University of Rhode Island

DigitalCommons@URI

Open Access Master's Theses

2015

Low Power Underwater Acoustic DPSK Detection: Theoretical Prediction and Experimental Results

Andrew Dunne

University of Rhode Island, andrew_dunne@my.uri.edu

Follow this and additional works at: <https://digitalcommons.uri.edu/theses>

Recommended Citation

Dunne, Andrew, "Low Power Underwater Acoustic DPSK Detection: Theoretical Prediction and Experimental Results" (2015). *Open Access Master's Theses*. Paper 791.
<https://digitalcommons.uri.edu/theses/791>

This Thesis is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Master's Theses by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons-group@uri.edu.

LOW POWER UNDERWATER ACOUSTIC DPSK DETECTION:
THEORETICAL PREDICTION AND EXPERIMENTAL RESULTS

BY

ANDREW DUNNE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2015

MASTER OF SCIENCE THESIS

OF

ANDREW DUNNE

APPROVED:

Thesis Committee:

Major Professor

Godi Fischer

Co-major Professor

Harold T. Vincent II

Peter F. Swaszek

James Hu

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2015

ABSTRACT

This thesis presents two methods of analyzing the effectiveness of a prototype differential phase-shift keying (DPSK) detection circuit. The first method is to make modifications to the existing hardware to reliably output and record the cross-correlation values of the DPSK detection process. The second method is to write a MATLAB detection algorithm which accurately simulates the detection results of the hardware system without the need of any electronics. These two systems were built, tested and verified with a bench test using computer generated DPSK signals. The hardware system was tested using real acoustic data from shallow and deep water at-sea tests to determine the effectiveness of the DPSK detection circuit in different ocean environments. The hydrophone signals from these tests were recorded so that the cross-correlation values could be verified using the MATLAB detector. As a result of this study, these two systems provided more insight into how well the DPSK detection prototype works and helped to identify ways of improving the detection reliability and overall performance of the DPSK detection circuit.

ACKNOWLEDGMENTS

I would like to take a moment to thank a few important people who helped me throughout this study. First, I would like to thank my Major Professor, Dr. Godi Fischer, who was always quick to help, but also allowed me the freedom needed to complete my research independently and further my skills in the engineering field. I also want to thank my Co-major Professor and employer, Dr. Harold T. Vincent II, who provided me with valuable on the job experiences and the opportunity to travel in order to collect valuable data for this thesis.

I would also like to thank other personnel from the University of Rhode Island who supported and assisted me throughout my research. Cathy Cipolla, for her advice which helped motivate me to complete my degree. Gary Savoie and Jeremiah Sullivan, for their assistance in the building, deploying, and recovering the mechanical systems used during at-sea testing. I would also like to thank Hayden Radke who assisted me in many ways throughout this study, showing incredible patience as I discussed my thesis with him.

Most importantly, I would like to thank my family. My brothers, Jonathan, Matthew, and Michael, provided perhaps the most motivation to earn my masters degree by teasing that I was the "least intelligent of the four." They also reminded me to take breaks from my research, to unwind and enjoy myself. Finally, I would like to thank my parents who taught me, at a very early age, the importance of education and perseverance. These are just two of the many important values that they instilled

in me throughout my life. They always encouraged me to be the best version of myself that I could be and continue to do so today. Their support and encouragement were invaluable in helping motivate me throughout this research.

TABLE OF CONTENTS

Abstract	ii
Acknowledgments	iii
Table of Contents.....	v
List of Tables	vii
List of Figures	viii
1 Introduction	1
2 Background	4
2.1 Phase-Shift Keying	4
2.2 Differential Phase-Shift Keying	8
2.3 Differential Binary Phase-Shift Keying	10
2.4 Gold Codes	12
2.5 Ocean Noise	17
3 Methods	20
3.1 Overview of Test Plan	20
3.2 Generating DPSK Codes	20
3.3 MATLAB DPSK Detection	23
3.4 Hardware Detector Modifications	27
3.4.1 DPSK PCB Modifications	28
3.4.2 Recording System	31
3.4.3 MATLAB Code	32
3.5 At-Sea Tests	33
3.5.1 Narragansett Bay Test	33
3.5.2 Bermuda Test	37

4	Results	42
4.1	Bench Test: Simulation and Hardware Results	42
4.1.1	PMF Bench Test Results.....	43
4.1.2	ROC Curve Bench Test Results	49
4.1.3	Improving the Bench Test Results	51
4.2	At-Sea Test Results	53
4.2.1	Narragansett Bay Test Results	53
4.2.2	Bermuda Test Results	58
5	Conclusion	64
	Appendix A: MATLAB Code	67
	Appendix B: VHDL Code	82
	Appendix C: C-Sharp Code.....	88
	Bibliography	89

LIST OF TABLES

2.1	Bandwidth efficiency of M-ary PSK signals	6
2.2	Comparison of power-bandwidth requirements of MPSK with BPSK. Probability of symbol error= 10^{-4} and identical noise environments	7
2.3	An example of the encoding process for DBPSK	10
2.4	Primitive polynomials	15
4.1	Wav File Descriptions	42

LIST OF FIGURES

1.1	DBV Technology's DPSK detection PCB	2
2.1	An example of a coherent BPSK signal for the data stream {10110}	5
2.2	Probability of symbol error versus SNR for MPSK (solid line) and DMPSK (dotted line) modulation schemes	9
2.3	DBPSK modulator	11
2.4	DBPSK demodulator	11
2.5	Simple shift register generator (SSRG)	13
2.6	Four-stage LFSR and its cycles	14
2.7	Four-stage LFSR maximal length sequence	14
2.8	Gold code generator using m-sequences {45} and {67}	16
2.9	Examples of auto-correlation (top) and cross-correlation of Gold codes	17
2.10	Multipath trajectories (top) and the corresponding time-domain signal followed by a series of echoes (bottom)	18
2.11	Wenz Curve representing deep-ocean ambient noise characteristics	19
3.1	Sample of the DPSK signal (bottom) generated in MATLAB from a Gold code binary sequence (top)	22
3.2	The same DPSK signal generated by the MATLAB code with varying SNR levels. Signal amplitude is 20mV	22
3.3	Hardware block diagram of the DPSK detector circuit	23
3.4	Simulated results of the front end of the DPSK detection circuit. SNR = 9dB	24
3.5	Software demodulation process. Comparator output (top), delayed comparator output (middle), and demodulated signal (bottom) resulting from the XNOR operation	25
3.6	Examples of the simulated cross-correlation process. (a) SNR = 12dB, (b) SNR = 0dB, (c) SNR = -6dB	27

3.7	Block diagram of the Hardware setup	28
3.8	Modifications to the DPSK detection board. 1) Altera's CPLD. 2) Extra GPIOs for hardware expansion. 3) Programming header for the CPLD	29
3.9	VHDL Block Diagrams	30
3.10	Oscilloscope capture of one cross-correlation package	31
3.11	Oscilloscope capture of multiple serial data packets	31
3.12	DPSK signal and cross-correlation data from a hardware bench test	32
3.13	DPSK signal and cross-correlation values for one transmitted signal	33
3.14	Electronics stack designed for the Narragansett Bay test	34
3.15	PVC pressure housing fully assembled	35
3.16	Pressure housing end cap with shorting plug and hydrophone cable	36
3.17	Both pressure housings mounted and ready for deployment	36
3.18	Electronics inside of glass pressure housing	38
3.19	Block diagram of transmitting surface buoy (cross section)	39
3.20	Surface buoy. From top left: tonpilz transducer, inflatable ring, transmitter electronics	40
3.21	Recording system pressure vessel inside of an orange hard hat for protection and easier handling	41
4.1	PMFs of the MATLAB detector cross-correlation values	46
4.2	PMFs of the hardware detector cross-correlation values	46
4.3	Comparison of MATLAB and hardware cross-correlation PMFs for an ideal signal and SNR of 12dB and 9dB	47
4.4	Comparison of MATLAB and hardware cross-correlation PMFs for SNR of 6dB, 3dB, and pure noise	47
4.5	A comparison of the cross-correlation results over time	48

4.6	PMF comparisons of Code1, Code2 and computer generated white Gaussian noise	49
4.7	MATLAB ROC curves at various SNR levels	50
4.8	Hardware ROC curves at various SNR levels	51
4.9	PMFs of MATLAB detector results using an improved band-pass filter	52
4.10	Hydrophone signal, preamplified signal, and cross-correlation results from the second Narragansett Bay test	54
4.11	Hydrophone signal, preamplified signal, and cross-correlation results from the second Narragansett Bay test zoomed in to show one seconds worth of data	54
4.12	MATLAB detection results versus hardware detection in 20 feet of water	55
4.13	Preamplified hydrophone signal and cross-correlation results for a shallow water test in 30 feet of water	56
4.14	Cross-correlation results from transmitting code 2 in 20 feet of water	57
4.15	Spectrogram of the Bermuda Test	58
4.16	Spectrogram zoomed in and filtered to show DPSK signals	59
4.17	Power spectral density of Bermuda noise	60
4.18	PMFs of MATLAB detector cross-correlation values using Bermuda noise	60
4.19	PMFs of hardware detector cross-correlation values using Bermuda noise	61
4.20	Comparison of MATLAB and hardware cross-correlation PMFs for an Ideal signal and SNR of 12dB and 9dB with Bermuda noise	61
4.21	Comparison of MATLAB and hardware cross-correlation PMFs for SNR of 6dB, 3dB and pure Bermuda noise	62
4.22	MATLAB ROC curves at various SNR levels using Bermuda noise	62
4.23	Hardware ROC curves at various SNR levels using Bermuda noise	63

CHAPTER 1

INTRODUCTION

The differential phase-shift keying (DPSK) detection circuit is a compact low-power printed circuit board (PCB) which continuously demodulates acoustic signals in the ocean environment and cross-correlates the results with stored binary sequences. Besides the use of an optional preamplifier and a comparator IC, all the DPSK demodulation and cross-correlation algorithms are implemented on a complex programmable logic device (CPLD). Originally designed by DBV Technology, LLC., the primary purpose of the DPSK detection circuit is an underwater acoustic retrieval system to aid in salvage missions and the recovery of oceanographic instruments. For example, the DPSK detection circuit can be integrated into a pressure housing containing other oceanographic equipment such as an acoustic Doppler current profiler (APDC). Using a hydrophone as an input signal, the DPSK detection circuit will continuously cross-correlate the demodulated input signal and compare it against a stored binary sequence on the CPLD. When it is time to recover the instrument, a DPSK signal of the modulated retrieval code will be transmitted from the surface. If properly demodulated, the DPSK detection circuit will recognize the transmitted signal as a retrieval command and the recovery process will begin.

Since this system is designed to support multiple year deployments it is of utmost importance for the detection process to have a low false detection rate. If this system cannot properly differentiate between different transmitted sequences as well the ambient noise of the underwater environment it could result in the loss of

scientific instruments and corresponding data, or worse, pose a threat to the safety of salvage/research divers. There are ways to minimize the chances of false detection. In the case where DPSK signals are used in command and control applications such as this one, it is more important to have a low false detection rate than a high detection rate. Setting a high detection threshold on the cross-correlation process will reduce the chance of a false detection event. The trade-off is that setting the threshold too high when there is signal distortion introduced by the ocean environment may make it impossible to retrieve objects on the ocean floor. For this reason, it is desired to determine the best balance between a missed detection and a false alarm.

Figure 1.1: DBV Technology's DPSK detection PCB

Prior to this study, the DPSK detection prototype has been tested by adjusting the cross-correlation threshold and counting the number of positive detections for every transmitted code. This process is time consuming and produces a true or false outcome with no real insight into how well the detection process is working. Having

access to the real time cross-correlation results would provide much more valuable information. The focus of this research is to design and build a hardware system to collect the cross-correlation data from the DPSK detection circuit and compare the results to a computer simulation of the detection process. In addition, most of the testing up to this point has been done in ideal acoustic conditions, not representative to the real ocean environments that the detector is designed for. It is of interest to test how well this device operates in both shallow and deep water and to document these results. The end goal is to provide methods for determining the best detection threshold to minimize the false detection rate while maintaining the best possible detection probability.

