Hard Decode
if hard_decode~=0
    bits=zeros(numel(demodulated),1);
    idx=(demodulated>=0);
    bits(idx)=1;
    demodulated=bits;
end
```

Appendix D.2 demapper_MAX_algorithm.m

```
% ----------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: demapper_MAX_algorithm.m
% -- Author   : Brian Leslie
% --
% ---------------------------------------------------------------------------

%% Description

% This function is used to generate the LLR values using the MAX algorithm
% given a vector of I/Q Data Points based on the BPS and the variance
% of the input vector.

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% mapper.m

%% Code
function demodulated=demapper_MAX_algorithm(vector, bps, variance)

variance = max(variance,1);   %variance of the white Gaussian noise

% Ratio of the concentric circles of LLR points in the Constellation
%ratio={0, 1, 1, 3.15, [2.84 5.27]};
```

```
ratio={0, 1, 1, 2.70, [2.64 4.64]};

% Defined Power Level
if (bps == 2)
    Power_Level = 49*sqrt(2);
    % FEC= 1/2 -> 49*sqrt(2);
    % FEC= 5/6 (7/8) -> 56*sqrt(2)
elseif (bps==3)
    Power_Level = 86;
else
    Power_Level = 90;
end

% Vector Setup
demodulated = zeros(1,length(vector)*bps);
p = zeros(2^bps,length(vector)*bps);
b = zeros(bps,length(vector));
is0 = cell(1,bps);
is1 = cell(1,bps);
P = zeros(1,2^bps);

% Map Expected Constellation points (P)
for i=0:2^bps-1
    P(i+1)=mapper(fliplr(dec2bin(i,bps)-48),bps,Power_Level,ratio{bps});
end

%figure;
%scatter(real(P),imag(P))

binary=fliplr(dec2bin(0:2^bps-1)-48);

for c=0:bps-1       %check bit 0 or 1
    is0{c+1}=find(binary(:,c+1)==0);
    is1{c+1}=find(binary(:,c+1)==1);
end

% is0 - set of vectors of indices in a binary number that have a value of 0
% is1 - set of vectors of indices in a binary number that have a value of 1

for i=1:length(vector)
    for j=1:2^bps
        p(j,i)=-(abs(vector(i)-P(j)).^2)/(2*variance);
    end

    for y=1:bps

        % Reset vectors
        closeToOne = zeros(1,2^(bps-1));
        closeToZero = zeros(1,2^(bps-1));

        for x=1:2^(bps-1)
            closeToOne(x) = p(is1{y}(x),i);
            closeToZero(x) = p(is0{y}(x),i);
        end
        % Determine the bits of the symbol from MSB to LSB
        % b1 - MSB || b2 - LSB
        b(y,i) = max( closeToOne ) - max ( closeToZero );
    end

    % Store the bits into the demodulated vector
    for j=bps:-1:1
        demodulated(bps*i-j+1) = b(bps-j+1,i);
    end

end
```

Appendix D.3 demapper_trueLLR_algorithm.m

```
% ---------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: demapper_trueLLR_algorithm.m
% -- Author   : Brian Leslie
% --
% --------------------------------------------------------------------------

%% Description

% This function is used to generate the LLR values using the true LLR
% algorithm given a vector of I/Q Data Points based on the BPS and the
% variance of the input vector.

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% mapper.m

%% Code
```

```
function demodulated=demapper_trueLLR_algorithm(vector, bps, variance)

variance = max(variance,1);   %variance of the white Gaussian noise

% Vector Setup
demodulated = zeros(1,length(vector)*bps);
p = zeros(2^bps,length(vector)*bps);
b = zeros(bps,length(vector));
is0 = cell(1,bps);
is1 = cell(1,bps);
P = zeros(1,2^bps);

% Ratio of the concentric circles of LLR points in the Constellation
ratio={0, 1, 1, 3.15, [2.84 5.27]};

% Defined Power Level
Power_Level = 90;

% Map Expected Constellation points (P)
for i=0:2^bps-1
    P(i+1)=mapper(fliplr(dec2bin(i,bps)-48),bps,Power_Level,ratio{bps});
end

binary=fliplr(dec2bin(0:2^bps-1)-48);

for c=0:bps-1       %check bit 0 or 1
    is0{c+1}=find(binary(:,c+1)==0);
    is1{c+1}=find(binary(:,c+1)==1);
end

% is0 - set of vectors of indices in a binary number that have a value of 0
% is1 - set of vectors of indices in a binary number that have a value of 1

% The minimum distances that would make the bit a value of 0 are subtracted
% by the minimum distance that would make the bit a value of 1.  If the
% resulting subtraction is negative the bit has a value of 0, otherwise if
% the result is positive the value is 1.
%
%              is0          is1
%  LSB = min( d1,d3 )-min( d2,d4 );
%  MSB = min( d1,d2 )-min( d3,d4 );

for i=1:length(vector)
    for j=1:2^bps
        p(j,i)=(1/sqrt(2*pi*variance))*exp(-1*(abs(vector(i)-
P(j))^2)/(2*variance));
    end

    for y=1:bps

        % Reset vectors
        closeToOne = zeros(1,2^(bps-1));
        closeToZero = zeros(1,2^(bps-1));
        for x=1:2^(bps-1)
            closeToOne(x) = p(is1{y}(x),i);
            closeToZero(x) = p(is0{y}(x),i);
        end
        % Determine the bits of the symbol from MSB to LSB
        % b1 - MSB
        % b2 - LSB
        b(y,i) = log( sum( closeToOne ) / sum ( closeToZero ) );
    end

    % Store the bits into the demodulated vector
    for j=bps:-1:1
        demodulated(bps*i-j+1) = b(bps-j+1,i);
    end

end
```

Appendix D.4 generate_DVBS2_BER.m

```
% ---------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: generate_DVBS2-BER.m
% -- Author   : Brian Leslie
% --
% --------------------------------------------------------------------------

%% Description

% This script is used to generate BER performance curves for different LLR
% Algorithms with different modulation schemes using the DVB-S2 Standard.
% Several of the modulations include:
% QPSK (BPS=2)
% 8PSK (BPS=3)
% 16APSK (BPS=4)
% 32APSK (BPS=5)
% The code should work for higher modulations but has not been implemented
% or tested.

% Execution Instructions:
% 1) Choose the number of input bits to run the tests with on the order of
% 10^amount.
% 2) Choose which algorithms to run or which plots to run.  The plots will
% automatically run the right algorithms depending on what is needed to
% generate the correct plot.
% 3) Choose a modulation range by selecting the start and end BPS values.
% 4) Save/Run

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% demapper_MAX_algorithm.m
% demapper_euclid_algorithm.m
% demapper_trueLLR_algorithm.m
% demapper_IP_algorithm.m
% demapper_old_mitre_LLR_algorithm.m
% dvbs2_enc.m
% dvbs2_dec.m

% Path and Command Window Setup
addpath ../mapper
addpath ../utilities
addpath ../demapper_algorithms
clc;
clear;
close all;

%% Code

% random bits
amount = 5;
in=round(rand(1*10^amount,1));

% setup              1 - yes | 0 - no
run_old_mitre_llr            = 0;
run_euclid_algorithm         = 0;
run_euclid_algorithm_sweep   = 0;
run_MAX_algorithm            = 0;
run_MAX_algorithm_interp     = 0;
run_MAX_algorithm_sweep      = 0;
run_MAX_algorithm_cv         = 0;
run_MAX_algorithm_1_var      = 0;
run_trueLLR_algorithm        = 0;
run_trueLLR_algorithm_sweep = 0;
run_IP_algorithm             = 0;
run_IP_algorithm_no_sweeep   = 0;

% Plots
run_plot    = 0; % Demapper method plot for all methods
run_plot2   = 0; % LLR Demapper Approx.(Direct) vs. True Plot (QPSK and 8PSK)
run_plot3   = 0; % MAX Demapper plot for Const vs. Calc Variance
run_plot4   = 0; % MAX vs. LLR vs. Euclid vs. IP(not scaled) Demapper plot
run_plot5   = 1; % MAX Demapper plot with sweeping Scale
run_plot6   = 0; % LLR Demapper plot with sweeping Scale
run_plot7   = 0; % IP LLR Demapper plot with no scale vs. scale
```

MITRE Approved for Public Release; Distribution Unlimited 14-0176

```
run_plot8   = 0; % LLR Algorithm Comparison Plot
run_plot9   = 0; % Euclid Demapper plot with sweeping Scale
run_plot10  = 0; % MAX vs. Interp Demapper Plot

% setup BPS range
startbps = 5;
endbps   = 5;