CHAPTER 2

BACKGROUND

2.1 Phase-Shift Keying

The acoustic DPSK detection circuit uses a type of digital modulation scheme known as differential phase-shift keying (DPSK). Before discussing this modulation scheme, it is beneficial to develop an understanding of the general modulation class of phase-shift keying (PSK) and why it was chosen over other types of modulation techniques. In general, PSK is an envelope term used to describe a modulation scheme where discrete phase shifts of a carrier frequency represent different binary values. To keep things simple, this discussion will focus primarily on binary PSK (BPSK). In BPSK, binary data are represented by two signals with different phases [18]. These phases are usually 0 and π radians and equally spaced. The signals can be represented as the following:

$$s_1(t) = A\cos(2\pi f_c t) \quad 0 \leq t \leq T, \quad \text{for } 1 \quad (2.1)$$

$$s_2(t) = A\cos(2\pi f_c t + \pi) = -A\cos(2\pi f_c t), \quad 0 \leq t \leq T, \quad \text{for } 0 \quad (2.2)$$

Where A is the signal amplitude and f_c is the carrier frequency. Each signal has the same frequency and energy. As the above equations show, the signal with no phase change is used to represent a binary value of 1 and the signal with a phase change is used to represent a binary value of 0. The effect of these signals on the overall structure of the modulated waveform can be seen in Figure 2.1.

Figure 2.1: An example of a coherent BPSK signal for the data stream {10110} [18].

As expected from equations (2.1) and (2.2), the amplitude or energy of the signal is constant. The frequency is also consistent however there are discontinuities between the bit boundaries, where the data bit values change. In order to decode the BPSK signal, the demodulation circuit must compare the incoming signal with a reference signal which must be synchronized to the carrier signal in both frequency and phase.

BPSK is just one special case of M-ary PSK (MPSK) where the order $M = 2$. Other more complex PSK modulation schemes use a larger number of discrete phase shifts to effectively increase the data bandwidth. The most popular of the MPSK schemes is Quadrature PSK (QPSK), where $M = 4$. In this case, four discrete phases are used to represent a pair of binary values also called dibits. For example phases of $\pi/4$, $3\pi/4$, $-3\pi/4$, and $-\pi/4$ of a constant carrier frequency will represent binary values of {11}, {01}, {00}, and {10} respectively. Since every phase shift now represents two bits rather than one, the bandwidth is doubled.

There are tradeoffs for the increase in bandwidth that the higher order M-ary PSK schemes provide. First, let's look at the effects of the symbol error as the order,

M, increases. The average probability of symbol error for coherent M-ary PSK can be represented as the following complementary error function:

$$P_s \cong \text{erfc} \left(\sqrt{\frac{E}{N_0}} \sin(\pi/M) \right) \quad (2.3)$$

Where E, is the signal energy, $N_0/2$ is the noise spectral density, and M is the order of the MPSK modulation scheme. This equation for coherent M-ary PSK signals show that the BER will increase not only as the SNR decreases, but also as M increases meaning that higher order modulation schemes increase the chances of incorrect bits of information. Next we look at the bandwidth efficiency of the M-ary PSK signal which is represented by the equation:

$$\rho = \frac{R_b}{B}$$

where R_b is the bit rate and the channel bandwidth (or the main spectral lobe of the M-ary signal) is given by:

$$B = \frac{2R_b}{\log_2 M}$$

therefore the bandwidth efficiency can be simplified to:

$$\rho = \frac{\log_2 M}{2} \quad (2.4)$$

Value of M	2	4	8	16	32	64
ρ (bits/s/Hz)	0.5	1	1.5	2	2.5	3

Table 2.1: Bandwidth efficiency of M-ary PSK signals [5]

The table above shows the bandwidth efficiency of various M-ary PSK schemes by plugging in values of M into equation (2.4). What is clear from this table is that the efficiency increases at a much slower rate than the order of the MPSK scheme. When designing a PSK modulation system, one must consider if the gain in bandwidth efficiency is worth the exponential increase in circuit design complexity introduced by these higher order systems. In addition, as equation (2.3) shows, for a higher order system to achieve an equal average symbol error probability as a low order system, the received signal energy must be increased to raise the SNR of the system. Table 2.2 shows these effects of M-ary PSK in comparison to a BPSK. In the table, the average probability of a symbol error is equal to 10^{-4} and each of the systems are operating in identical noise environments.

Value of M	$\frac{(\text{Bandwidth})_{M\text{-ary}}}{(\text{Bandwidth})_{\text{Binary}}}$	$\frac{(\text{Average Power})_{M\text{-ary}}}{(\text{Average Power})_{\text{Binary}}}$
4	0.5	0.34 dB
8	0.333	3.91 dB
16	0.25	8.52 dB
32	0.2	13.52 dB

Table 2.2: Comparison of power-bandwidth requirements for MPSK with BPSK. Probability of symbol error= 10^{-4} and identical noise environments [12]

From the table it is apparent that QPSK (M = 4) provides the best trade off of bandwidth for power. Because of this, QPSK is often the most commonly used MPSK modulation scheme. In practice, any values of M > 8 are usually not as common

because of these tradeoffs as well as the increased BER and added complexity in circuit design.

2.2 Differential Phase-Shift Keying

The types of MPSK schemes discussed up to this point all fall under the category of coherent PSK modulation schemes. The DPSK detector prototype developed by DBV Technology uses a non-coherent form of phase-shift keying known as differential phase-shift keying (DPSK). This modulation technique eliminates the need for a coherent reference signal at the receiver meaning that the transmitter and receiver no longer have to be synced up. Not needing to know the starting phase of the source is a huge advantage of DPSK in underwater applications where synchronization of the transmitting source on the surface to the receiver in the ocean can be very challenging. DPSK follows the same basic rules of PSK discussed in section 2.1 but causes a slight increase in symbol error probability [4].

Figure 2.2: Probability of symbol error versus SNR for MPSK (solid line) and DMPSK (dotted line) modulation schemes. [18].

The solid lines in figure 2.2 are good representations of equation (2.3) discussed earlier. As expected, the SNR must be increased for the higher orders to keep the symbol error probability constant. The dotted lines represent the DMPSK counterparts of the MPSK schemes. From this figure, it is clear that DPSK will always yield a higher chance of an error occurring. Interestingly, differential binary phase-shift keying (DBPSK) shows a minimal change in symbol error probability over its

MPSK counterpart, BPSK. DBPSK is the modulation scheme used by the DBV Technology's prototype DPSK detector.

2.3 Differential Binary Phase-Shift Keying

DBPSK eliminates the need of a coherent reference signal by first encoding the binary signal on the transmit side before modulation and then comparing the previous bit sample to the current bit sample to determine the binary value on the receiver side. The encoding process can be expressed by the equation:

$$d_k = \overline{a_k \oplus d_{k-1}}$$

where a_k is the current bit of the binary code to be transmitted while d_k and d_{k-1} are the current and previous differentially encoded bits. The value of d_k determines the transmitted phase. Table 2.3 helps to visualize this encoding process. In this example, the binary sequence to be transmitted is {1 0 0 1 0 0 1 1}. From the table, note that every time a_k is 1, the current transmitted phase and previously transmitted phase are the same. Every time a_k is 0, the current phase and previous phase are π radians apart.

a_k	1	0	0	1	0	0	1	1
d_{k-1}	1	1	0	1	1	0	1	1
Differentially encoded Sequence, d_k	1	1	0	1	1	0	1	1
Transmitted Phase	0	0	π	0	0	π	0	0

Table 2.3: An example of the encoding process for DBPSK [5]

The block diagram for the differential binary encoding process is shown in figure 2.3.

Figure 2.3: DBPSK modulator [18].

The DBPSK signal is also represented by the following equations:

$$\xi_1(t) = \begin{cases} A \cos(2\pi f_c t), & 0 \leq t \leq T \\ A \cos(2\pi f_c t), & T \leq t \leq 2T \end{cases}, \quad \text{for } 1 \quad (2.5)$$

$$\xi_2(t) = \begin{cases} A \cos(2\pi f_c t), & 0 \leq t \leq T \\ -A \cos(2\pi f_c t), & T \leq t \leq 2T \end{cases}, \quad \text{for } 0 \quad (2.6)$$

Where, again, A is the signal amplitude and f_c is the carrier frequency. When comparing this pair of equations with equation (2.1) and (2.2) it is easy to see that the actual phase of the signal is no longer what determines the bit value in the message. Instead, if the current and previous signals are in phase with each other the binary value is '1' and if the two are out of phase, the binary value is '0.' Now, instead of using a reference frequency in the demodulation process, a delay of one bit cycle is implemented on the receiver side allowing for a comparison between the previous and current bit. The outcome of this comparison determines the value of the sent binary symbol. A block diagram of the electronics for a standard DBPSK receiver is shown in figure 2.4.

Figure 2.4: DBPSK demodulator [18]

For more information regarding the phase-shift keying modulation schemes or any in depth derivations of the equations from the previous sections, it is recommended to read chapter 4 of *Digital Modulation Techniques* [18] as well as chapter 6 from *Digital Communications* [5]. Both of these sources are listed in the bibliography section of this paper.

2.4 Gold Codes

One desired ability of the DPSK Detector is being able to deploy multiple systems at once and communicate with each one individually. An advantage of this is the ability to retrieve individual systems when multiple devices are within listening range of the transmitted recovery signal. To achieve this, a set of Gold codes are used to individually address each system. Gold codes (or Gold sequences) are special binary sequences that yield the theoretically minimum cross-correlation values that one can possibly expect when compared to other sequences from the same set [11]. The idea of Gold codes was first published in 1967 by Robert Gold as a way to prevent spread spectrum communication systems in multiplexing applications from interfering with each other. In this case, each communication link would employ a different maximal encoding sequence. When different systems operated in the same environment one communication link would interfere with another when a receiver would lock onto the cross-correlation peaks obtained by correlating with the encoding sequence of a different communication link. In general, the cross-correlation function between different maximal sequences may be relatively large making this a common problem [3].

Gold codes are generated by combining maximal length sequences obtained from linear feedback shift registers (LFSR). One standard form of a binary LFSR known as the Fibonacci implementation or a simple shift register generator (SSRG) is shown in figure 2.5. The shift register consists of binary storage elements (boxes) which transfer their contents to the right after each clock pulse. The contents of the registers are linearly combined with the binary (0,1) coefficients a_k and are fed back into the first stage. The periodic cycle of the state depends on the initial state and on the coefficients (feedback taps) a_k [10].

Figure 2.5: Simple shift register generator (SSRG) [10]

The outputs of these LFSRs are considered pseudorandom sequences. The binary value associated with each feedback coefficients will determine the length and quantity of each periodic cycle produced. For example, Figure 2.6 shows a 4 stage LFSR where all the feedback coefficients are set to binary '1'. In this case, the LFSR has four pseudorandom cycles, the three shown as well as a cycle of all zeros when each stage is preloaded with binary value '0'.

Figure 2.6: Four-stage LFSR and its cycles [Pickholtz1982, Figure 5]

Figure 2.7: Four-stage LFSR maximal length sequence [10]

Maximal length sequences are sequences generated from an LFSR with period $2^r - 1$. For example, a 4 stage LFSR will have two possible cycles. One cycle of all zeros as is common to all LFSRs and one pseudorandom sequence of length 15. This useful pseudorandom sequence is the maximal length sequence and is generated by properly choosing the feedback coefficients [10]. Figure 2.7 is an example of a LFSR which generates a maximal length sequence. It can also be represented in polynomial form as:

$$X^4 + X^3 + 1$$

which can also be written in octal form {31}. Finding the primitive polynomials for each LFSR of any size will yield the maximal length sequences needed to generate

gold codes. All the primitive polynomials used to generated maximal length sequences for up an LFSR of 8 degrees is listed in table 2.4. This table does not include the reciprocals of the primitive polynomials, but those will also generate maximal length sequences. The table shows that {23} is a primitive polynomial which can be written as:

$$X^4 + X + 1$$

Therefore since the previous example from figure 2.7 is a reciprocal of this primitive polynomial, it does indeed produce a maximal length sequence. It is always possible to choose the feedback coefficients so as to achieve maximal length [13].

Degree	Primitive polynomials
2	$\langle 7 \rangle$
3	$\langle 13 \rangle$
4	$\langle 23 \rangle$
5	$\langle 45 \rangle, \langle 75 \rangle, \langle 67 \rangle$
6	$\langle 103 \rangle, \langle 147 \rangle, \langle 155 \rangle$
7	$\langle 211 \rangle, \langle 217 \rangle, \langle 235 \rangle, \langle 367 \rangle, \langle 277 \rangle, \langle 325 \rangle, \langle 203 \rangle, \langle 313 \rangle, \langle 345 \rangle$
8	$\langle 435 \rangle, \langle 551 \rangle, \langle 747 \rangle, \langle 453 \rangle, \langle 545 \rangle, \langle 537 \rangle, \langle 703 \rangle, \langle 543 \rangle$

Table 2.4: Primitive polynomials [13]

Gold codes are generated by the exclusive or-ing (or modulo-2 adding) of two maximal length sequences with the same length as shown in figure 2.8. every change in phase position between the two generated maximal length sequences causes a new Gold sequence to be generated. Any 2-register Gold code generator of length, r

(the length of the individual LFSRs) can generate 2^r-1 sequences (of length 2^r-1) plus the two m-sequences, giving a total of 2^r+1 sequences [9],[2].

Figure 2.8: Gold code generator using m-sequences {45} and {67} [2]

If the pair of maximal length sequences are chosen correctly, the cross-correlation values between each Gold code is guaranteed to be bounded and minimized. In the case above, where the length of the register is five, the cross-correlation will never be greater than 7 and less than -9. These predictable cross-correlation properties make it possible to set a correlation threshold so that only the correct sequence will be detected, a useful characteristic to have for the DPSK detection prototype. Figure 2.9 helps to visualize this concept where the auto-correlation and cross-correlation gaps are maximized.

Figure 2.9: Examples of auto-correlation (top) and cross-correlation of Gold codes [9]

2.5 Ocean Noise

As explained in the introduction, it is of interest to see how well the DPSK detection circuit operates in different ocean environments. This research includes gathering data from both a shallow water environment and a deep ocean environment, however this is a bit of an oversimplification of the problem considering there is a vast amount of factors that could degrade or interfere with

signal detection. In shallow water, it is expected that multipath will be one of the main causes of bit errors in the DPSK demodulation process. Multipath is the term used when a given signal can propagate from a source to a receiver along many different paths due to successive reflections at the surface and seabed interfaces. This will cause a direct signal to be received followed by a series of echoes (Figure 2.10).

Figure 2.10: Multipath trajectories (top) and the corresponding time-domain signal followed by a series of echoes (bottom) [7]

Depending on the depth of the water, positioning of hydrophone in the water column, and length of the transmitted signal, the echoes could overlap the desired signal (such as B, C, and D) causing detection issues [7]. In shallow water, noise levels could be very high due to heavy shipping, nearby surf, higher biological noise, shore-based noises, off-shore drilling rigs, and more making detection of the DPSK signals challenging [6].

Deep ocean noise is much easier to model but also has its own set of challenges to overcome. The Wenz Curve (Figure 2.11) is a generalized model of the deep-water ambient noise. This model, which was measured using omnidirectional receivers, shows that noise sources below 500Hz are dominated by biological noise and shipping traffic. Above 500Hz, the strongest source of ambient noise comes from local wind speeds and the sea state [6]. Another challenge with deep water detection is attenuation due to the distance between the source and receiver. This can be overcome with the help of a well-designed preamplifier or a louder source level.

Figure 2.11: Wenz Curve representing deep-ocean ambient noise characteristics [6]

There are a variety of other factors that may degrade the acoustic signals. Some of these include a change in spectrum due to the transducer, path loss due to geometric spreading, and Doppler frequency spread due to a moving source or receiver [1]. These sources and others can negatively impact the ability of the DPSK detection circuit to operate properly, therefore, this study is important to see how susceptible this detector is to the real underwater environments it is meant for.

CHAPTER 3

METHODS

3.1 Overview of Test Plan

Before diving into the details, it is important to understand the overview of the test plan which can be broken down into three tasks. The first task is to write a MATLAB script which simulates the DPSK demodulation and cross-correlation processes in a way which closely represents the processes implemented in hardware. The purpose of this is to develop a model of how the hardware detection circuit will ideally behave. The second objective is to modify the VHDL code and hardware of DBV Technology's prototype detector in order to access and record the real time cross-correlation values which will be used to directly represent how well the hardware detection process is working. The final task is to perform at-sea tests in both deep and shallow water environments and to compare the hardware DPSK detection results with the MATLAB DPSK detection results. A direct comparison like this will give insight into how well the hardware detection process works, how it can be improved, and how to determine the best detection threshold settings.

3.2 Generating DPSK Codes

The DPSK detection circuit uses one 32 bit binary sequence (or code) for addressing purposes and a combination of additional codes for command and control applications. For this study, the detection of only one 32 bit sequence (the address code) is considered for simplification of the problem. Out of a set of 33 unique Gold codes used by the DPSK detection circuit, two were chosen for this

research. The first will be referred to as Code 1 which the DPSK detector should recognize as its stored sequence and the other will be called Code 2. Since Code 2 is a different code within the same Gold code set, it should have a low cross-correlation value as explained in section 2.4.

A MATLAB function was created to convert these Gold codes into DPSK signals (Appendix A.1). First, these two codes were encoded using the same process shown in figure 2.3 from chapter 2. The MATLAB implementation can be seen in the excerpt below:

```
encode = ones(1,length(code)+1);  
for i = 1:length(code)  
    encode(i+1) = not(xor(code(i),encode(i)));  
end
```

The binary values obtained by the encoding process determines the phase shifts of the DPSK signal. To achieve this in MATLAB, the encoding array was adjusted to match the array size of the entire transmitted signal. Since these signals will later be converted into .wav files, the sampling frequency was chosen to be 96kHz. It was previously determined that the carrier frequency for the DPSK detector is 12kHz and that 6 complete cycles will represent on bit of the Gold sequence. This means that in MATLAB, each of the 32 binary values will be represented by 48 bits in this adjusted-length array called "binary_signal." The binary value was also adjusted to -1 and 1, instead of 0 and 1, so when the binary code is multiplied with the carrier frequency, the phase shifts are generated.

Figure 3.1: Sample of the DPSK signal (bottom) generated in MATLAB from a Gold code binary sequence (top).

This MATLAB script makes it easy to generate DPSK signals based off of any provided binary sequence. The code can also be easily adjusted to generate DPSK signals for different length codes, frequencies, and bit rates. The main purpose of this MATLAB script is to generate DPSK .wav files to be used for bench testing and later to be used as input signals for a transmitter source during the at-sea tests. Signal intensity and SNR can also be adjusted to simulate how the DPSK detection circuit operates in noisy environments.

Figure 3.2: The same DPSK signal generated by the MATLAB code with varying SNR levels. Signal amplitude is 20mV.

3.3 MATLAB DPSK Detection

To better understand how noise affects the DPSK detection process, the hardware DPSK detector was essentially recreated in a MATLAB script. The analog to digital conversion, sampling, demodulation, and cross-correlation processes are all handled as closely as possible to how they are implemented on the CPLD (see Appendix A.2). The following block diagram shows the basic demodulation process implemented in hardware. Besides the bandpass filter and comparator circuits, all of the demodulation and cross-correlation processes are implemented on a CPLD.

Figure 3.3: Hardware block diagram of the DPSK detector circuit

Figure 3.4 show the simulation results of the front end (BPF and Comparator) of the DPSK detector implemented in MATLAB. In this specific example, the signal-to-noise ratio was set to 9 dB. Within the time window shown, there are two phase shifts created by the modulated signal. Its difficult to see until the signal passes through the bandpass filter. At this point most of the noise is removed and the phase shifts are easier to see. The signal then passes through the comparator which hardlimits the value to a 0 or 1 depending on the amplitude of the filtered signal.

Figure 3.4: Simulated results of the front end of the DPSK detection circuit.
SNR = 9dB.

Now that the signal has been digitized, the DPSK signal can be demodulated. As shown in figure 3.3, this process is completed by comparing the current data bit from the comparator with the previous data bit. In hardware, this is done using a first in, first out (FIFO) data buffer which has been delayed the proper sample size. In MATLAB, the comparator data is stored in an array and a new variable is created that simply shifts the array by the correct amount of samples. The original comparator array is zero padded to keep the arrays the same length so that they can be easily compared with an exclusive-or function. The result of this process is the demodulated signal (figure 3.5).

Figure 3.5: Software demodulation process. Comparator output (top), delayed comparator output (middle), and demodulated signal (bottom) resulting from the XNOR operation.

The final step in the process is to cross-correlate the demodulated signal with the desired code (Code 1). Three examples of the cross-correlation process are presented in figure 3.6. The top three plots (a) are the results of the simulation process when the SNR of the signal is 12 decibels. The top (red) binary sequence is the ideal demodulated signal. It will always show a demodulated Code 1 with no missed detections. The binary sequence directly below it (blue) is the demodulated signal resulting from the simulated input of the system. Each of the black dots on the demodulated sequence represents the data samples collected for the cross-correlation process. In this case, the SNR was high enough that no bit errors occurred resulting in a perfect cross-correlation. This is visualized in the plot to the right of the demodulated signals. The red line represents the autocorrelation of Code 1 while the blue line represents the cross-correlation with the input signal. When the SNR = 0 dB

(b), there are a few bit errors that occur in the demodulated signal. These bit errors are represented by red dots rather than black dots. When looking at the cross-correlation plot, the maximum correlation value is now around 24 meaning that the detection threshold needs to be significantly lowered at this SNR for a detection to occur. As expected, when the SNR is decreased further the cross-correlation results worsen. In figure 3.6c, the SNR value is set to -6dB resulting in a demodulated signal containing more bit errors.

Now that the detection process works in MATLAB, the concept is extended to accept input signals from large binary or wav files. This allows for easy comparison to the hardware cross-correlation values and makes it possible to record hydrophone data and play it directly into the MATLAB detector.

Figure 3.6: Examples of the simulated cross-correlation process. (a) SNR = 12dB, (b) SNR = 0dB, (c) SNR = -6dB

3.4 Hardware Detector Modifications

During normal operation the DPSK detection circuit does not output it's cross-correlation results. In order to determine how well the hardware detector operates

in different ocean environments there has to be a way to access the cross-correlation results in real time. In addition, storing these values for future analysis allows for easy and direct comparison with the MATLAB detector. The block diagram in figure 3.7 shows the major electronic components used in the hardware setup. The DPSK detection circuit needs to be modified to output cross-correlation values in real time across a serial interface. A USB data acquisition (DAQ) system is used to record the cross-correlation values as well as the analog input signal simultaneously. A Raspberry Pi 2 is used to control the USB DAQ system and to store all the recorded data onto a USB flash drive. The following subsections detail each major piece of the hardware system.

Figure 3.7: Block diagram of the Hardware setup

3.4.1 DPSK PCB Modifications

As mentioned earlier, the entire DPSK detection process is implemented on a CPLD. The specific CPLD highlighted in figure 3.8, is the Altera Max V 5M240Z [8]. These devices are programmable using VHSIC Hardware Description Language (VHDL) with the free Quartus II Web Edition Software. The physical device is reconfigured using Altera's USB Blaster programmer via the 10-pin header on the board [14]. The DPSK detection PCB was built with a surface mount LED for troubleshooting and extra general-purpose input/output (GPIO) pins which are utilized for the hardware

modifications. The goal of modifying the VHDL code is to continuously output serial packets containing cross-correlation samples that can be recorded using a DAQ system.

Figure 3.8: Modifications to the DPSK detection board. 1) Altera's CPLD. 2) Extra GPIOs for hardware expansion. 3) Programming header for the CPLD

There are plenty of additional logic elements available on the CPLD to allow for added functionality. This space is used for two new blocks of VHDL code written for this study. Within the VHDL code, there is a logic vector called 'xcorrData' which holds the 8 bit signed integer value representing the latest cross-correlation value. Serial.vhd is an entity (or block of VHDL code) used to package this variable and send it to a GPIO one bit at a time. To accomplish this, a second process within DPSKDetect.vhd is written to take care of handshaking between the cross-correlation process and serial.vhd while making as few modifications to the original code as possible. The purpose of this code is to accept only one new cross-correlation value per bit sequence comparison and to relay this information to serial.vhd only when it has completed sending the previous data sample. While serial.vhd runs at 28.8kHz (used as the baud rate), this process operates at a much higher clock rate of 12MHz.

The block diagrams for the VHDL code and how they interface with the original DPSK detection code is shown in Figure 3.9. The full code is in Appendix B.1 and B.2.

Figure 3.9: VHDL Block Diagrams

The two oscilloscope screen captures are examples of the cross-correlation samples obtained from the serial output of the DPSK detection circuit. Figure 3.10 shows a capture of one sample. Each sample includes a start bit and a stop bit. The data itself is sent packaged with the least significant bit first. In this specific example, the cross correlation value is binary '00000010' or a decimal value of 2. When a sample is not being sent there is idle time and the output of the DPSK detector is kept high. Figure 3.11 shows multiple samples sent continuously.

Figure 3.10: Oscilloscope capture of one cross-correlation package

Figure 3.11: Oscilloscope capture of multiple serial data packets

3.4.2 Recording System

The DAQ system used to record the input signal and cross-correlation data is the Measurement Computing USB-1608GX. This is a 16-bit USB powered DAQ that is capable of recording up to 500kSps and up to 16 single-ended (SE) channels. It also has a built in programmable preamplifier circuit which can be used to adjust the data windows from +/- 10V down to +/-1V to maximize the bit resolution for small signals [15]. For this study, only the first two SE channels were used. The first channel is set

to record the input signal to the DPSK. The second channel is used to record the serial cross-correlation data. The sampling rate was selected to be 96kSps per channel. A C-sharp program was written on a Raspberry Pi 2 to initialize the recorder and to store the data on a 32GB USB flash drive. The data was split into one minute long binary files (Appendix C). The rc.local file on the Raspberry Pi 2 was also modified to automatically start the recording process when powered up.