% Configurations
stop = 0;
index = 0;
scale = [10, 12, 15, 18, 23];
scale2 = [10, 9, 11, 12, 13];
ratio={0, 1, 1, 3.15, [2.84 5.27]};
modcod=[0, 4, 12, 18, 24];
target=[0, 48.5026, 56.7131, 55.6821, 49.5085];
const_var = {1, 7056.7, 3345.6, 280, 115};
rate_array = [0, 12, 23, 23, 34]; % changed bps3 from 35 to 23
backwards_away = [0, 0, 0, 0, 0];
iterations_array = [0, 35, 38, 55, 55];
rscale_range={1, 1:30, 1:30, 15:45, 30:60};
rscale_range_euclid={1, 1:30, 1:30, 1:30, 1:30};
scale_MAX = [1, 12, 17, 29, 45];
scale_LLR = [1, 12, 17, 22, 43];
scale_EUC = [1, 9, 12, 16, 20];

%range={[], .6:.05:15, 5:.05:15, 7.5:.05:15, 12:.05:15}; % current version
%range={[], .75:.0025:15, 6.7, 8.8, 12.8}; % var testing version
%range={[], .8:.01:15, 5.5:.01:15, 8.7:.01:15, 12.5:.01:15}; % Es/No test
version
%range={[], 0:.025:15, 3:.05:15, 5:.1:15, 7:.1:15};
%range={[], .6:.05:15, 5:.05:15, 6:.05:15, 8.5:.05:15}; before iteration
range={[], .7:.05:15, 5.5:.05:15, 8.7:.05:15, 12.5:.05:15};% version for
sweeping
%range={[], .60:.05:15, 10:.1:12, 12:.1:14, 15:.1:17};
%range={[], 5:.1:7, 10:.1:12, 12:.1:14, 15:.1:17};

col='kbgrc';

% plot setups
if ( run_plot )
    run_old_mitre_llr = 1;
    run_euclid_algorithm = 1;
    run_MAX_algorithm = 1;
    run_trueLLR_algorithm = 1;
    run_IP_algorithm = 1;
end
if ( run_plot2 )
    run_old_mitre_llr = 1;
    run_trueLLR_algorithm = 1;
end
if ( run_plot3 )
    run_MAX_algorithm = 1;
    run_MAX_algorithm_cv = 1;
    run_MAX_algorithm_1_var = 1;
end
if ( run_plot4 )
    run_trueLLR_algorithm = 1;
    run_MAX_algorithm = 1;
    run_euclid_algorithm = 1;
    run_IP_algorithm_no_sweeep = 1;
end
if ( run_plot5 )
    run_MAX_algorithm_sweep = 1;
    run_MAX_algorithm = 1;
end
if ( run_plot6 )
    run_trueLLR_algorithm_sweep = 1;
    run_trueLLR_algorithm = 1;
end
if ( run_plot7 )
    run_IP_algorithm = 1;
    run_IP_algorithm_no_sweeep = 1;
end
if ( run_plot8 )
    run_trueLLR_algorithm = 1;
    run_MAX_algorithm = 1;
    run_euclid_algorithm = 1;
    run_IP_algorithm = 1;
end
if ( run_plot9 )
    run_euclid_algorithm = 1;
    run_euclid_algorithm_sweep = 1;
end
if ( run_plot10 )
    run_MAX_algorithm = 1;
    run_MAX_algorithm_interp = 1;
end

for bps=startbps:endbps

    % tx
    rate=rate_array(bps);
    iterations = iterations_array(bps);
    [enc, esize, dsize]=dvbs2_enc(in,rate,64,1,backwards_away(bps)); % 16 to
64
    fec_rate=dsize/esize;
    %     enc=in;
    map=mapper(enc,bps,90,ratio{bps});
    c_pow=mean(real(map).^2+imag(map).^2);

    % rx
    ber=[];
    ber2=[];
    ber3=[];
    ber4=[];
    ber5=[];
    ber6=[];
    ber7=[];
    ber8=[];
    ber9=[];
    ber10=[];
    ber11=[];
    ber12=[];
    varout=[];
    ebno=[];
    fprintf(1,'\n###########\n# BPS = %d #\n###########\n\n',bps);
    for Es/No=range{bps}

        tempber7 = [];
        tempber8 = [];
        tempber11 = [];
        tempber12 = [];
        temperrs7 = [];
        temperrs8 = [];
        temperrs11 = [];
        temperrs12 = [];

        fprintf(1,'----------------------------------\n');
        fprintf(1,'esno = %d\n',esno);
        noisy=round(awgn(map,esno,'measured'));
        noise=noisy-map;
        n_pow=mean(real(noise).^2+imag(noise).^2);

        index = index + 1;
        ebno(index) = 10*log10(c_pow/n_pow)-10*log10(bps*fec_rate);

        if (run_MAX_algorithm || run_trueLLR_algorithm ||
run_MAX_algorithm_sweep || run_trueLLR_algorithm_sweep ||
run_MAX_algorithm_interp)
            variance = var(noise);
            varout(index) = variance;
        end

        if (run_old_mitre_llr && bps < 4)
            fprintf(1,'\nRunning run_old_mitre_llr...\n\n');
            demap=demapper_old_mitre_LLR_algorithm(noisy,bps,0);
            %      demap=demapper_euclid(noisy,bps,0)/4;
            demap=demap*scale(bps)/rms(demap);
            demap=rail(demap+32,0,63);
            dec=dvbs2_dec(demap, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
            errs=sum(abs(in(1:length(dec))-dec'));
            ber(index) = errs/length(dec);
            fprintf(1, 'MITRE LLR = %d (errors) | %d (bits)\n', errs,
length(dec));

            if (errs == 0)
                stop = 1;
            end
        end

        if ( run_euclid_algorithm )
```

```
            fprintf(1,'\nRunning run_euclid_algorithm...\n\n');
            demap3=demapper_euclid_algorithm(noisy,bps);
            demap3=demap3*scale_EUC(bps)/rms(demap3);
            demap3=rail(demap3+32,0,63);
            dec3=dvbs2_dec(demap3, rate, 64, esize, dsize, ebno, 1, 
backwards_away(bps), iterations);
            errs3=sum(abs(in(1:length(dec3))-dec3'));
            ber3(index) = errs3/length(dec3);
            fprintf(1, 'EUCLID    = %6d (errors) | %d (bits)\n', errs3, 
length(dec3));

            if (errs3 == 0)
                stop = 1;
            end
        end

        if ( run_euclid_algorithm_sweep )
            fprintf(1,'\nRunning run_euclid_algorithm_sweep...\n\n');
            tempdemap11=demapper_euclid_algorithm(noisy,bps);
            index2 = 0;
            for rscale=rscale_range_euclid{bps}
                index2 = index2 + 1;
                demap11=tempdemap11*rscale/rms(tempdemap11);
                demap11=rail(demap11+32,0,63);
                dec11=dvbs2_dec(demap11, rate, 64, esize, dsize, ebno, 1, 
backwards_away(bps), iterations);
                errs11=sum(abs(in(1:length(dec11))-dec11'));
                temperrs11(index2) = errs11;
                tempber11(index2) = errs11/length(dec11);
            end
            ber11 = [ber11 tempber11];
            array_loc = 0;
            fprintf(1,'EUCLID\n');
            for rscale=rscale_range_euclid{bps}
                array_loc = array_loc + 1;
                fprintf(1, 'Scale = %d | %d (errors) | %d (bits)\n', rscale, 
temperrs11(array_loc), length(dec11));
            end

            if (min(temperrs11) == 0)
                stop = 1;
            end
        end

        if ( run_MAX_algorithm )
            fprintf(1,'\nRunning run_MAX_algorithm...\n\n');
            demap4=demapper_MAX_algorithm(noisy,bps,variance);
            demap4=demap4*scale_MAX(bps)/rms(demap4);
            demap4=rail(demap4+32,0,63);
            dec4=dvbs2_dec(demap4, rate, 64, esize, dsize, ebno, 1, 
backwards_away(bps), iterations);
            errs4=sum(abs(in(1:length(dec4))-dec4'));
            ber4(index) = errs4/length(dec4);
            fprintf(1, 'MAX       = %6d (errors) | %d (bits)\n', errs4,
length(dec4));
            if (errs4 == 0)
                stop = 1;
                last_esno_value(bps-1) = esno;
            end
        end