3.4.3 MATLAB Code

A MATLAB script was written to plot the hydrophone data and cross-correlation data from the binary files (Appendix A.4). This m-file concatenates a specified number of binary files into a variable array for the Hydrophone data and a separate array for the serial cross-correlation data. From here, the cross-correlation data is converted back into its signed integer value so the decimal value can be plotted.

Figure 3.12: DPSK signal and cross-correlation data from a hardware bench test.

Figure 3.13: DPSK signal and cross-correlation values for one transmitted signal

3.5 At-Sea Tests

It was desired to obtain hydrophone data and cross-correlation results from the hardware detection circuit in both shallow and deep ocean environments. Two test locations were chosen for this study. The shallow water test was completed in the Narragansett Bay just north of the Jamestown bridge and the deep water test was completed in Bermuda.

3.5.1 Narragansett Bay Test

With the hardware modifications complete, the electronics were packaged into a custom waterproof housing designed specifically for the Narragansett Bay test. All of the electronics associated with the modified hardware detection circuit (DAQ, embedded computer, USB memory, and DPSK detection circuit) were mounted on an aluminum bracket inside this pressure housing. In addition, a 7-cell Alkaline D-Cell

battery pack (10.5V) was used to power the electronics for up to 40 hours and a switching voltage regulator circuit was built to regulate the voltage down to the 5 volt power supply required by the Raspberry Pi 2. The DPSK detection circuit uses a linear voltage regulator on the PCB to further reduce the supply to 1.8 volts.

Figure 3.14: Electronics stack designed for the Narragansett Bay test.

The shallow water housing was made out of a 4 inch schedule 40 PVC pipe which is rated for a maximum operating pressure of 133 PSI which gives an operational water depth rating of approximately 300 feet. A grey PVC rod was modified into an end cap with through holes for two Subconn circular micro series bulkhead connectors. The PVC endcap mates to a schedule 40 union to create a surface seal against an o-ring. One of the two bulkhead connectors is used for turning the system on and off with a shorting plug (figure 3.16). The other bulkhead mates to a HTI-97-DA/AC hydrophone from High Tech, Inc. These hydrophones have an average sensitivity of -192.6 dB re 1V/ μ Pa.

The DPSK detection prototype has an optional 40 decibel pre amplifier built in that can be added or removed with a few jumpers. To determine how well the detection process works with and without the optional pre-amplifier two hardware circuits were built for the shallow water test (figure 3.17).

Figure 3.15: PVC pressure housing fully assembled

Figure 3.16: Pressure housing end cap with shorting plug and hydrophone cable

Figure 3.17: Both pressure housings mounted and ready for deployment

3.5.2 Bermuda Test

Accessing deep water was made possible by involvement with an at-sea test of Son-O-Mermaid, a joint project from Frederik Simons of Princeton University and Harold T. Vincent II of The University of Rhode Island funded by a grant from the National Science Foundation (NSF). The data collection was completed during down time aboard the R/V Atlantic Explorer, a research vessel owned and operated by the Bermuda Institute of Ocean Sciences (BIOS) in coordination with the University-National Oceanographic Laboratory System (UNOLS) fleet.

Two systems were built for the deep ocean test, a receiver to be submerged to full ocean depth and a transmitting source on the surface. At the time of this trip the modifications to the hardware detection circuit were not yet complete. Instead of implementing a receiver system similar to the shallow water test, a DAQ system was used to record only the transmitted acoustic signals. The resulting acoustic files can be used in the hardware and MATLAB detection processes at a later date. The receiver system which consisted of a hydrophone, DAQ system, embedded computer, and timed recovery system was packaged into a glass pressure housing with enough batteries to run the recorder for over 24 hours. The glass pressure housing (figure 3.18) is a 14mm thick 17-inch diameter VITROVEX glass sphere from Nautilus. These glass spheres are designed specifically for deep ocean

Figure 3.18: Electronics inside of glass pressure housing

instrumentation with a depth rating of 6,700 meters [16]. For the transmitter, a surface buoy (figure 3.19) was built out of 8-inch PVC pipe with flanges and blind flanges on both ends to form end caps. On the bottom of the instrument, a tonpilz transducer was bonded to the blind flange. The transducer was driven by a 600-watt (TI-600 Amplifier) Class D Audio power amplifier. The modulated DPSK signal wav files were supplied by a WAV Trigger (WIG-13660) by SparkFun Electronics and code selection was done using an Arduino Micro microcontroller circuit. An Iridium/GPS antenna was placed close to the top end cap to relay timing and position information. 3 6-cell lead acid batteries powered the system and were located close to the bottom of the tube to help orient the tube vertically. An inflatable flotation ring was used as buoyancy.

Figure 3.19: Block diagram of transmitting surface buoy (cross section)

Figure 3.20: Surface buoy. From top left: tonpilz transducer, inflatable ring, transmitter electronics.

An outline of the deep water acoustic data collection is as follows. First, the receiver pressure vessel was deployed to 5,000 meters depth. A few hours after deployment of the recording system, the surface buoy containing a transmitter source was deployed and towed by the research vessel. The research vessel maneuvered at a low speed making several passes over the deployment location of the receiver. The GPS tracker inside the surface buoy reported back time and position information to determine what time the buoy was located above the recorders

presumed position. After several hours of transmitting, the surface buoy was retrieved. The submerged pressure housing was programmed to begin ascending to the surface 12 hours after deployment and approximately 14 hours after deployment it was successfully recovered.

Figure 3.21: Recording system pressure vessel inside of an orange hard hat for protection and easier handling.

CHAPTER 4

RESULTS

4.1 Bench Test: Simulation and Hardware Results

Using the MATLAB DPSK signal generator described in chapter 3, wav files were created and used as the input signals of MATLAB detector and the Hardware detector. Each wav file was five minutes long and repeated a modulated DPSK signal at a rate of four times a second. The following table describes each of the wav files used.

File Name	Description (each file is 5 minutes long)
CODE1.wav	Ideal DPSK signal. Matches Gold sequence to be detected
SNR12.wav	CODE1 with additive white Gaussian Noise (SNR = 12)
SNR9.wav	CODE1 with additive white Gaussian Noise (SNR = 9)
SNR6.wav	CODE1 with additive white Gaussian Noise (SNR = 6)
SNR3.wav	CODE1 with additive white Gaussian Noise (SNR = 3)
SNR0.wav	CODE1 with additive white Gaussian Noise (SNR = 0)
SNR-3.wav	CODE1 with additive white Gaussian Noise (SNR = -3)
SNR-6.wav	CODE1 with additive white Gaussian Noise (SNR = -6)
NOISE.wav	Pure white Gaussian Noise (WGN) generated in MATLAB
CODE2.wav	Ideal DPSK signal. Different code from same Gold sequence

Table 4.1: Wav File Descriptions

To obtain the cross-correlation values for the MATLAB detector, the wav files were loaded directly into the MATLAB DPSK detection script. All of the cross-correlation results were saved in a MATLAB workspace for later analysis. For the hardware portion, the wav files were played out of the sound card of a laptop through a standard 3.5mm headphone jack which was plugged directly into the audio input of the DPSK detection circuit. The signal amplitude was constant at 250mV. Both the input signal and cross-correlation results were recorded with the DAQ and stored in one minute long binary files on the USB flash drive. Once the desired amount of data was recorded, the binary files were transferred to a laptop computer and loaded by the Serial2Xcorr.m MATLAB file described in section 3.4.3. These cross-correlation results were also saved to a workspace to be compared to the MATLAB detection results. The cross-correlation data can be analyzed in many useful ways to evaluate how well the detection process works. For this study, this evaluation is done using probability mass functions (PMFs) and receiver operating characteristic (ROC) curves.

4.1.1 PMF Bench Test Results

Figure 4.1 shows the PMFs of the cross-correlation values obtained using the MATLAB detector at different SNR levels. The PMF was generated using the 1,200 data points which correspond to where the demodulated Code1 should be detected. When the cross-correlation value is 32, the DPSK demodulation process detected every bit perfectly. Imperfect detections will lead to lower cross-correlation values. From the figure, it's clear that when the wav file containing a modulated signal with

no noise is loaded into the MATLAB detector there is perfect correlation every time. When the SNR is 12dB or 9dB, the detector still works very well with a detection probability greater than 0.9. Around 6dB and 3dB SNR, the noise begins to create more frequent bit errors in the DPSK demodulation process causing the majority of cross-correlation values to range between 15 and 32. The MATLAB detector shows that this DPSK process is not very effective when SNR is 0dB or less. In fact, looking at the cross-correlation values, there is very little difference between these noisy signals and pure random noise. When a purely computer generated white Gaussian noise signal is loaded into the MATLAB detector, the cross-correlations results in the PMF resemble a Gaussian curve centered at zero. The majority of the time, this random noise alone will produce cross-correlation values between -15 and 15, however the PMF shows that its possible for a cross-correlation value of 20 to occur on occasion. This is valuable insight to have when determining a detection threshold setting considering there is a significant risk of false detection due to noise alone. In practice, this could result in a loss of scientific instruments. Setting a cross-correlation threshold around 24 would lead to detecting the modulated DPSK signal most of the time for SNRs as low as 6dB.

Figure 4.2 shows the PMFs generated using the hardware detector cross-correlation values. Overall, the results from the hardware detection circuit are very similar to those of the MATLAB detector. What is noticeable right away is that even with a perfect DPSK signal with no noise there are a few missed detections. In addition, even when the SNR level is relatively high there are still low cross-

correlation values which are not predicted by the MATLAB simulation (see figure 4.3 and 4.4). The probability of these low cross-correlation values increase as the noise level increases as well. To take a closer look at this issue, the cross-correlation values obtained from the transmitted DPSK signals were plotted in the time domain. All the values in figure 4.5 were obtained from the sound file SNR12.wav. From this figure, it is easy to see that the cross-correlation data resulting from the MATLAB detection process show perfect demodulation of the transmitted signal nearly every time over the 5 minute file. Although the hardware detector has a 0.8 probability of perfect demodulation, it is clear that there are still many poor cross-correlation results. These poor results also seem to happen periodically which suggests that there may be an issue associated with clock offset in the hardware detectors sampling process. These results also highlight the importance of this study since this is a previously unknown issue that would not have gone otherwise unnoticed.

Although the hardware detector has these poor cross-correlation results at times, it still has higher cross-correlation values than the MATLAB detector when comparing the results with SNRs of 6dB and 3dB. The hardware detection cross-correlation PMFs for -3dB and -6dB were not included in this figure due to the inability to reliably determine which cross-correlation values correspond to the detection of a received DPSK signal. Unlike the software simulation where these cross-correlation samples are perfectly spaced, clock drift from both the digital to analog converter of the laptop sound card and the DAQ system cause the cross-correlation data rate to be inconsistent by one or two samples. This hardware

problem was easily solved for higher SNRs when the cross-correlation values were above 20, but in noisier situations, it becomes more challenging. This situation brings to light the importance of being able to simulate these results in MATLAB. Figure 4.3 and 4.4 shows a clearer comparison of the MATLAB detector and hardware detector for an ideal signal and various SNRs.

Figure 4.1: PMFs of the MATLAB detector cross-correlation values

Figure 4.2: PMFs of the hardware detector cross-correlation values

Figure 4.3: Comparison of MATLAB and hardware cross-correlation PMFs for an Ideal signal and SNR of 12dB and 9dB

Figure 4.4: Comparison of MATLAB and hardware cross-correlation PMFs for SNR of 6dB, 3dB, and pure noise

Figure 4.5: A comparison of the cross-correlation results over time

In addition to determining how the detectors work in noisy environments, it's also important to verify that they will not detect other binary codes within the same Gold code set. Figure 4.6 shows the cross-correlation results of wav files CODE1.wav, CODE2.wav, and NOISE.wav for both detectors. These results show that when Code 2 is demodulated, and Code 1 is the desired binary sequence, the cross-correlation value is predictable and low.

Figure 4.6: PMF comparisons of Code1, Code 2, and computer generated white Gaussian noise.

4.1.2 ROC Curve Bench Test Results

The PMFs only reveal what happens during the 1,200 data points when the DPSK signal is present. Since the bit rate of the DPSK detector is 2kHz, there are 600,000 cross-correlation values within the 5 minutes of data. False detections can occur from any of these values. The ROC curve uses all the cross-correlation values to plot the true positive rate (TPR) against the false positive rate (FPR) at a variety of threshold settings. These curves provide a means of cost/benefit analysis for deciding the best cross-correlation threshold for the DPSK detection circuit. Figure 4.7 is the ROC curve for the MATLAB simulation. As expected, the ideal signal yields the best

possible results. Any threshold above 18 will result in a positive detection every time with no risk of a false detection. Once the threshold falls to 18 or below, the positive detection will still occur every time, but the chance of a false detection increases. If the SNR is 6dB or higher, it is possible to set a threshold to 24 or higher without any real risk of a false detection. When the SNR is 0dB or lower, the ROC curves closely resemble the white Gaussian noise (WGN) ROC curve. In these situations, the threshold has to be greatly reduced to have any chance of a true positive detection, however, this will drastically increase the probability of false detection.

Figure 4.7: MATLAB ROC curves at various SNR levels

Figure 4.8 shows the ROC curves generated from the hardware cross-correlation values. Just like with the PMF, the hardware detector ROC curves are very similar to those of the MATLAB detector. The noticeable difference here is that it is impossible to obtain a perfect true positive rate or detection for any SNR level no matter what

the threshold settings are. Also as shown in the PMFs, the detection results of 0dB SNR signal seems better for the hardware detector, although still poor.

Figure 4.8: Hardware ROC curves at various SNR levels

4.1.3 Improving the Bench Test Results

The bench test verifies the MATLAB detector and hardware detector modifications both work properly, however it introduces a few ideas that could improve the hardware detection results. In figure 4.3 and 4.4, notice that the hardware detection produces a few single digit cross-correlation values even when the SNR is relatively high. This is not predicted by the MATLAB detector. One possible reason for this result is that the sampling process of the hardware detection circuit does not always select samples at the signal anti-nodes. In MATLAB, the detection process is ideal and always samples at the signals peak values where the effect of the noise is minimized. The hardware detector may begin sampling at the antinodes but

due to clock drift from both the detection circuit and the computer audio card producing the DPSK signal, the hardware detector may occasionally sample at the signal nodes maximizing the potential for bit errors. This could be fixed with the introduction of a peak detection circuit in future research.

Another possible improvement was introduced while modeling the hardware bandpass filter in MATLAB. As could be presumed, narrowing the band around the carrier frequency and increasing the order of the filter greatly improves the simulated DPSK detection as shown in figure 4.9 when implementing a second order bandpass filter with cutoff frequencies of 8kHz and 16kHz. Currently the bandpass filter is a simple second order resistor-capacitor (RC) circuit with cut off frequencies of 2kHz and 50kHz. The PMFs in the figure show that there is near perfect cross-correlation detection for SNRs as low as 6dB, a noticeable improvement over the current model in figure 4.3 and 4.4.

Figure 4.9: PMFs of MATLAB detector results using an improved band-pass filter

4.2 At-Sea Test Results

4.2.1 Narragansett Bay Test Results

The deployment and recovery of both shallow water instruments were successful. Unfortunately, the DPSK detection system containing no preamplifier did not produce any usable data for a shallow water cross-correlation analysis due to an issue with the input signal. Although the DPSK detector operated correctly, there was an issue with the hydrophone. This was the first test involving this specific hydrophone and it was later determined that because there is no built in preamplifier, the hydrophone is not suitable to drive the input of the USB-1608GX. A zero gain preamp would need to be designed to accept the input from a capacitive sensor without altering the signal. This issue did not affect the performance of the other shallow water instrument because of the built in optional preamplifier of DPSK detector. Due to time restrictions, the shallow water results only include the preamplified hardware DPSK detector. During a second shallow water test the zero gain DPSK detection electronics were replaced with a simple USB audio recorder and a raspberry pi used to record the hydrophone directly. When combined with the 40dB detector set up, it is now possible to record the hydrophone signal, 40dB preamplified signal, and cross-correlation data.

Figure 4.10: Hydrophone signal, preamplified signal, and cross-correlation results from the second Narragansett Bay test.

Figure 4.11: Hydrophone signal, preamplified signal, and cross-correlation results from the second Narragansett Bay test zoomed in to show one seconds worth of data

Figures 4.10 and 4.11 show the hydrophone signal, preamplified signal, and cross-correlation results all as a function of time. Unfortunately, it seems that the DPSK demodulation process did not function very well in shallow water. The

hydrophone data was used to test the MATLAB detector as well to see if the results were consistent. According to the figure below, the MATLAB detector seems to perform similarly overall but does not correlate perfectly with the results in hardware. This could be due to the clock drift issue explained in section 4.1.2. If only considering the SNR level of the signal, these poor cross-correlation results are surprising. What this really means is that SNR level is only one of many factors that

Figure 4.12: MATLAB detection results versus hardware detection in 20 feet of water.

affect signal detection. As explored in Chapter 2, multipath is a major source of interference in the shallow water environment. It's expected that as the water depth increases, the signal overlap with the multipath becomes less of an issue resulting in fewer bit errors. The shallow water test discussed above was carried out in 20 feet of water. The first shallow water test was in 30 feet of water. The following figure shows the recording of the preamplified hydrophone signal and the cross-correlation results for both hardware and MATLAB detection processes.

Figure 4.13: Preamplified hydrophone signal and cross-correlation results for a shallow water test in 30 feet of water

Although the cross-correlation results still are not ideal, there is a clear improvement over the same test carried out in 20 feet of water. It would be interesting to extend this concept by repeating multiple shallow water tests in varying depths to see if the results continue to improve as the depth increases. In addition to multipath, there are other factors that could have interfered with detection. The water conditions were calm during the first test but choppy due to an approaching storm during the second test. Also during the first test the transmit transducer was hung off the side of the boat but fixed to a transducer pole during the second test. It was later determined that the beam path of the transducer is very directional meaning that the transducer orientation will highly effect the outcome on the receiver side.

As part of the shallow water test, it was also important to make sure that the DPSK detection prototype would not mistakenly detect other Gold codes. To test this, Code 2 was transmitted for a portion of the shallow water test and it was verified that the cross-correlation values remained under 20, a value reasonable for correlating against noise as shown in during the bench test. A portion of the recorded data and MATLAB simulation results are shown in figure 4.14.

Figure 4.14: Cross-correlation results from transmitting code 2 in 20 feet of water

It was desired to generate PMFs and ROC curves based off of the data obtained in the shallow water test so that a comparison could be made with the bench test results. Unfortunately, the same issue that prevented the generation of these plots for SNR levels of -3dB and -6dB in the bench test prevented the generation of the plots from the shallow water results.

4.2.2 Bermuda Test Results

The deployment and recovery of the deep water instrument was successful. The DAQ system inside the glass housing recorded properly and there were what appear to be very faint DPSK signals present as shown in the spectrograms (figure 4.15 and 4.16). A few binary files containing the hydrophone signals were played into the MATLAB detector and Hardware detector, but no signals could be detected. Looking at the spectrogram, there are what appear to be tones at 12kHz which could interfere with the detection process in addition to the attenuation of the signal.

Figure 4.15: Spectrogram of the Bermuda Test.

Figure 4.16: Spectrogram zoomed in and filtered to show DPSK signals

Even though commands could not be detected directly from the recorded signals, it is still interesting to compare Bermuda's deep ocean noise to the AWGN generated by the 'randn()' function in MATLAB. Five minutes worth of Bermuda data from files recorded after the surface buoy stopped transmitting were used to represent the Bermuda noise. For this portion of the study, it was assumed that this deep ocean noise is wide-sense stationary (wss) meaning that the signal detection probability will not vary over time. Figure 4.17 shows the power spectral density (PSD) of the Bermuda noise. This figure can be compared to the wenz curve discussed in section 2.5.

Figure 4.17: Power spectral density of Bermuda noise.

The noise voltage levels was scaled to match the SNR levels used in the bench test.

The bench test was repeated using the Bermuda noise. The resulting PMFs and ROC curves are in the figures below.

Figure 4.18: PMFs of MATLAB detector cross-correlation values using Bermuda noise

Figure 4.19: PMFs of hardware detector cross-correlation values using Bermuda noise

Figure 4.20: Comparison of MATLAB and hardware cross-correlation PMFs for an Ideal signal and SNR of 12dB and 9dB with Bermuda noise

Figure 4.21: Comparison of MATLAB and hardware cross-correlation PMFs for SNR of 6dB, 3dB and pure Bermuda noise

Figure 4.22: MATLAB ROC curves at various SNR levels using Bermuda noise

Figure 4.23: Hardware ROC curves at various SNR levels using Bermuda noise

Comparing these results to the original bench test with computer generated noise, show that the MATLAB and hardware DPSK detectors work just as well with real deep sea noise. In fact, both detectors seem to have worked a little better on occasion. This shows that is plausible that the detectors could operate well in the deep ocean environment if the SNR could be increased despite the tone recorded at 12kHz. The results of the deep ocean test suggest that detection can be improved by increasing the transmission source level, increasing the preamplifier gain and filtering circuit on the receiver front end, or moving the carrier frequency to a band that does not contain interference due to a constant tone.

CHAPTER 5

CONCLUSION

The overall objective of this investigation was to develop a method for testing DBV Technology's DPSK detection prototype which would be less time consuming and provide more insight into the detection process than previous methods. Modifications were successfully made to the hardware detection circuit to reliably output the cross-correlation values of the detection process and a MATLAB detection algorithm was written which accurately simulates the detection results of the hardware system without the need of any electronics. A bench test was used to verify the proper operation of both these detection systems as well as determine how well the DPSK demodulation process works in noisy environments. This was done by adjusting the SNR level of the input signals using computer generated white Gaussian noise.

Up to this point, most testing was carried out in ideal acoustic conditions, not representative of the real ocean environment that the detector is designed for. As for the other purpose of this study, real acoustic data was recorded in the Narragansett Bay and Bermuda to obtain detection results for shallow and deep water environments. It was of interest to determine how the different properties of these two acoustic environments would affect the detection process. The results obtained from the shallow water study show that multipathing is likely a major cause of interference in the successful detection of the DPSK signals. The rate of detection seemed to improve in 30 feet of water opposed to 20 feet of water. Although no

detections were obtained using the recorded signals, the results for the Bermuda test show that detection in the deep ocean's ambient noise is possible as long as the SNR level is high enough. More data should be collected before making any final decisions, but initial results from all the noise environments show that setting a cross-correlation threshold higher than 24 should eliminate any concern of false detection from ambient noise.

Although this investigation was successful overall, there are still flaws that should be addressed to improve the test set up as well as the detection process. As demonstrated by the bench test and shallow water test, determining if a cross-correlation result is associated with a demodulated DPSK signal can be challenging with enough signal distortion. Developing a way to gain more accuracy in the analysis of cross-correlation values would be beneficial to determining the effectiveness of the detection process. Also correcting the issue preventing the hydrophone from being recorded in the shallow water test will allow for a comparison between the preamplified DPSK detector and non-preamplified DPSK detector. In addition, a study taking a closer look at the operating conditions associated with multipath would provide important information about the minimum, maximum, and optimal depths of the DPSK detection circuit in shallow water. This could be achieved by repeating the shallow water test multiple times in varying water depths.

The testing methods developed during this study could also be used as a tool to help improve the actual detection process. For example, future research could involve building several multi-order band-pass filters and determine the best tradeoff

between detection results and added power consumption. These results could be predicted with the MATLAB DPSK detector rather than investing time in designing a physical preamplifier circuit. It was also suggested that designing a peak detector for the hardware detection circuit might improve the cross-correlation results by fighting the effects of the CPLDs potential clock drift. Another way to improve the detection circuit could be to increase the Gold sequence length from 32 bits to 64 bits or higher. This will presumably widen the gap between the cross-correlation values obtained by noise and the desired Gold codes allowing for lower cross-correlation thresholds to be set while lessening the chance of false detection. There are many other modifications that could be made to the DPSK detection circuit. The systems developed for this investigation provide a good starting point for exploring these improvements.

APPENDIX A:

MATLAB CODE

This section includes all the major MATLAB scripts that were used for this study.