        if ( run_MAX_algorithm_interp )
            fprintf(1,'\nRunning run_MAX_algorithm_interp...\n\n');
            index2=0;
            for bitscale=0:7
                index2 = index2 + 1;
                demap12 = demapper_interp_llr(noisy,bps,variance,bitscale);
                demap12 = demap12*scale(bps)/rms(demap12);
                demap12= rail(demap12+32,0,63);
                dec12=dvbs2_dec(demap12, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
                errs12=sum(abs(in(1:length(dec12))-dec12'));
                temperrs12(index2) = errs12;
                tempber12(index2) = errs12/length(dec12);
            end
            ber12 = [ber12 tempber12];
            fprintf(1,'MAX\n');
            for bitscale=1:8
                fprintf(1, 'Bottom Bits = %d | %6d (errors) | %d (bits)\n',
(bitscale-1), temperrs12(bitscale), length(dec12));
            end
            if (min(temperrs12) == 0)
                stop = 1;
            end

        end

        if ( run_MAX_algorithm_sweep )
            fprintf(1,'\nRunning run_MAX_algorithm_sweep...\n\n');
            tempdemap7=demapper_MAX_algorithm(noisy,bps,variance);
            index2 = 0;
            for rscale=rscale_range{bps}
                index2 = index2 + 1;
                demap7=tempdemap7*rscale/rms(tempdemap7);
                demap7=rail(demap7+32,0,63);
                dec7=dvbs2_dec(demap7, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
                errs7=sum(abs(in(1:length(dec7))-dec7'));
                temperrs7(index2) = errs7;
                tempber7(index2) = errs7/length(dec7);
            end
            ber7 = [ber7 tempber7];
            array_loc = 0;
            fprintf(1,'MAX\n');
            for rscale=rscale_range{bps}
                array_loc = array_loc + 1;
                fprintf(1, 'Scale = %d | %d (errors) | %d (bits)\n', rscale,
temperrs7(array_loc), length(dec7));
```

```
            end

            if (min(temperrs7) == 0)
                stop = 1;
            end
        end

        if ( run_MAX_algorithm_cv )
            fprintf(1,'\nRunning run_MAX_algorithm_cv...\n\n');
            demap6=demapper_MAX_algorithm(noisy,bps,const_var{bps});
            demap6=demap6*scale(bps)/rms(demap6);
            demap6=rail(demap6+32,0,63);
            %dumpdemap = demap6(1:10)';
            %dumpdemap
            dec6=dvbs2_dec(demap6, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
            errs6=sum(abs(in(1:length(dec6))-dec6'));
            ber6(index) = errs6/length(dec6);
            fprintf(1, 'MAX       = %d (errors) | %d (bits)\n', errs6,
length(dec6));

            if (errs6 == 0)
                stop = 1;
            end
        end

        if ( run_MAX_algorithm_1_var )
            fprintf(1,'\nRunning run_MAX_algorithm_1_var...\n\n');
            demap10=demapper_MAX_algorithm(noisy,bps,1);
            demap10=demap10*scale(bps)/rms(demap10);
            demap10=rail(demap10+32,0,63);
            %dumpdemap = demap10(1:10)';
            %dumpdemap
            dec10=dvbs2_dec(demap10, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
            errs10=sum(abs(in(1:length(dec10))-dec10'));
            ber10(index) = errs10/length(dec10);
            fprintf(1, 'MAX       = %d (errors) | %d (bits)\n', errs10,
length(dec10));

            if (errs10 == 0)
                stop = 1;
            end
        end

        if ( run_trueLLR_algorithm )
            fprintf(1,'\nRunning run_trueLLR_algorithm...\n\n');
            demap5=demapper_trueLLR_algorithm(noisy,bps,variance);
            demap5=demap5*scale_LLR(bps)/rms(demap5);
            demap5=rail(demap5+32,0,63);
            dec5=dvbs2_dec(demap5, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
            errs5=sum(abs(in(1:length(dec5))-dec5'));
            ber5(index) = errs5/length(dec5);
            fprintf(1, 'TRUE LLR  = %6d (errors) | %d (bits)\n', errs5,
length(dec5));

            if (errs5 == 0)
                %stop = 1;
            end
        end

        if ( run_trueLLR_algorithm_sweep )
            fprintf(1,'\nRunning run_trueLLR_algorithm_sweep...\n\n');
            tempdemap8=demapper_trueLLR_algorithm(noisy,bps,variance);
            index2 = 0;
            for rscale=rscale_range{bps}
                index2 = index2 + 1;
                demap8=tempdemap8*rscale/rms(tempdemap8);
                demap8=rail(demap8+32,0,63);
                dec8=dvbs2_dec(demap8, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations);
                errs8=sum(abs(in(1:length(dec8))-dec8'));
                temperrs8(index2) = errs8;
                tempber8(index2) = errs8/length(dec8);
            end
            ber8 = [ber8 tempber8];
            array_loc = 0;
            fprintf(1,'TRUE LLR\n');
            for rscale=rscale_range{bps}
                array_loc = array_loc + 1;
                fprintf(1, 'Scale = %d | TRUE LLR  = %d (errors) | %d
(bits)\n', rscale, temperrs8(array_loc), length(dec8));
            end

            if (min(temperrs8) == 0)
                stop = 1;
            end
        end

        if ( run_IP_algorithm )
            fprintf(1,'\nRunning run_IP_algorithm...\n\n');

tempnoisy=noisy*target(bps)/sqrt(mean(real(noisy).^2+imag(noisy).^2)/2);
            demap2=demapper_IP_algorithm(tempnoisy/128,modcod(bps));
            %demap2=demap2*scale(bps)/rms(demap2);
            demap2=rail(demap2+32,0,63);
            dec2=dvbs2_dec(demap2, rate, 64, esize, dsize, ebno, 1,
backwards_away(bps), iterations); % 16 to 64
            errs2=sum(abs(in(1:length(dec2))-dec2'));
            ber2(index) = errs2/length(dec2);
            fprintf(1, 'IP LLR    = %6d (errors) | %d (bits)\n', errs2,
length(dec2));

            if (errs2 == 0)
                %stop = 1;
            end
        end
        if ( run_IP_algorithm_no_sweeep )
            fprintf(1,'\nRunning run_IP_algorithm_no_sweeep...\n\n');

tempnoisy=noisy*target(bps)/sqrt(mean(real(noisy).^2+imag(noisy).^2)/2);
            demap9=demapper_IP_algorithm(tempnoisy/128,modcod(bps));
            %demap9=demap2*scale(bps)/rms(demap2); % compare without scale
            demap9=rail(demap9+32,0,63);
            dec9=dvbs2_dec(demap9, rate, 64, esize, dsize, ebno, 1, 
backwards_away(bps), iterations);
            errs9=sum(abs(in(1:length(dec9))-dec9'));
            ber9(index) = errs9/length(dec9);
            fprintf(1, 'IP LLR   = %d (errors) | %d (bits)\n', errs9, 
length(dec9));

            if (errs9 == 0)
                stop = 1;
            end
        end

        if ( stop )
            stop = 0;
            break
        end
    end

    if ( run_plot )
        figure;
        if (run_old_mitre_llr && bps < 4)
            semilogy(ebno,ber,'r','linewidth',3)
        end

        if ( bps < 4 )
            hold on
        end

        if ( run_euclid_algorithm )
            semilogy(ebno,ber3,'b','linewidth',3)
        end

        if ( bps > 3 )
            hold on
        end

        if ( run_IP_algorithm )
            semilogy(ebno,ber2,'k','linewidth',3)
        end

        if ( run_MAX_algorithm )
            semilogy(ebno,ber4,'g','linewidth',3)
        end

        if ( run_trueLLR_algorithm )
```

```
            semilogy(ebno,ber5,'m','linewidth',3)
        end

        grid on
        if ( bps < 4 )
            legend('LLR','Euclid','IP LLR','MAX','True LLR');
        else
            legend('Euclid','IP LLR','MAX','True LLR');
        end
        title(['ber vs esno demapper method plot for BPS=',48+bps,', Number
of Bits= 1*10^',48+amount]);
    end


    if ( run_plot2 )
        figure;
        semilogy(ebno,ber,'r','linewidth',3)
        hold on
        semilogy(ebno,ber5,'b','linewidth',3)
        grid on
        legend('LLR Approx','LLR True')
        title(['BER vs ESNO, LLR Demapper Approx vs True Plot for
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
    end


    if ( run_plot3 )
        figure;
        semilogy(ebno,ber4,'r','linewidth',3);
        hold on
        semilogy(ebno,ber6,'b','linewidth',3);
        semilogy(ebno,ber10,'k','linewidth',3);
        grid on
        legend('Variable Variance','Constant Variance','Variance of One');
        title(['BER vs ESNO, MAX Demapper plot for BPS=',48+bps,', Number of
Bits= 1*10^',48+amount]);
    end

    if ( run_plot4 )
        figure;
        semilogy(ebno,ber5,'b','linewidth',3);
        hold on
        semilogy(ebno,ber4,'r','linewidth',3);
        semilogy(ebno,ber3,'k','linewidth',3);
        semilogy(ebno,ber9,'g','linewidth',3);
        grid on
        legend('LLR(scale)','MAX(scale)','Euclid(scale)','IP LLR(no scale)');
        title(['BER vs ESNO, MAX vs LLR vs Euclid vs IP Demapper plot for
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
    end