A.1 DPSK_Signal_Generator.m

This code was used to generate the wav files used for all the bench testing. It was also used to generate the input signals for the transmitter source during the at sea exercises.

```
clear all; close all; clc;

fs = 96e3; % sample frequency
fc = 12000; % carrier frequency
SNR = -6; %in decibels
Vs = 1; %signal voltage level

%{
At this fs, there are 48 samples per bit.
Explained: Each bit is 6 cycles. The carrier frequency is 12kHz and
the sample rate is 96kHz. This means there are (96/12 = ) 8 samples
per cycle. 8 samples/cycle * 6 cycles/bit = 48 samples/bit. There
will be 32 data bits but we need to add one more bit to account for
the differential signal.
%}

t = 0:1/fs:(33*48-1)/fs; %33 bits multiplied by 48 samples/bit
carrier = sin(2*pi*fc*t);

% Code1
code = [1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0
1];
% Code2
%code = [1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 1 1
0 1];

%encoding the sequence to be multiplied by the carrier frequency
encode = ones(1,length(code)+1);
for i = 1:length(code)
    encode(i+1) = not(xor(code(i),encode(i)));
end

%Adjusting the encoded signal to have the same array dimension as the
%carrier.
originalcode = zeros(1,48*32);
for j = 1:length(code)
    if code(j)== 1
        originalcode(1+48*(j-1):48*j) = 1;
```

```

        else
            originalcode(1+48*(j-1):48*j) = 0;
        end
    end
    originalcodepadded = [zeros(1,48) originalcode];

    binary_signal = zeros(1,48*33);
    for ii = 1:length(encode)
        if encode(ii)== 1
            binary_signal(1+48*(ii-1):48*ii) = 1;
        else
            binary_signal(1+48*(ii-1):48*ii) = -1;
        end
    end

    %Array multiplying the carrier signal with the binary signal to
    %created the DBPSK signal to be used with the CPDL's demodulation
    %circuit.
    DBPSK = Vs*carrier.*binary_signal;
    Vn = Vs/(10^(SNR/20));

    IdealLoop = [];
    for n = 1:600% was 60 (now 10 minutes)
        IdealLoop = [IdealLoop zeros(1,.0035*fs) DBPSK zeros(1,.23*fs)...
                    zeros(1,.0035*fs) DBPSK zeros(1,.23*fs)...
                    zeros(1,.0035*fs) DBPSK zeros(1,.23*fs)...
                    zeros(1,.0035*fs) DBPSK zeros(1,.23*fs)];
    end
    WhiteNoise = Vn*(randn(1,length(IdealLoop)));
    NoisyLoop = IdealLoop + WhiteNoise;

    figure(1)
    subplot(2,1,1)
    plot(0:1/48:(length(originalcodepadded)-1)/48,...
        originalcodepadded,'-r', 'linewidth',2.5)
    axis([8 14 -.5 1.5])
    set(gca,'xgrid','on')
    set(gca, 'YTickLabelMode', 'Manual')
    set(gca, 'YTick', [])
    title('Binary Code', 'FontSize',20)
    subplot(2,1,2)
    plot(0:1/48:(length(originalcodepadded)-1)/48,DBPSK, 'linewidth',2.5)
    axis([8 14 -1.5 1.5])
    set(gca,'xgrid','on')
    set(gca, 'YTickLabelMode', 'Manual')
    set(gca, 'YTick', [])
    title('Modulated Signal', 'FontSize',20)

    figure(2)
    plot(NoisyLoop)
    hold on
    plot(IdealLoop, 'g')
    plot(WhiteNoise,'r')

    % create wavefiles for testing the DPSK Detector
    wavwrite(IdealLoop, fs, 'Code1_10m(Narr).wav');

```



```
wavwrite(NoisyLoop, fs, 'SNR-6.wav');
wavwrite(WhiteNoise, fs, 'NOISE.wav');
```

A.2 Simulated_DPSK_Detection.m

This code was used as the initial MATLAB DPSK detector.

```
clear all; close all; clc;
SNR = 6; %in decibels
Vs = 1; %signal voltage level
fs = 96e3;
fc = 12000;
CyclesPerBit = 6;
SamplesPerBit = (fs/fc)*CyclesPerBit;

%=====
%   MODULATION PROCESS
%=====
t = 0:1/fs:(33*SamplesPerBit-1)/fs; %33 bits multiplied by 48
                                   %samples/bit
carrier = sin(2*pi*fc*t);
code = [0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1
0];

%encoding the sequence to be multiplied by the carrier frequency
encode = ones(1,length(code)+1);
for i = 1:length(code)
    encode(i+1) = not(xor(code(i),encode(i)));
end

%Adjusting the encoded signal to have the same array dimension as the
%carrier.
binary_signal = zeros(1,SamplesPerBit*33);
for ii = 1:length(encode)
    if encode(ii)== 1
        binary_signal(1+48*(ii-1):SamplesPerBit*ii) = 1;
    else
        binary_signal(1+48*(ii-1):SamplesPerBit*ii) = -1;
    end
end

%Array multiplying the carrier signal with the binary signal to
%created the DBPSK signal to be used with the CPDL's demodulation
%circuit.
DBPSK = Vs*carrier.*binary_signal;

%Account for noise
Vn = Vs/(10^(SNR/20));
hf_noise = Vn*(randn(1,length(DBPSK)));
lf_noise = 0*sin(2*pi*10*t);
%Simulating a signal in a noisy environment.
RealisticSignal = DBPSK + hf_noise + lf_noise; %this last part is low
                                              %frequency noise (waves and such)
```

```

% The demodulation will occur on the CPLD but there should be an
% analog circuit on the front end amplifying the signal and then
% bandpassing it to reduce high and low frequencies.
[b,a] = butter(2,[12000-2400 12000+2400]/(fs/2));
% FilteredSignal is the simulated noisy DBPSK signal passing through
% a band pass filter.
FilteredSignal = filtfilt(b,a,RealisticSignal);

% figure(1) shows the clean DBPSK signal, the simulated noisy signal
% and the simulated band passed signal. The band passed signal is what
% will be played out of the audio card and into the CPLD.
figure(1)
subplot(3,1,1)
plot(t, DBPSK, 'b')
axis([(13*SamplesPerBit-1)/fs (19*SamplesPerBit-1)/fs -2 2])
% (33*48-1)/fs
title('Clean DPSK Signal')
subplot(3,1,2)
plot(t, RealisticSignal, 'r')
axis([(13*SamplesPerBit-1)/fs (19*SamplesPerBit-1)/fs -2 2])
title('DPSK Signal w/ Noise')
subplot(3,1,3)
plot(t, FilteredSignal, 'g')
axis([(13*SamplesPerBit-1)/fs (19*SamplesPerBit-1)/fs -2 2])
title('Filtered DPSK Signal')

%=====
% DEMODULATION AND DETECTION PROCESS
%=====
input = FilteredSignal;
comparator = zeros(1,length(input));
for i = 1:length(input)
    if input(i) <= 0.01
        comparator(i) = 0;
    else
        comparator(i) = 1;
    end
end
IdealComparator = zeros(1,length(DBPSK));
for i = 1:length(DBPSK)
    if DBPSK(i) <= 0.01
        IdealComparator(i) = 0;
    else
        IdealComparator(i) = 1;
    end
end
samples = 2:4:length(comparator); % fs = 96khz, fc = 12khz. 96/12 = 4,
% therefore we sample every 4
Delayed_RealisticDigital = [zeros(1,SamplesPerBit/4)
comparator(samples)]; % padded front side to add delay
Windowed_RealisticDigital = [comparator(samples)
zeros(1,SamplesPerBit/4)];
Mixed_RealisticDigital = not(xor(Windowed_RealisticDigital,
Delayed_RealisticDigital));
bit_locations = 18:12:length(Mixed_RealisticDigital)-12;

```

```

% The following few lines of code simulate the demodulation that will
% occur on the CPLD. The purpose of this simulation is to verify
%that the whole process works as expected.
Delayed_IdealDigital = [zeros(1,SamplesPerBit/4)
IdealComparator(samples)]
Windowed_IdealDigital = [IdealComparator(samples)
zeros(1,SamplesPerBit/4)];
Mixed_IdealDigital = not(xor(Windowed_IdealDigital,
Delayed_IdealDigital));

%the position where each bit is sampled
nT = 18:12:length(Mixed_IdealDigital)-12; %first sample thrown away
%(due to delay in xor). best case for detection would be to start in
%the middle of the next sample
n1 = 0;
n2 = 0;
error_bit = NaN(1,32);
error_location = NaN(1,32);
good_bit = NaN(1,32);
good_location = NaN(1,32);
for j = 1:32
    if
Mixed_RealisticDigital(bit_locations(j))~=Mixed_IdealDigital(nT(j))
        n1 = n1 + 1;
        error_bit(n1) = Mixed_RealisticDigital(bit_locations(j));
        error_location(n1) = bit_locations(j);
    else
        n2 = n2 + 1;
        good_bit(n2) = Mixed_RealisticDigital(bit_locations(j));
        good_location(n2) = bit_locations(j);
    end
end

%The following plots the mixed signal outputs. The top subplot shows
%the ideal mixed digital signal and the bottom plot shows a
%simulation of ht mixed signal using the signal processing techniques
%found in the hardware. This includes re-sampling and a comparator
%output of a noisy input signal. The black dots on each of the plots
%represent the bit values which match up with the original code.
figure(2)
subplot(2,1,1)
plot(0:1/2e3:(length(Mixed_IdealDigital)-
1)/2e3,Mixed_IdealDigital,'r',nT/2e3,Mixed_IdealDigital(nT),'k.','MarkerSize',15)
axis([0 length(Mixed_IdealDigital)/2e3 -1 2])
subplot(2,1,2)
plot(0:1/2e3:(length(Mixed_RealisticDigital)-
1)/2e3,Mixed_RealisticDigital,'b')
hold on
plot(error_location/2e3,error_bit,'r.','MarkerSize',15)
plot(good_location/2e3,good_bit,'k.','MarkerSize',15)
hold off
axis([0 length(Mixed_RealisticDigital)/2e3 -1 2])
subplot(2,1,1)
title('Ideal Detection')

```

```

ylabel('Binary Value')
xlabel('Time (Sec.)')
subplot(2,1,2)
title('Simulated Detection')
ylabel('Binary Value')
xlabel('Time (Sec.)')

Code2Compare = (code*2)-1;
Ideal = (double(Mixed_IdealDigital(nT))*2)-1;
Realistic = (double(Mixed_RealisticDigital(bit_locations))*2)-1;

Ideal2 = (double(Mixed_IdealDigital(nT)));
Realistic2 = (double(Mixed_RealisticDigital(bit_locations)));
XcorrIdealSignal = xcorr(Code2Compare,Ideal);
XcorrInputSignal = xcorr(Code2Compare,Realistic);

figure(3)
p = plot(1:length(XcorrInputSignal),XcorrInputSignal,'-b',1:length(XcorrIdealSignal),XcorrIdealSignal,'-r')
set(p(1),'linewidth',2);
set(p(2),'linewidth',1);
axis([0 64 -10 35])
legend('Cross-Correlation','Autocorrelation')

```

A.3 MATLAB_DPSK_Detector.m

This is the main MATLAB detection simulation. It reads a binary file containing acoustic data or loads data directly from the wav files. It demodulates the signals and cross-correlates the result with the StoredCode variable. All the simulated cross-correlation results used in chapter 4 originate from this MATLAB file.

```

clear all; close all; clc;

fs = 96e3;
fc = 12000;
CyclesPerBit = 6;
SamplesPerBit = (fs/fc)*CyclesPerBit;

StoredCode = [1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1
0 0 0 1];

% % Reading from wav file
% % file names : 'IDEAL.wav' , 'SNR12.wav', 'SNR9.wav', 'SNR6.wav',
'SNR3.wav', 'SNR0.wav', 'SNR-3.wav', 'SNR-6.wav', 'NOISE.wav'
% file = 'NOISE.wav';
% signal = wavread(file);

% Reading from Binary File

```

```

DataLocation = 'C:\(Folder Location)\';
Data = [];
for N = 2:2
    fid = fopen(strcat(DataLocation, 'data', int2str(N), '.bin'));
    TempData = fread(fid, 'uint16');
    fclose(fid);
    fid = fopen(strcat(DataLocation, 'data', int2str(N), '.bin'));
    data = fread(fid, [2, length(TempData)/2], 'uint16');
    fclose(fid);
    Data = [Data data];
end
Hydrophone = [(Data(1,:)-2^15)*10/(2^15)];
signal = Hydrophone;

t = 1/fs:1/fs:length(signal)/fs;

comparator = zeros(1,length(signal));
for i = 1:length(signal)
    if signal(i) <= 0.01
        comparator(i) = 0;
    else
        comparator(i) = 1;
    end
end
end

samples = 2:4:length(comparator);

Delayed_RealisticDigital = [zeros(1,SamplesPerBit/4)
comparator(samples)];
Windowed_RealisticDigital = [comparator(samples)
zeros(1,SamplesPerBit/4)];
Mixed_RealisticDigital = not(xor(Windowed_RealisticDigital,
Delayed_RealisticDigital));

bit_locations = 18:12:length(Mixed_RealisticDigital)-12;

Code2Compare = (StoredCode*2)-1;
DemodSeq = (double(Mixed_RealisticDigital(bit_locations))*2)-1;

XcorrSimulation = xcorr(Code2Compare, DemodSeq);
XcorrSimulationData =
XcorrSimulation(1:length(DemodSeq)+length(Code2Compare));
plot(XcorrSimulationData)

%save XcorrSimulation.mat XcorrSimulationData fs