    if ( run_plot5 )
        figure;
        semilogy(ebno,ber7(8,:),'b','linewidth',2)
        hold on
        semilogy(ebno,ber7(10,:),'r','linewidth',2)
        semilogy(ebno,ber7(12,:),'g','linewidth',2)
        semilogy(ebno,ber7(15,:),'y','linewidth',2)
        semilogy(ebno,ber7(18,:),'k','linewidth',2)
        semilogy(ebno,ber7(23,:),'c','linewidth',2)
        semilogy(ebno,ber7(25,:),'m','linewidth',2)
        grid on
        legend('8','10','12','15','18','23','25');
        title(['BER vs ESNO, MAX Demapper plot with sweeping Scale for
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
    end

    if ( run_plot6 )
        figure;
        semilogy(ebno,ber8(8,:),'b','linewidth',2)
        hold on
        semilogy(ebno,ber8(10,:),'r','linewidth',2)
        semilogy(ebno,ber8(12,:),'g','linewidth',2)
        semilogy(ebno,ber8(15,:),'y','linewidth',2)
        semilogy(ebno,ber8(18,:),'k','linewidth',2)
        semilogy(ebno,ber8(23,:),'c','linewidth',2)
        semilogy(ebno,ber8(25,:),'m','linewidth',2)
        grid on
        legend('8','10','12','15','18','23','25');
        title(['BER vs ESNO, LLR Demapper plot with sweeping Scale for
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
    end

    if ( run_plot7 )
        figure;
        semilogy(ebno,ber2,'b','linewidth',3)
        hold on
        semilogy(ebno,ber9,'r','linewidth',3)
        grid on
        legend('IP with Scale','IP without Scale');
        title(['IP LLR Demapper plot with no scale vs scale for
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
    end

    if ( run_plot8 )
        figure;
        if ( bps==2)
            semilogy(ebno,ber2,'b','linewidth',3) % IP
            hold on
            semilogy(ebno,ber3,'r','linewidth',3) % euclid
            semilogy(ebno,ber4,'k','linewidth',3) % max/llr
            legend('IP','Euclid','MAX/LLR');
        else
            semilogy(ebno,ber2,'b','linewidth',3) % IP
            hold on
            semilogy(ebno,ber3,'r','linewidth',3) % euclid
            semilogy(ebno,ber4,'k','linewidth',3) % max
            semilogy(ebno,ber5,'g','linewidth',3) % llr
            legend('IP','Euclid','MAX','LLR');
        end
        grid on
        xlabel('EbNo [dB]');
        ylabel('BER');
        title(['LLR Algorithm Comparison Plot for BPS=',48+bps,', Number of 
Bits= 1*10^',48+amount]);
    end
    
    if ( run_plot9 )
        figure;
        semilogy(ebno,ber11(8,:),'b','linewidth',2)
        hold on
        semilogy(ebno,ber11(10,:),'r','linewidth',2)
        semilogy(ebno,ber11(12,:),'g','linewidth',2)
        semilogy(ebno,ber11(15,:),'y','linewidth',2)
        semilogy(ebno,ber11(18,:),'k','linewidth',2)
        semilogy(ebno,ber11(23,:),'c','linewidth',2)
        semilogy(ebno,ber11(25,:),'m','linewidth',2)
        grid on
        legend('8','10','12','15','18','23','25');
        title(['BER vs ESNO, LLR Demapper plot with sweeping Scale for 
BPS=',48+bps,', Number of Bits= 1*10^',48+amount]);
        pause(.1)
    end
    
    if ( run_plot10 )
        figure;
        semilogy(ebno,ber4,'b','linewidth',3);
        hold on
        semilogy(ebno,ber12(1,:),'g','linewidth',3);
        semilogy(ebno,ber12(2,:),'r','linewidth',3);
        semilogy(ebno,ber12(3,:),'c','linewidth',3);
        semilogy(ebno,ber12(4,:),'m','linewidth',3);
        semilogy(ebno,ber12(5,:),'y','linewidth',3);
        semilogy(ebno,ber12(6,:),'k','linewidth',3);
        %semilogy(ebno,ber12(7,:),'b','linewidth',3);
        %semilogy(ebno,ber12(8,:),'g','linewidth',3);
        %semilogy(ebno,ber12(9,:),'r','linewidth',3);
        grid on
        legend('MAX','Btm Bits=0','Btm Bits=1','Btm Bits=2','Btm Bits=3','Btm 
Bits=4','Btm Bits=5');
        title(['BER vs ESNO, MAX vs Interp Demapper Plot for BPS=',48+bps,', 
Number of Bits= 1*10^',48+amount]);
    end
    
end
```

Appendix D.5 generate_llrParams.m

```
% ----------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: generate_llrParams.m
% -- Author   : Brian Leslie
% --
% ---------------------------------------------------------------------------

%% Description

% This function is used to update the LLRParam files by adding the LUT
% values to be stored in RAM.

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Z:\designtop\src\components\llr\tb_rtl\testConfig\...

%% Code

function [ ] = generate_llrParams( data, bps )

% Modulation
modes={'qpsk', '8psk', '16apsk', '32apsk'};

% Test Types
testNumber={'t0', 't1', 't2'};

% mtc values based on BPS
mtc_value={'16', '48', '72', '96'};

% Parallel Test Mode
testMode={'p2', 'p4'};

for testModeLoop=1:length(testMode) % p2 or p4 % for
modesLoop=1:length(modes) % qpsk ...
    for testNumberLoop=1:length(testNumber) % t0 ..
        %fid=1;
        fprintf(1,'\n\n MODE: %s NAME: mitre_%s_%s\n',
testMode{testModeLoop}, modes{(bps-1)}, testNumber{testNumberLoop});
        fid=fopen(['Z:\designtop\src\components\llr\tb_rtl\testConfig\'
testMode{testModeLoop} '\mitre_' modes{(bps-1)} '_'
testNumber{testNumberLoop} '\llrParams.txt'],'w');

fprintf(fid,'#################################################################
################\n');
        fprintf(fid,'#\n');
        fprintf(fid,'# Parameter setting script. Automatically
generated.\n');
        fprintf(fid,'#\n');
        fprintf(fid,'# Note, a blank line halts processing of this
script\n');
        fprintf(fid,'#\n');

fprintf(fid,'#################################################################
################\n');
        fprintf(fid,'null null 1000000 setDelayConstant\n');
        fprintf(fid,'llrTop.llrRTLPSChannelps1 mct %s configure\n',
mtc_value{(bps-1)});
        fprintf(fid,'llrTop.llrRTLPSChannelps1 use_IP 0 configure\n');
        fprintf(fid,'llrTop.llrRTLPSChannelps1 addr 0 configure\n');
        fprintf(fid,'llrTop.llrRTLPSChannelps1 burst_incr 1 configure\n');

        for j=1:length(data)
            fprintf(fid,'llrTop.llrRTLPSChannelps1 data %d configure\n',
round(data(j)));
        end
        fclose(fid);
    end
end

end
```

Appendix D.6 generate_LUT.m

```
% ---------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: generate_LUT.m
% -- Author   : Brian Leslie
% --
% --------------------------------------------------------------------------

%% Description

% This script is used to generate the Lookup Tables for the LLR values.
% By creating a matrix of data points based on the precision specified for
% the LUT the LLR value is then generated at each of those points.  The LUT
% are different depending on the BPS because of how the LLR value is
% calculated.  This script generates the LUT for:
% QPSK (BPS=2)
% 8PSK (BPS=3)
% 16APSK (BPS=4)
% 32APSK (BPS=5)
% The code should work for higher modulations but has not been implemented
% or tested.

%% Files

% In order for this script to work the following scripts need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% demapper_MAX_algorithm.m
% generate_LLRParams.m

% Path and Command Window Setup
addpath ../mapper
addpath ../utilities
addpath ../demapper_algorithms
clc;
clear;
close all;

%% Code

% LUT Width - Number of Bits (MAX 8)
LUT_Width = 5;

% Create test vectors until the maximum modulation.
Max_Bps = 5;

% Update LLRParams File Control
% 1 - sends LUT to update the Param files
% 0 - Does NOT send LUT to update the Param files
LLRParams_Flag = 0;

% Save/Update MAT data files
% 1 - Save/Update
% 0 - Does NOT Save/Update
gen_MAT_Files = 0;

% Master Plot Control
Master_Plot_Flag = 0;

% Imagesc Plot OR Mesh Plot Control
% 1 - Imagesc Plot
% 0 - Mesh Plot
Imagesc_Plot_Flag = 1;
Mesh_Plot_Flag = ~Imagesc_Plot_Flag;

% Vector Setup
in = zeros(1,2^(2*LUT_Width));
sample = zeros(1,4);
```

```
% Indexing Variable
count = 0;

% Sampled Variance for each BPS value that is used to calculate the LLR
% values
%var = [ 1, 6964.5, 3477.7, 1546.8, 718.6246]; % OLD
var = [ 1, 6964.5, 1742.76, 832.07593, 259.93593]; % New

% Calc distance between I/Q Points in LUT
Dist_Point_to_Point = 2^(8-LUT_Width);
Max_Pos_Value = 128 - Dist_Point_to_Point;
Max_Neg_Value = 0 - Dist_Point_to_Point;

% Seperate I/Q plane into data points based on LUT percision.
for Q = [0:Dist_Point_to_Point:Max_Pos_Value -
128:Dist_Point_to_Point:Max_Neg_Value]
    for I = [0:Dist_Point_to_Point:Max_Pos_Value -
128:Dist_Point_to_Point:Max_Neg_Value]
        count = count + 1;
        in(count) = complex(I,Q);
    end
end

% Generate LUT
for BPS=2:Max_Bps
    % Calc LLR values for all I/Q Data Points for a given BPS
    LLR_MAX_Results = demapper_MAX_algorithm(in,BPS,var(BPS));

    % Sample Maximum LLR value
    %sample(BPS-1) = max(abs(LLR_MAX_Results));

    LLR_MAX_Results_Reshaped = reshape(LLR_MAX_Results,BPS,[])';

    for i=1:BPS
        sample(BPS-1,i) = max(abs(LLR_MAX_Results_Reshaped(:,i)));
        LLR_MAX_Results_Reshaped(:,i) =
LLR_MAX_Results_Reshaped(:,i).*(31/max(abs(LLR_MAX_Results_Reshaped(:,i))));
    end

    LLR_MAX_Results_Reshaped=rail(LLR_MAX_Results_Reshaped,-32,31);
    LLR_MAX_Results_Reshaped=round(LLR_MAX_Results_Reshaped);

    %LLR_MAX_Results=LLR_MAX_Results.*(31/max(abs(LLR_MAX_Results)));
    %LLR_MAX_Results=rail(LLR_MAX_Results,-32,31);
    %LLR_MAX_Results=round(LLR_MAX_Results);

    % Reshape Results
    %LLR_MAX_Results_Reshaped = reshape(LLR_MAX_Results,BPS,[])';

    if (Master_Plot_Flag)
        for i=1:BPS
            % Reshape Table for Plotting
            LLR_MAX_Results_Reshaped2 =
reshape(LLR_MAX_Results_Reshaped(:,i),32,32)';
            figure;
            if (Imagesc_Plot_Flag) % Imagesc Plot
                colormap('gray')
                imagesc(flipud(LLR_MAX_Results_Reshaped2))
            else % Mesh Plot
                mesh(flipud(LLR_MAX_Results_Reshaped2))
            end
        end
    end

    % Shift values that are negative by the offset value
    offset = 64;
    index = LLR_MAX_Results_Reshaped<0;
    LLR_MAX_Results_Reshaped(index) = LLR_MAX_Results_Reshaped(index)+offset;
    generated_LUT_Vector = zeros(length(LLR_MAX_Results_Reshaped(:,1)),1);

    % Reshape LUT into a Vector
    for i=1:BPS

generated_LUT_Vector=generated_LUT_Vector+(LLR_MAX_Results_Reshaped(:,i)*offs
et^(i-1));
    end
    data=generated_LUT_Vector;
    if (gen_MAT_Files)
        if (BPS==2)
            save('llr_data_qpsk.mat','data');
        elseif (BPS == 3)
            save('llr_data_8psk.mat','data');
        elseif (BPS == 4)
            save('llr_data_16apsk.mat','data');
        elseif (BPS == 5)
            save('llr_data_32apsk.mat','data');
        end
    end

    % Update Param Files with Updated LUT
    if (LLRParams_Flag)
        fprintf(1,'Generated LUT Vector for BPS = %d\n',BPS);
        generate_llrParams(generated_LUT_Vector',BPS);
    end
end
```

Appendix D.7 demapper_interp_llr.m

```
% ---------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: demapper_interp_llr.m
% -- Author   : Brian Leslie
% --
% ---------------------------------------------------------------------------

%% Description

% This function is used to calculate LLR values using bi-linear
% interpolation using the MAX algorithm.

%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% demapper_MAX_algorithm.m


%% Code

function [ output ] = demapper_interp_llr( noisy, BPS, variance, bottom_bits
)

format long;

% Flags: 1 - ON | 0 - OFF
% Print debug statements *WARNING* DO NOT RUN WITHOUT BREAKPOINT AT LINE
% 143
debug_flag = 0;
% Include Scatter plot of I/Q data with output
scatter_plot_flag = 0;

% LUT Width
LUT_Width = 8-bottom_bits;

% Create test vectors until the maximum modulation.
Max_Bps = 5;


%% LUT Scale Calculation

% Vector Setup
in = zeros(1,2^(2*LUT_Width));
sample = zeros(1,4);
count = 0;
var = [ 1, 6964.5, 1742.76, 832.07593, 259.93593]; % New
```

```
% Calc distance between I/Q Points in LUT
Dist_Point_to_Point = 2^(8-LUT_Width);
Max_Pos_Value = 128 - Dist_Point_to_Point;
Max_Neg_Value = 0 - Dist_Point_to_Point;

% Seperate I/Q plane into data points based on LUT percision.
for Q = [0:Dist_Point_to_Point:Max_Pos_Value -
128:Dist_Point_to_Point:Max_Neg_Value]
    for I = [0:Dist_Point_to_Point:Max_Pos_Value -
128:Dist_Point_to_Point:Max_Neg_Value]
        count = count + 1;
        in(count) = complex(I,Q);
    end
end

% Generate LUT
for temp_BPS=2:Max_Bps
    % Calc LLR values for all I/Q Data Points for a given BPS
    LLR_MAX_Results = demapper_MAX_algorithm(in,temp_BPS,var(temp_BPS));

    % Sample Maximum LLR value
    %sample(BPS-1) = max(abs(LLR_MAX_Results));

    LLR_MAX_Results_Reshaped = reshape(LLR_MAX_Results,temp_BPS,[])';

    for i=1:temp_BPS
        Max_LLR_Values(temp_BPS-1,i) =
max(abs(LLR_MAX_Results_Reshaped(:,i)));
    end
end


%% Function Code

% Vector Setup
I_btm_bits = zeros(1,length(noisy));
Q_btm_bits = zeros(1,length(noisy));

% Split noisy signal into I/Q Data
noisy_I = rail(real(noisy),-128,127);
noisy_Q = rail(imag(noisy),-128,127);

% Calc I_Low and I_High points
I_Low = floor((noisy_I+128)./(2^bottom_bits)).*(2^bottom_bits)-128;
I_High = I_Low+(2^bottom_bits);
too_high = (I_High == 128);
I_High(too_high) = 128-(2^bottom_bits);

% Calc Q_Low and Q_High points
Q_Low = floor((noisy_Q+128)./(2^bottom_bits)).*(2^bottom_bits)-128;
Q_High = Q_Low+(2^bottom_bits);
too_high = (Q_High == 128);
```

```
Q_High(too_high) = 128-(2^bottom_bits);

% Generate vector of bottom bits of the noisy I/Q Data to be used as the
% offset in the interpolation
%I_btm_bits = mod(noisy_I,(2^bottom_bits))';
%Q_btm_bits = mod(noisy_Q,(2^bottom_bits))';

if (scatter_plot_flag)
    scatter(I_Low(1),Q_Low(1),'r');
    hold on
    scatter(I_Low(1),Q_High(1),'g');
    scatter(I_High(1),Q_Low(1),'b');
    scatter(I_High(1),Q_High(1),'k');
    axis([-128 127 -128 127]);
end

% Calculate the LLR values at each of the 4 Points
fprintf(1,'----------------\nBottom Bits = %d\n', bottom_bits);
results_I_L_Q_L = demapper_MAX_algorithm(complex(I_Low,Q_Low),BPS,variance);
results_I_L_Q_H = demapper_MAX_algorithm(complex(I_Low,Q_High),BPS,variance);
results_I_H_Q_L = demapper_MAX_algorithm(complex(I_High,Q_Low),BPS,variance);
results_I_H_Q_H =
demapper_MAX_algorithm(complex(I_High,Q_High),BPS,variance);

results_I_L_Q_L_Reshaped = reshape(results_I_L_Q_L,BPS,[])';
results_I_L_Q_H_Reshaped = reshape(results_I_L_Q_H,BPS,[])';
results_I_H_Q_L_Reshaped = reshape(results_I_H_Q_L,BPS,[])';
results_I_H_Q_H_Reshaped = reshape(results_I_H_Q_H,BPS,[])';