```

A.4 ReadSerialXcorr.m

This is the MATLAB code used to open the binary files containing the hydrophone data and cross-correlation data from the hardware detector. The cross-correlation

data from the binary files are stored as a continuous serial stream until they are read and repackaged in this file.

```
clear all; close all; clc;
fs = 96e3;
DataLocation = 'C:\(Folder_Location)\';
Data = [];
for N = 2:2
    fid = fopen(strcat(DataLocation, 'data', int2str(N), '.bin'));
    TempData = fread(fid, 'uint16');
    fclose(fid);
    fid = fopen(strcat(DataLocation, 'data', int2str(N), '.bin'));
    data = fread(fid, [2, length(TempData)/2], 'uint16');
    fclose(fid);
    Data = [Data data];
end
```

end

```
[m n] = size(Data);
Hydrophone = [(Data(1,:)-2^15)*10/(2^15)];
SerialData = [(Data(2,:)-2^15)*10/(2^15)];
t=1/fs:1/fs:n/fs;

figure(1)
plot(t,Hydrophone,'r', t,SerialData,'b')
%Amount of seconds worth of Xcorr Data.
seconds = 60;
DataWindow = SerialData(1:seconds*fs);
BinaryValue = DataWindow > 1;
BinaryValueReshaped = reshape(BinaryValue, [fs/4,seconds*4]);
SerialBuffer = [];
array = zeros(1,16);
data_count = 1;
XcorrRecord = zeros(1,length(DataWindow)/48,'int8'); %48 is the
samples per bit
SerialMatrix = zeros(length(DataWindow)/48, 27);
sample = 0;

for i = 1:seconds*4
    tempBV = [SerialBuffer BinaryValueReshaped(i,:)];
    SerialBuffer = [];
    for j = 1:length(tempBV)
        array = [array(2:16) tempBV(j)];
        if isequal(array, [ones(1,15) 0])
            if length(tempBV(j:end)) <= 29
                SerialBuffer = tempBV(j:end);
                tempBV=[];
                break;
            else
                sample = sample + 1;
                SerialMatrix(sample,:) = tempBV(j+3:j+29);
                array = zeros(1,16);
                j = j + 27;
            end
        end
    end
end
```

```

        end
    end
end

for ii = 1:sample
    temp1 = [sum(SerialMatrix(ii,1:3)) sum(SerialMatrix(ii,4:7)) ...
            sum(SerialMatrix(ii,8:10))
            sum(SerialMatrix(ii,11:13)) ...
            sum(SerialMatrix(ii,14:17))
            sum(SerialMatrix(ii,18:20)) ...
            sum(SerialMatrix(ii,21:23))
            sum(SerialMatrix(ii,24:27))];
    temp2 = temp1 >= 2;
    str_DV = num2str(fliplr(temp2));
    XcorrRecord(ii) =
typecast(uint8(bin2dec(str_DV)), 'int8');
end

figure(2)
plot(XcorrRecord, 'LineWidth',1)

```

A.5 ROC_Generator.m

This file reads in all the cross-correlation data from both the hardware and MATLAB detectors and generates ROC curves from the data.

```

clear all; close all; clc;

fs = 96e3;
fc = 12000;
CyclesPerBit = 6;
SamplesPerBit = (fs/fc)*CyclesPerBit;
prefix = {'XcorrHardware_', 'XcorrSimulation_'};
file = {'CODE1.mat', 'SNR12.mat', 'SNR9.mat', ...
        'SNR6.mat', 'SNR3.mat', 'SNR0.mat', ...
        'SNR-3.mat', 'SNR-6.mat', 'NOISE.mat'};
ThresholdHigh = 32;
ThresholdLow = 10;
posDetections = zeros(length(file), ThresholdHigh - ThresholdLow
+1,2);
falDetections = zeros(length(file), ThresholdHigh - ThresholdLow
+1,2);
signal_rateINtime = .25; %seconds
bitrate = 2e3; %DPSK bits per second
signal_rateINsamples = signal_rateINtime*bitrate;

for p = 1:length(prefix)
    for f = 1:length(file)
        load(char(strcat(prefix(p), file(f))));
        [value, index] = max(XcorrData(1:signal_rateINsamples));
        detectionLocation = index;
    end
end

```

```

        firstSampleLoc = detectionLocation -
signal_rateINsamples*floor(detectionLocation/signal_rateINsamples);
        DetectionMax = floor(length(XcorrData)/signal_rateINsamples);

        for k = ThresholdLow:ThresholdHigh
            Threshold = k;
            SampleLoc = firstSampleLoc;
            for i = 1:DetectionMax
                LimitL = SampleLoc - 2;
                LimitR = SampleLoc + 2;
                [tempXcorr, tempLoc] =
max(XcorrData(LimitL:LimitR));
                SampleOffset = tempLoc-(length(LimitL:LimitR)+1)/2;
                if tempXcorr >=Threshold
                    posDetections(f,k-ThresholdLow+1,p) =
posDetections(f,k-ThresholdLow+1,p) + 1;
                end
                if SampleOffset+SampleLoc+signal_rateINsamples >
length(XcorrData)
                    break;
                else
                    SampleLoc =
SampleOffset+SampleLoc+signal_rateINsamples;
                end
            end

            detections = 0;
            for j = 1:length(XcorrData)
                if XcorrData(j) >= Threshold
                    detections = detections + 1;
                end
            end
            falDetections(f,k-ThresholdLow+1,p) = detections -
posDetections(f,k-ThresholdLow+1,p);
        end
    end
end

CM = jet(length(file));
figure(1)
for L = 1:length(file);

plot(falDetections(L,:,1)/(length(XcorrData)),posDetections(L,:,1)/De
tectionMax,'color',CM(L,:), 'marker','.', 'linewidth',2)
    hold on
end
hold off

for L = 1:length(file);
    for M = 1:32-ThresholdLow
        x = falDetections(L,M,1)/(length(XcorrData));
        y = posDetections(L,M,1)/DetectionMax;
        str = [num2str(M+ThresholdLow)];

        text(x,y,str, 'HorizontalAlignment', 'left', 'VerticalAlignment', 'top', '
Color',CM(L,:), 'FontSize',8)
    end
end

```



```

        end
    end

    %axis([0 1 0 1])
    xlabel('Probability of False Detection')
    ylabel('Probability of Positive Detection')
    legend('Ideal Signal', 'SNR = 12dB', 'SNR = 9dB', 'SNR = 6dB', 'SNR = 3dB', ...
        'SNR = 0dB', 'SNR = -3dB', 'SNR = -6dB',
        'Noise','Location','SouthEast')
    title('Hardware DPSK Detector')

figure(2)
for L = 1:length(file);

plot(falDetections(L,:,2)/(length(XcorrData)),posDetections(L,:,2)/De
tectionMax,'color',CM(L,:), 'marker','.', 'linewidth',2)
    hold on
end
hold off

for L = 1:length(file);
    for M = 1:32-ThresholdLow
        x = falDetections(L,M,2)/(length(XcorrData));
        y = posDetections(L,M,2)/DetectionMax;
        str = [num2str(M+ThresholdLow)];

        text(x,y,str,'HorizontalAlignment','left','VerticalAlignment','top','
Color',CM(L,:), 'FontSize',8)
    end
end

xlabel('Probability of False Detection')
ylabel('Probability of Positive Detection')
legend('Ideal Signal', 'SNR = 12dB', 'SNR = 9dB', 'SNR = 6dB', 'SNR = 3dB', ...
    'SNR = 0dB', 'SNR = -3dB', 'SNR = -6dB',
    'Noise','Location','SouthEast')
title('Software DPSK Detector')

```

A.6 PMF_Generator.m

This file reads in all the cross-correlation data from both the hardware and MATLAB detectors and uses this data to generate multiple probability mass functions (PMFs)

```

clear all; close all; clc;

fs = 96e3;

```

```

fc = 12000;
CyclesPerBit = 6;
SamplesPerBit = (fs/fc)*CyclesPerBit;
prefix = {'XcorrHardware_', 'XcorrSimulation_'};
file = {'CODE1.mat', 'SNR12.mat', 'SNR9.mat',...
        'SNR6.mat', 'SNR3.mat', 'SNR0.mat',...
        'SNR-3.mat', 'SNR-6.mat', 'NOISE.mat'};

ThresholdHigh = 32;
ThresholdLow = 1;
posDetections = zeros(length(file),ThresholdHigh - ThresholdLow +1);
falDetections = zeros(length(file),ThresholdHigh - ThresholdLow +1);
signal_rateINtime = .25; %seconds
bitrate = 2e3; %DPSK bits per second
signal_rateINsamples = signal_rateINtime*bitrate;

for p = 1:length(prefix)

    for f = 1:length(file)
        load(char(strcat(prefix(p), file(f))));
        [value, index] = max(XcorrData(1:signal_rateINsamples));
        detectionLocation = index;
        firstSampleLoc = detectionLocation -
signal_rateINsamples*floor(detectionLocation/signal_rateINsamples);
        DetectionMax = floor(length(XcorrData)/signal_rateINsamples);
        DetectionSample(f,:) = zeros(1,length(-20:1:32));
        SampleLoc = firstSampleLoc;
        for a = 1:DetectionMax
            LimitL = SampleLoc - 2;
            LimitR = SampleLoc + 2;
            [tempXcorr, tempLoc] = max(XcorrData(LimitL:LimitR));
            SampleOffset = tempLoc-(length(LimitL:LimitR)+1)/2;
            fixedXcorr = round(tempXcorr + 21);
            DetectionSample(f,fixedXcorr) =
DetectionSample(f,fixedXcorr) +1;
            if SampleOffset+SampleLoc+signal_rateINsamples >
length(XcorrData)
                break;
            else
                SampleLoc =
SampleOffset+SampleLoc+signal_rateINsamples;
            end
        end

    end

    DetectionSampleNormal(:, :,p) = DetectionSample/DetectionMax;
end

% Plots Hardware
figure(1)
subplot(9,1,'Gap',[0 0], 'XTickL', 'Margin', 'YTickL', 'None')
CM = jet(length(file));
for L = 1:length(file);
    subplot(1,L)

```

```

        bar(-
20:1:32,DetectionSampleNormal(L,:,1),1,'FaceColor',CM(L,:), 'EdgeColor
','k')%CM(L,:))
        axis([-20 33 0 1])
end

figure(2)
CM = jet(length(file));
for L = 1:length(file);
    bar(-
20:1:32,DetectionSampleNormal(L,:,1),1,'FaceColor',CM(L,:), 'EdgeColor
',CM(L,:))
    axis([-20 33 0 1])
    hold on
end
hold off
legend('Ideal Code 1', 'SNR = 12dB', 'SNR = 9dB', 'SNR = 6dB', 'SNR =
3dB',...
        'SNR = 0dB', 'SNR = -3dB', 'SNR = -6dB', 'White Noise',
'Location','SouthEast')
xlabel('Bits')
ylabel('Probability')

figure(3)
subplot(2,2,1)
bar(-
20:1:32,DetectionSampleNormal(1,:,1),1,'FaceColor',CM(1,:), 'EdgeColor
',CM(1,:))
axis([-20 33 0 1])
title('Ideal Signal')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,2)
bar(-
20:1:32,DetectionSampleNormal(2,:,1),1,'FaceColor',CM(2,:), 'EdgeColor
',CM(2,:))
axis([-20 33 0 1])
title('SNR = 12dB')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,3)
bar(-
20:1:32,DetectionSampleNormal(4,:,1),1,'FaceColor',CM(4,:), 'EdgeColor
',CM(4,:))
axis([-20 33 0 1])
title('SNR = 6dB')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,4)
bar(-
20:1:32,DetectionSampleNormal(9,:,1),1,'FaceColor',CM(9,:), 'EdgeColor
',CM(9,:))
axis([-20 33 0 1])
title('Noise')
xlabel('Bits')
ylabel('Probability')

```

```

% Plots Matlab
figure(4)
subplot(9,1,'Gap',[0 0],'XTickL','Margin','YTickL','None')
CM = jet(length(file));
for L = 1:length(file);
    subplot(L)
    bar(-
20:1:32,DetectionSampleNormal(L,:,2),1,'FaceColor',CM(L,:), 'EdgeColor
','k')%CM(L,:))
    axis([-20 33 0 1])
end

figure(5)
CM = jet(length(file));
for L = 1:length(file);
    bar(-
20:1:32,DetectionSampleNormal(L,:,2),1,'FaceColor',CM(L,:), 'EdgeColor
',CM(L,:))
    axis([-20 33 0 1])
    hold on
end
hold off
legend('Ideal Code 1', 'SNR = 12dB', 'SNR = 9dB', 'SNR = 6dB', 'SNR =
3dB',...
'SNR = 0dB', 'SNR = -3dB', 'SNR = -6dB', 'White Noise',
'Location','SouthEast')
xlabel('Bits')
ylabel('Probability')

figure(6)
subplot(2,2,1)
bar(-
20:1:32,DetectionSampleNormal(1,:,2),1,'FaceColor',CM(1,:), 'EdgeColor
',CM(1,:))
axis([-20 33 0 1])
title('Ideal Signal')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,2)
bar(-
20:1:32,DetectionSampleNormal(2,:,2),1,'FaceColor',CM(2,:), 'EdgeColor
',CM(2,:))
axis([-20 33 0 1])
title('SNR = 12dB')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,3)
bar(-
20:1:32,DetectionSampleNormal(4,:,2),1,'FaceColor',CM(4,:), 'EdgeColor
',CM(4,:))
axis([-20 33 0 1])
title('SNR = 6dB')
xlabel('Bits')
ylabel('Probability')
subplot(2,2,4)

```

```
bar(-
20:1:32,DetectionSampleNormal(9,:,2),1,'FaceColor',CM(9,:), 'EdgeColor
',CM(9,:))
axis([-20 33 0 1])
title('Noise')
xlabel('Bits')
ylabel('Probability')
```

APPENDIX B:

VHDL Code

The DPSK algorithm was written to the CPLD using VHDL code. All of the modifications made to the VHDL code to access the cross-correlation values and make them available for recording are documented here.