%results_I_L_Q_L_Reshaped2 = reshape(results_I_L_Q_L,BPS,[])';

for i=1:BPS
    %sample_scale_results(BPS-1,i) = max(abs(results_I_L_Q_L_Reshaped(:,i)));
    results_I_L_Q_L_Reshaped(:,i) =
results_I_L_Q_L_Reshaped(:,i).*(31/Max_LLR_Values(BPS-1,i));
    results_I_L_Q_H_Reshaped(:,i) =
results_I_L_Q_H_Reshaped(:,i).*(31/Max_LLR_Values(BPS-1,i));
    results_I_H_Q_L_Reshaped(:,i) =
results_I_H_Q_L_Reshaped(:,i).*(31/Max_LLR_Values(BPS-1,i));
    results_I_H_Q_H_Reshaped(:,i) =
results_I_H_Q_H_Reshaped(:,i).*(31/Max_LLR_Values(BPS-1,i));

    %results_I_L_Q_L_Reshaped2(:,i) =
results_I_L_Q_L_Reshaped2(:,i).*(31/sample_scale_results(BPS-1,i));
    %sample_scale_results(BPS-1,i)
    %Max_LLR_Values(BPS-1,i)
end

I_L_Q_L = round(reshape(results_I_L_Q_L_Reshaped',1,[]));
I_L_Q_H = round(reshape(results_I_L_Q_H_Reshaped',1,[]));
I_H_Q_L = round(reshape(results_I_H_Q_L_Reshaped',1,[]));
I_H_Q_H = round(reshape(results_I_H_Q_H_Reshaped',1,[]));
```

```
% Bi-Linear Interpolation
for j = 1:length(noisy)

    x = noisy_I(j); %(I_btm3_bits./(2^bottom_bits)) + I_L_Q_L;
    y = noisy_Q(j); %(Q_btm3_bits./(2^bottom_bits)) + I_L_Q_L;
    x1 = I_Low(j);
    x2 = I_High(j);
    y1 = Q_Low(j);
    y2 = Q_High(j);
    btm_bit_offset_I_direction = x-x1;  % bottom bit offset for I direction
    btm_bit_offset_I_direction_m = x2-x;
    btm_bit_offset_Q_direction = y-y1; % bottom bit offset for Q direction
    btm_bit_offset_Q_direction_m = y2-y;

    if (debug_flag)
        fprintf(1,'============================================');
        fprintf(1,'\nx=%d | y=%d\n', x, y);
        fprintf(1,'x1=%d | x2=%d | y1=%d | y2=%d\n', x1, x2, y1, y2);
        fprintf(1,'btm_bit_offset_I_direction=%d |
btm_bit_offset_I_direction_m=%d\n', btm_bit_offset_I_direction,
btm_bit_offset_I_direction_m);
        fprintf(1,'btm_bit_offset_Q_direction=%d |
btm_bit_offset_Q_direction_m=%d\n', btm_bit_offset_Q_direction,
btm_bit_offset_Q_direction_m);
    end

    for i=1:BPS
        if (x2==x1)
            horizontal_low_interp(BPS*(j-1)+i) = I_L_Q_L(BPS*(j-
1)+i)*(2^bottom_bits); %R1
            horizontal_high_interp(BPS*(j-1)+i) = I_L_Q_H(BPS*(j-
1)+i)*(2^bottom_bits); %R2

        else
            horizontal_low_interp_left(BPS*(j-1)+i) =
btm_bit_offset_I_direction*I_H_Q_L(BPS*(j-1)+i);
            horizontal_low_interp_right(BPS*(j-1)+i) =
btm_bit_offset_I_direction_m*I_L_Q_L(BPS*(j-1)+i);

            horizontal_high_interp_left(BPS*(j-1)+i) =
btm_bit_offset_I_direction*I_H_Q_H(BPS*(j-1)+i);
            horizontal_high_interp_right(BPS*(j-1)+i) =
btm_bit_offset_I_direction_m*I_L_Q_H(BPS*(j-1)+i);

            horizontal_low_interp(BPS*(j-1)+i) =
horizontal_low_interp_left(BPS*(j-1)+i)+horizontal_low_interp_right(BPS*(j-
1)+i);
            horizontal_high_interp(BPS*(j-1)+i) =
horizontal_high_interp_left(BPS*(j-1)+i)+horizontal_high_interp_right(BPS*(j-
1)+i);

            if (debug_flag)
```

```
                fprintf(1,'S_L_Left = (%d * %d) = %d\n',
btm_bit_offset_I_direction,I_L_Q_L(BPS*(j-1)+i),
horizontal_low_interp_left(BPS*(j-1)+i));
                fprintf(1,'S_L_Right = (%d * %d) = %d\n',
btm_bit_offset_I_direction_m,I_H_Q_L(BPS*(j-1)+i),
horizontal_low_interp_right(BPS*(j-1)+i));
                fprintf(1,'S_H_Left = (%d * %d) = %d\n',
btm_bit_offset_I_direction,I_L_Q_H(BPS*(j-1)+i),
horizontal_high_interp_left(BPS*(j-1)+i));
                fprintf(1,'S_H_Right = (%d * %d) = %d\n',
btm_bit_offset_I_direction_m,I_H_Q_H(BPS*(j-1)+i),
horizontal_high_interp_right(BPS*(j-1)+i));
                fprintf(1,'\nS_L = (%d + %d) =
%d\n',horizontal_low_interp_left(BPS*(j-
1)+i),horizontal_low_interp_right(BPS*(j-1)+i), horizontal_low_interp(BPS*(j-
1)+i));
                fprintf(1,'S_H = (%d + %d) =
%d\n',horizontal_high_interp_left(BPS*(j-
1)+i),horizontal_high_interp_right(BPS*(j-1)+i),
horizontal_high_interp(BPS*(j-1)+i));
            end

        end

        if (y2==y1)
            LLR(BPS*(j-1)+i) = horizontal_low_interp(BPS*(j-
1)+i)*(2^bottom_bits);
        else
            LLR(BPS*(j-1)+i) =
btm_bit_offset_Q_direction*horizontal_high_interp(BPS*(j-1)+i) +
btm_bit_offset_Q_direction_m*horizontal_low_interp(BPS*(j-1)+i);
            if (debug_flag)
                fprintf(1,'\nLLR = %d*%d +
%d*%d\n',btm_bit_offset_Q_direction,horizontal_high_interp(BPS*(j-
1)+i),btm_bit_offset_Q_direction_m,horizontal_low_interp(BPS*(j-1)+i));
                fprintf(1,'LLR = %d + %d =
%d\n',btm_bit_offset_Q_direction*horizontal_high_interp(BPS*(j-
1)+i),btm_bit_offset_Q_direction_m*horizontal_low_interp(BPS*(j-
1)+i),btm_bit_offset_Q_direction*horizontal_high_interp(BPS*(j-
1)+i)+btm_bit_offset_Q_direction_m*horizontal_low_interp(BPS*(j-1)+i));
            end
        end
        LLR(BPS*(j-1)+i) = LLR(BPS*(j-
1)+i)/((2^bottom_bits)*(2^bottom_bits));

        if (debug_flag)
            fprintf(1,'LLR(%d)=%d\n',i-1,LLR(BPS*(j-1)+i));
            fprintf(1,'----------------------------------------\n');
        end
    end
end

LLR = round(LLR);
output = LLR;
```

```
end
```

Appendix D.8 create_test_vectors.m

```
% ----------------------------------------------------------------------------
% --
% -- This software was produced for the U.S. Government under Contract
% -- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
% -- Computer Software and Noncommercial Computer Software Documentation
% -- Clause (DFARS) 252.227-7014 (JUN 1995)
% --
% -- Copyright (C) 2013 The MITRE Corporation.  ALL RIGHTS RESERVED.
% --
% -- File name: create_test_vectors.m
% -- Author   : Brian Leslie
% --
% ---------------------------------------------------------------------------

%% Description

% This script is used to generate the test vectors used when running the
% LLR Demapper component designed in VHDL to test the dataIn and dataOut.


%% Files

% In order for this script to work the following files need to be
% included in the same directory or the path to the directory where they
% can be found needs to be added

% Files:
% mapper.m

% Path and Command Window Setup
addpath ../mapper
addpath ../utilities
addpath ../demapper_algorithms
clc;
clear;
close all;

%% Code

% Local output or Server output
output_local = 0;
home_dir = 'bleslie';

% LUT Width - Number of Bits (MAX 8)
LUT_Width = 5;
bottom_bits = 8-LUT_Width;
```

```
% create test vectors until the maximum modulation.
max_BPS = 6;

% Use default seed value for random number generator so that it produces
% the same random numbers as if you restarted MATLAB.
rng('default');

% random bits -> 10^amount
amount = 5;
in=round(rand(1*10^amount,1));

%range={[], 0:.025:15, 3:.05:15, 5:.1:15, 7:.1:15};
ratio={0, 1, 1, 3.15, [2.84 5.27]};
esnovalues=[0, 0.81, 6.64, 8.8, 12.65];

% Sampled Variance for each BPS value that is used to calculate the LLR
% values
%var = [6964.5, 3477.7, 1546.8, 718.6246]; % Old
var = [6964.5, 1742.76, 832.07593, 259.93593]; % New

% Use DVB-S2 encoding on the incoming data bits.
[enc, esize, dsize]=dvbs2_enc(in,12,16,1,0);
fec_rate=dsize/esize;

% Power Level
Power_Level = 90;

% Modulations
modes={'qpsk', '8psk', '16apsk', '32apsk'};

% Test Types
testNumber={'t0', 't1', 't2'};

% Parallel Test Mode
testMode={'p2', 'p4'};

% LUT Width
LUT_Width_string={'1', '2', '3', '4', '5', '6', '7', '8'};

for testModeLoop=1:length(testMode) % Parallel Test Mode Loop
    for modesLoop=1:length(modes) % Modulations Loop
        BPS=modesLoop+1;
        fprintf(1,'BPS=%d | esno=%d\n',BPS, esnovalues(BPS));
        % Data In
        if (BPS == 2)
            Power_Level = 56*sqrt(2);
            % FEC= 1/2 -> 49*sqrt(2);
            % FEC= 5/6 (7/8) -> 56*sqrt(2)
        elseif (BPS==3)
            Power_Level = 86;
        else
            Power_Level = 90;
        end
        dataIn=mapper(enc,BPS,Power_Level,ratio{BPS});
        noisyDataIn=round(awgn(dataIn,esnovalues(BPS),'measured',0));
        noisyDataIn=complex(rail(real(noisyDataIn),-
128,127),rail(imag(noisyDataIn),-128,127));
        noise=noisyDataIn-dataIn;
        %variance = var(noise);
        variance = var(modesLoop);
        % Data Out
        dataOut=demapper_interp_llr(noisyDataIn,BPS,variance,bottom_bits);

        for testNumberLoop=1:length(testNumber)
            if (output_local)
                fprintf(1,'Created llr_dataIn/Out at: C: Users %s Documents
local_testing testVector LUT_Width_%s %s mitre_%s_%s llr_dataIn.txt\n',
home_dir, LUT_Width_string{LUT_Width}, testMode{testModeLoop},
modes{modesLoop}, testNumber{testNumberLoop});
                mat2drive(noisyDataIn,['C:\Users\' home_dir
'\Documents\local_testing\testVector\LUT_Width_' LUT_Width_string{LUT_Width}
'\' testMode{testModeLoop} '\mitre_' modes{modesLoop} '_'
testNumber{testNumberLoop} '\llr_dataIn.txt'], (testModeLoop*2), 1, 600 );
                mat2drive(dataOut,['C:\Users\' home_dir
'\Documents\local_testing\testVector\LUT_Width_' LUT_Width_string{LUT_Width}
'\' testMode{testModeLoop} '\mitre_' modes{modesLoop} '_'
testNumber{testNumberLoop} '\llr_dataOut.txt'], ((testModeLoop*2)*max_BPS),
0, 600*(modesLoop+1));
            else
                fprintf(1,'Created llr_data at: Z: designtop src components
llr tb_rtl testVector %s  mitre_ %s _ %s\n', testMode{testModeLoop},
modes{modesLoop}, testNumber{testNumberLoop});

mat2drive(noisyDataIn,['Z:\designtop\src\components\llr\tb_rtl\testVector\'
testMode{testModeLoop} '\mitre_' modes{modesLoop} '_'
testNumber{testNumberLoop} '\llr_dataIn.txt'], (testModeLoop*2), 1, 600 );

mat2drive(dataOut,['Z:\designtop\src\components\llr\tb_rtl\testVector\'
testMode{testModeLoop} '\mitre_' modes{modesLoop} '_'
testNumber{testNumberLoop} '\llr_dataOut.txt'], ((testModeLoop*2)*max_BPS),
0, 600*(modesLoop+1));
            end

        end
    end
end
```

Appendix E MITRE LLR Core

Appendix E.1 VHDL

```
------------------------------------------------------------------------------------------
-- This software was produced for the U.S. Government under Contract
-- No. FA8721-09-C-0002, and is subject to the Rights in Noncommercial
-- Computer Software and Noncommercial Computer Software Documentation
-- Clause (DFARS) 252.227-7014 (JUN 1995)
--
-- Copyright (C) 2013  The MITRE Corporation.  ALL RIGHTS RESERVED.
------------------------------------------------------------------------------------------
-- Function: This core is designed to calculate the LLR values using bi-linear
-- interpolation and Lookup tables (LUT).
-- Author : Brian Leslie
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.utility_pk.all;

entity mitre_llr_core is
  generic (
    P_DIN_W   : integer :=8;
    P_DOUT_W  : integer :=6;
    P_MAX_BPS : integer :=5;
    P_LUT_W   : integer :=5
  );
  port (
    i_clk           : in std_logic;
    i_reset         : in std_logic;
    i_bps           : in unsigned(bitwidth(P_MAX_BPS+1)-1 downto 0);

    -- i_symbol_data -> q (Top 8 Bits), i (Bottom 8 Bits)
    i_symbol_data     : in std_logic_vector(2*P_DIN_W-1 downto 0);

    i_eob_in          : in std_logic;
    i_din_valid       : in std_logic;
    o_din_rdy         : out std_logic;

    o_llr_data        : out std_logic_vector(P_MAX_BPS*P_DOUT_W-1 downto 0);
    o_eob_out         : out std_logic;
    o_dout_valid      : out std_logic;
    i_dout_rdy        : in std_logic;
```

```
     i_data           : in std_logic_vector(P_MAX_BPS*P_DOUT_W-1 downto 0);
     i_addr           : in std_logic_vector(2*P_LUT_W-1 downto 0);
     i_wr             : in std_logic


  );
end mitre_llr_core;

architecture synth of mitre_llr_core is

  -- Constant Declarations
  constant C_PIPELINE_DELAY : integer := 7;
  constant C_R_W            : integer := P_DIN_W-P_LUT_W+2;
  constant C_LLR_W          : integer := (P_DOUT_W+(2*(P_DIN_W-P_LUT_W+2)));
  constant C_LLR_VECTOR_W   : integer := ((P_DOUT_W+(2*(P_DIN_W-
P_LUT_W+2)))*P_MAX_BPS);

   -- RAM
  type ram_t is array(natural range <>) of unsigned(P_MAX_BPS*P_DOUT_W-1 downto 0);
  type ram_tt is array(natural range<>) of ram_t(2**(2*P_LUT_W)-1 downto 0);

  -- LLR
  type llr_t is array(natural range <>) of signed(P_DOUT_W-1 downto 0);
  type llr_round_t is array(natural range <>) of signed(C_LLR_W-C_R_W-1 downto 0);
  type llr_round_large_t is array(natural range <>) of signed(C_LLR_W-1 downto 0);

   -- RAM
  signal ram : ram_tt(3 downto 0);

  -- Four cordinates on the I/Q plane
  signal i_l            : unsigned(P_LUT_W-1 downto 0);
  signal i_h            : unsigned(P_LUT_W-1 downto 0);
  signal q_l            : unsigned(P_LUT_W-1 downto 0);
  signal q_h            : unsigned(P_LUT_W-1 downto 0);

  -- Remainder of I/Q Data bits not used for the Four LUT Cordinates
  signal r_i            : signed(C_R_W-1 downto 0);
  signal r_i_delay      : signed(C_R_W-1 downto 0);
  signal r_q            : signed(C_R_W-1 downto 0);
  signal r_q_delay      : signed(C_R_W-1 downto 0);
  signal r_q_delay_two   : signed(C_R_W-1 downto 0);
  signal r_q_delay_three : signed(C_R_W-1 downto 0);
  signal r_i_minus_delay : signed(C_R_W-1 downto 0); -- P_DIN_W - r_i_delay
  signal r_q_minus_delay : signed(C_R_W-1 downto 0); -- P_DIN_W - r_q_delay
  -- Four points using the cordinates for Bilinear Interpolation
  signal q_low_i_low    : llr_t(P_MAX_BPS-1 downto 0);
  signal q_high_i_low   : llr_t(P_MAX_BPS-1 downto 0);
  signal q_low_i_high   : llr_t(P_MAX_BPS-1 downto 0);
  signal q_high_i_high  : llr_t(P_MAX_BPS-1 downto 0);