B.1 DPSKDetect.vhd

Handshaking with Cross-Correlation Process and Serial Protocol

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

entity DPSKDetect is
    port(
        CLOCK_12 : in std_logic;
        PB : in std_logic;
        LED: out std_logic;
        XLED: out std_logic;
        XcorrOut: out std_logic;
        comparator : in std_logic;
        valve : out std_logic
    );
end entity DPSKDetect;

Architecture AndrewThesis of DPSKDetect is
    signal clock_24k, clock_2k : std_logic;
    ----- SERIAL -----
    SIGNAL TX_LED : STD_logic;
    SIGNAL RequestXcorr : STD_logic;
    SIGNAL SentData : STD_logic;
    SIGNAL xcorrData : STD_logic_vector(7 downto 0);
    SIGNAL Busy : std_logic;
    SIGNAL SerialEnable : std_logic;
    SIGNAL Data2Transmit :std_logic_vector(7 downto 0);

    Type HardwareTest_state is (start, Prep_Xcorr_S1, Prep_Xcorr_S2, TX_S1, TX_S2, TX_S3);
    signal HT : HardwareTest_state;
    signal SerialData: std_logic;
begin
    XLED <= INDICATOR;
    LED <= NOT TX_LED;
    XcorrOut <= SerialData;
```

```

Clock1 : entity work.Clock24k port map (CLOCK_12, PB, clock_24k);
Clock2 : entity work.Clock2k port map (clock_24k, PB, clock_2k);
Serial_Protocol: entity work.Serial port map (CLOCK_12 => CLOCK_12,
reset => PB,
DataOut => SerialData,
Data2Transmit => Data2Transmit,
Busy => Busy,
SerialEnable => SerialEnable
);

```

----- DPSK Detection Process -----

```

.
.
.      (Removed from this code)
.
.

```

----- HARDWARE TEST CODE -----

```

RS485_Process: PROCESS(CLOCK_12, PB)
BEGIN
    IF(PB = '1') THEN
        HT <= start;
    ELSIF(CLOCK_12'event AND CLOCK_12 = '1') THEN
        CASE HT IS
            WHEN start =>
                TX_LED <= '0';
                SerialEnable <= '0';
                RequestXcorr <= '0';
                Data2Transmit <= "00000000";
                HT <= Prep_Xcorr_S1;
            WHEN Prep_Xcorr_S1 => -- handshaking with xcorr process
                RequestXcorr <= '1';
                if(SentData = '0') then
                    HT <= Prep_Xcorr_S1;
                elsif(SentData = '1') then
                    HT <= Prep_Xcorr_S2;
                end if;
            WHEN Prep_Xcorr_S2 => -- ack data received
                RequestXcorr <= '0';
                Data2Transmit <= xcorrData;
                HT <= TX_S1;
            WHEN TX_S1 =>
                TX_LED <= '1';
                if(Busy = '0') then -- handshaking with serial process
                    HT <= TX_S2;
                else
                    HT <= TX_S1;
                end if;
            WHEN TX_S2 =>
                SerialEnable <= '1';
                if(Busy = '1') then
                    HT <= TX_S3;
                else

```

```

        HT <= TX_S2;
    end if;
    WHEN TX_S3 =>
        SerialEnable <= '0';
        if(Busy = '0') then    -- delay until all data is sent
            HT <= start;
        else
            HT <= TX_S3;
        end if;
    WHEN OTHERS =>
        HT <= start;
    END CASE;
END IF;
END PROCESS;
end architecture AndrewThesis;

```

B.2 Serial.vhd

Serial Data Protocol

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

```

```

Entity Serial IS
    PORT(    CLOCK_12: IN std_logic;
            reset : in std_logic;
            DataOut : out std_logic;
            Data2Transmit: in std_LOGIC_VECTOR(7 downto 0);
            Busy : out std_LOGIC;
            SerialEnable: in std_logic
    );
END Serial;

```

ARCHITECTURE Communication OF Serial IS

```

    signal PackagedDataOut: STD_LOGIC_VECTOR(10 downto 0);
    SIGNAL BaudCLOCK: STD_logic;
    Type TX_DATA is (tx_check, tx_package, tx_send);
    signal TXD : TX_DATA;

    BEGIN
        DataOut <= PackagedDataOut(0);
        SerialBaud: entity work.CustomBaudRate port map (CLOCK_12, BaudCLOCK);

        PROCESS(BaudCLOCK, reset)
            variable bit_counter: integer range 0 to 10;
        BEGIN
            IF(reset='1')THEN
                TXD <= tx_check;
            ELSIF(BaudCLOCK'event AND BaudCLOCK = '1') THEN

```



```

case TXD is
when tx_check =>
    Busy <= '0';
    PackagedDataOut <= "1111111111";
    if(SerialEnable = '0') then
        TXD <= tx_check;
    elsif(SerialEnable = '1') then
        TXD <= tx_package;
    end if;
when tx_package =>
    Busy <= '1';
    bit_counter := 0;
    -- [end - data reversed for lsb to msb - start]
    PackagedDataOut <= "1" & Data2Transmit & "01";
    TXD <= tx_send;
when tx_send =>
    if(bit_counter >= 10) then
        TXD <= tx_check;
    else
        PackagedDataOut <= '1' & PackagedDataOut(10 downto 1);
        bit_counter := bit_counter + 1;
    end if;
when others =>
    TXD <= tx_check;
end case;
END IF;
END PROCESS;
END Communication;

```

APPENDIX C:

C-SHARP CODE

The following code was written to record the hydrophone signals as well as the serial data containing the cross-correlation results of the hardware DPSK detector.

C.1 Measurement Computing DAQ Code

This is the C Sharp code which ran on the Raspberry Pi 2 used for controlling the Measurement Computing USB-1608GX DAQ and repackaging the recorded data into binary files on a USB flash drive.

```
using System;
using System.IO;
using System.Text;
using MeasurementComputing.DAQFlex;

public class LinuxMermaid
{
    public static void Main()
    {
        DaqDevice Device;
        string[] deviceNames =
        DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);
        Device = DaqDeviceManager.CreateDevice(deviceNames[0]);
        Device.SendMessage("AISCAN:XFRMODE=BLOCKIO");
        Device.SendMessage("AISCAN:LOWCHAN=0");
        Device.SendMessage("AISCAN:HIGHCAN=1");
        Device.SendMessage("AISCAN:CAL=ENABLE");
        Device.SendMessage("AISCAN:RATE=96000");
        Device.SendMessage("AI{0}:CHMODE=SE");
        Device.SendMessage("AI{1}:CHMODE=SE");
        Device.SendMessage("AI{0}:Range=BIP10V"); // Hydrophone
        Device.SendMessage("AI{1}:Range=BIP10V"); // XcorrData
        Device.SendMessage("AISCAN:SAMPLES=0");

        int N=1;
        while (File.Exists(@"media/6A32-EAF3/ThesisData/data" +
                           N.ToString() + ".bin") == true)
        {
            N++;
        }
    }
}
```

```

FileStream stream = new FileStream("/media/6A32-
    EAF3/ThesisData/data" + N.ToString() + ".bin",
    FileMode.Create);
BinaryWriter owrite = new BinaryWriter(stream);

// Start the scan
Device.SendMessage("AISCAN:START");

double[,] scanData;

// Write the string to a file.
while (true)
{
    try
    {
        // Read and display data and statu
        for (int q = 0; q <= 59; q++) // 1 minute files
        {
            scanData = Device.ReadScanData(96000, 0);
            for (int i = 0; i <= (96000-1); i++)
            {
                for (int j = 0; j <= 1; j++) // (total channels - 1)
                {
                    if (scanData[j, i] > 65535)
                    {
                        scanData[j, i] = 65535;
                    }
                    else if (scanData[j, i] < 0)
                    {
                        scanData[j, i] = 0;
                    }
                    owrite.Write(Convert.ToUInt16(scanData[j, i]));
                }
            }
        }
        owrite.Close();
        N++;
        stream = new FileStream("/media/6A32-
            EAF3/ThesisData/data" + N.ToString() + ".bin",
            FileMode.Create);
        owrite = new BinaryWriter(stream);
    }
    catch(Exception e)
    {
        Console.WriteLine("An error occurred: '{0}'", e);
        owrite.Close();
        N++;
        stream = new FileStream("/media/6A32-
            EAF3/ThesisData/data" + N.ToString() + ".bin",
            FileMode.Create);
        owrite = new BinaryWriter(stream);
    }
}

```

```
        Device.SendMessage("AISCAN:SAMPLES=0");  
    }  
}  
}
```

BIBLIOGRAPHY

- [1] Akyildiz, Ian F., et al. "Challenges for Efficient Communication in Underwater Acoustic Sensor Networks. SIGBED Rev., 1(2):3-8, 2004
- [2] Ansari, M. Z., et. al. "Gold Code Sequences." National Institute of Technology Durgapur, 2008
- [3] Gold, R. "Optimal Binary Sequences for Spread Spectrum Multiplexing." IEEE Transactions on Information Theory, vol. 13, no. 4, p. 619-621, October 1967
- [4] Haug, O. T. "Acoustics communication for use in underwater sensor networks." Thesis, Norwegian University of Science and Technology, July 2009.
- [5] Haykin, S. "Digital Communications, 4th ed." New York: John Wiley & Sons, 2001. Print.
- [6] Kinsler, Lawrence E., et al. "Fundamentals of Acoustics, 4th Edition," John Wiley & Sons, Inc., NJ, 1999
- [7] Lurton, Xavier. "An Introduction to Underwater Acoustics: Principles and Applications," Springer, NY, 2010
- [8] MAX V Device Handbook. Altera Corporation, May 2011.
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/max-v/max5_handbook.pdf
- [9] Meel, J. "Spread Spectrum (SS) Applications." De Nayer Instituut, Belgium, 1999
- [10] Pickholtz, R. L. "Theory of Spread-Spectrum Communications -- A Tutorial." IEEE Transactions on communications, vol. 30, no. 5, May 1982
- [11] Rodriguez, F. Henriquez, N. Cruz Cortes, J.M. Rocha-Perez. "Generation of Gold-sequences with Applications to Spread Spectrm Systems"
- [12] Shanmugan, K. S. "Digital and Analog Communication Systems." New York: Wiley, 1979.
- [13] Tan F Wong, "A tutorial in Spread Spectrum & CDMA", Department of Electrical Engineering, University of Florida

- [14] USB-Blaster Download Cable User Guide. Altera Corporation, August 2015.
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_usb_blstr.pdf
- [15] USB-1608GX Multifunction DAQ Device User's Guide. Rev 1. Measurement Computing Corporation, November 2014.
<http://mccdaq.com/PDFs/manuals/USB-1608GX.pdf>
- [16] VITROVEX Glass sphere Datasheet. Nautilus Marine Service GmbH, March 2014. http://www.nautilus-gmbh.com/fileadmin/images_nautilus/002_VITROVEX/Instruments/Data_sheets/140101_VITROVEX_deep_sea_17_inch_spheres.pdf
- [17] WAV Trigger Online User Guide. Robertsonics, 2013.
<http://robertsonics.com/wav-trigger-online-user-guide/>
- [18] Xiong, F. "Digital Modulation Techniques, 2nd ed." Boston, MA: Artech House, 2006.