  -- Linear Interpolation in the i-direction (x-direction)
  signal s_h            : llr_round_t(P_MAX_BPS-1 downto 0);  -- upper interpolation
  signal s_l            : llr_round_t(P_MAX_BPS-1 downto 0);  -- lower interpolation
  signal s_h_left       : llr_round_t(P_MAX_BPS-1 downto 0);
  signal s_h_right      : llr_round_t(P_MAX_BPS-1 downto 0);
  signal s_l_left       : llr_round_t(P_MAX_BPS-1 downto 0);
  signal s_l_right      : llr_round_t(P_MAX_BPS-1 downto 0);

  -- Pipeline for Tracking Data Validity
  signal valid_pipeline : std_logic_vector(C_PIPELINE_DELAY downto 0);

  -- Pipeline for Tracking End of Block Flag
  signal eob_pipeline   : std_logic_vector(C_PIPELINE_DELAY downto 0);

  --  Rounding Implemenation
  signal stall          : std_logic;
  signal llr_pad        : llr_round_large_t(P_MAX_BPS-1 downto 0);
  signal llr_pad_left   : llr_round_large_t(P_MAX_BPS-1 downto 0);
  signal llr_pad_right  : llr_round_large_t(P_MAX_BPS-1 downto 0);
  signal llr_pad_flat   : std_logic_vector(C_LLR_VECTOR_W-1 downto 0);

begin

  ---------------------------------------------------------------------------
  -- Load Lookup Table Values into RAM
  ---------------------------------------------------------------------------
  sRAMPrc: process(i_clk)
  begin
    if (rising_edge(i_clk)) then
      if (i_wr = '1') then
        for i in 0 to 3 loop
          ram(i)(to_integer(unsigned(i_addr))) <= unsigned(i_data(P_MAX_BPS*P_DOUT_W-1
downto 0));
        end loop;
      end if;
    end if;
  end process;

  ---------------------------------------------------------------------------
```

```
 -- Calculate LLR Values and Load them into output buffer
 -----------------------------------------------------------------------------
 gen_LLR_values: process(i_clk)
 begin
  if (rising_edge(i_clk)) then
   if (i_reset = '1') then

    -- Set Valid and EOB Pipelines on Reset
    valid_pipeline <= (others => '0');
    eob_pipeline <= (others => '0');

   elsif (i_dout_rdy = '1') then

    -- Shift Valid Data Pipeline Values and Load the next Value from
    -- the i_din_valid Input Port
    valid_pipeline(valid_pipeline'high downto 1) <= valid_pipeline(valid_pipeline'high-1
downto 0);
    valid_pipeline(0) <= i_din_valid;

    -- Shift End of Block Flag Pipeline Values and Load the next Value from
    -- the i_din_valid Input Port
    eob_pipeline(eob_pipeline'high downto 1) <= eob_pipeline(eob_pipeline'high-1 downto 0);
    eob_pipeline(0) <= i_eob_in;

    -- Define 4 cordinate points on I/Q plane for interpolation
    i_l <= unsigned(i_symbol_data(P_DIN_W-1 downto P_DIN_W-P_LUT_W));
    if (unsigned(i_symbol_data(P_DIN_W-1 downto P_DIN_W-P_LUT_W)) =
2**(P_LUT_W-1)-1) then
     i_h <= unsigned(i_symbol_data(P_DIN_W-1 downto P_DIN_W-P_LUT_W));
    else
     i_h <= unsigned(i_symbol_data(P_DIN_W-1 downto P_DIN_W-P_LUT_W))+1;
    end if;

    q_l <= unsigned(i_symbol_data((2*P_DIN_W)-1 downto (2*P_DIN_W)-P_LUT_W));
    if ( unsigned(i_symbol_data((2*P_DIN_W)-1 downto (2*P_DIN_W)-P_LUT_W)) =
2**(P_LUT_W-1)-1) then
     q_h <= unsigned(i_symbol_data((2*P_DIN_W)-1 downto (2*P_DIN_W)-P_LUT_W));
    else
     q_h <= unsigned(i_symbol_data((2*P_DIN_W)-1 downto (2*P_DIN_W)-P_LUT_W))+1;
    end if;

    -- Determine remainder of I/Q data to be used for offset in the interpolation
    r_i <= signed("00" & i_symbol_data(P_DIN_W-P_LUT_W-1 downto 0));
    r_q <= signed("00" & i_symbol_data((2*P_DIN_W)-P_LUT_W-1 downto P_DIN_W));
    -- Signal Delays to Allign the Data Output
    r_i_delay <= r_i;
    r_q_delay <= r_q;
    r_q_delay_two <= r_q_delay;
    r_q_delay_three <= r_q_delay_two;
    r_i_minus_delay <= (2**(P_DIN_W-P_LUT_W))-r_i;
    r_q_minus_delay <= (2**(P_DIN_W-P_LUT_W))-r_q_delay_two;

    -- Calculate LLR for each bit in the symbol
    for i in P_MAX_BPS-1 downto 0 loop

     -- Load LLR values from the LUT in RAM
     q_low_i_low(i) <= signed(ram(0)(to_integer(q_l&i_l))(P_DOUT_W*(i+1)-1 downto
P_DOUT_W*i));
     q_high_i_low(i) <= signed(ram(1)(to_integer(q_h&i_l))(P_DOUT_W*(i+1)-1 downto
P_DOUT_W*i));
     q_low_i_high(i) <= signed(ram(2)(to_integer(q_l&i_h))(P_DOUT_W*(i+1)-1 downto
P_DOUT_W*i));
     q_high_i_high(i) <= signed(ram(3)(to_integer(q_h&i_h))(P_DOUT_W*(i+1)-1 downto
P_DOUT_W*i));

     -- Lower Horizontal Linear Interpolation
     s_l_left(i) <= q_low_i_high(i)*r_i_delay;
     s_l_right(i) <= q_low_i_low(i)*r_i_minus_delay;
     s_l(i) <= s_l_left(i)+s_l_right(i);

     -- Upper Horizontal Linear Interpolation
     s_h_left(i) <= q_high_i_high(i)*r_i_delay;
     s_h_right(i) <= q_high_i_low(i)*r_i_minus_delay;
     s_h(i) <= s_h_left(i)+s_h_right(i);

     -- Vertical Linear Interpolation
     llr_pad_left(i) <= s_h(i)*r_q_delay_three;
     llr_pad_right(i) <= s_l(i)*r_q_minus_delay;
     llr_pad(i) <= llr_pad_left(i)+llr_pad_right(i);

    end loop;
   end if;
  end if;
 end process;
 ----------------------------------------------------------------------------
 -- Set ready pipeline output to input value
 ----------------------------------------------------------------------------
 o_din_rdy <= i_dout_rdy;
```

```
  -----------------------------------------------------------------------------
  -- stall signal definition to the opposite of i_dout_rdy
  -----------------------------------------------------------------------------
  stall <= not(i_dout_rdy);

  -----------------------------------------------------------------------------
  -- Rounding Entity
  -----------------------------------------------------------------------------
  round: entity work.bitQuantize
    generic map(
      DIN_W     => (P_DOUT_W+2*(2+(P_DIN_W-P_LUT_W))),
      DOUT_W    => P_DOUT_W,
      SHIFT     => (2*(P_DIN_W-P_LUT_W)),
      COUNT     => P_MAX_BPS,
      Q_MODE    => 1,
      V_MODE    => 2
    )
    port map(
        clk       => i_clk,
        reset     => i_reset,
        stall     => stall,
        din       => llr_pad_flat,
        dout      => o_llr_data,
        started   => open,
        overflow  => open
     );

  -----------------------------------------------------------------------------
  -- Load Rounding Buffer
  -----------------------------------------------------------------------------
  load_round_bus: for i in 0 to P_MAX_BPS-1 generate
    llr_pad_flat(((C_LLR_W)*(i+1)-1) downto (C_LLR_W*i)) <= std_logic_vector(llr_pad(i));
  end generate load_round_bus;

  -----------------------------------------------------------------------------
  -- Load Valid Data Flag into output port
  -----------------------------------------------------------------------------
  o_dout_valid <= valid_pipeline(valid_pipeline'high);

  -----------------------------------------------------------------------------
  -- Load End of Block Flag into output port
  -----------------------------------------------------------------------------
  o_eob_out <= eob_pipeline(eob_pipeline'high);

end synth;
```