Project Group P16: İlker KESER 270206059 Ufuk Kağan BAYRAM 270206048 Yahya EKİN 260206054

BER Performance Analysis of Hamming Codes for QPSK in AWGN Channels

Project Final Report

Objective

The name of the project is "BER Performance Analysis of Hamming Codes for QPSK in AWGN Channels". The aim of the project is to investigate the bit error performance of Hamming encoded and QPSK modulated signals. QPSK modulation (Quadrature-Phase-Shift-Keying) is a modulation where the phase of the signal is changed to represent message symbols. The 2-bit binary data represents the phase shifts such as 45, 135, 225 and 315 degrees. With this modulation, 2 bits of data can be represented with one message symbol. In the interest of measuring the error rate, the Bit Error Rate (BER) will be used. The BER can be calculated by dividing the number of errors in incoming bits to the number of total transmitted bits. The communication channel is an additive Gaussian white noise channel, which is used in order to represent the noise which is similar to real world examples. The main purpose of the project is examining the effect of using Hamming Codes to prevent bit errors in modulated signals. The (7,4) Hamming Code adds three parity bits to 4-bit binary data to point to the wrong 1 bit in the reconstructed data. To achieve this, the transmitted data will be encoded by hamming code. After encoding, it is modulated with QPSK modulation, then it will be transmitted on the AWGN channel. On the receiver side, the incoming signal is demodulated, then the hamming coded binary bitstream is decoded. After correcting the errors with hamming decoding, we will be able to see the performance of hamming codes in terms of BER. Our project can be examined in 7 main steps:

1) Signal Generation: We generate a random binary signal whose size can be decided by the user. We need data size and input to be changeable so that we can see the results for different sizes and numbers to get more reliable outcomes.

2) Hamming Encoding: Data signal is grouped into 4 bits, then encoded with (7,4) hamming codes. Hamming encoding is implemented before modulation thus, we wanted to see the error-reductive effect of the hamming encoding on communication system.

3) QPSK Modulation: The hamming encoded data is grouped into two bits. Every two bit is represented by complex numbers. In this way, the message symbols are generated.

4) SNR Adding: To simulate the channel, AWGN noise in different magnitudes (dB) is added to QPSK modulated signal.

5) QPSK Demodulation: Noisy signal which is coming through channel is thresholded in complex form. By this, constellation diagram is obtained, and the symbols are assigned to its quadrants. Then symbols are converted to bits according to their quadrants.

6) Hamming Decoding: Incoming bits are grouped into 7-bit arrays and hamming decoded. If there is an error in data bits, one-bit errors are corrected.

7) BER Calculations: We calculate the error analysis for 2 signals. One with hamming code implemented and the other one directly modulated without hamming coding. This way we will be able to see if hamming coding does reduce the error rate and is efficient enough. It is also important to see the changes on various sized and trials for each data.

After implementing all the steps, the MATLAB implementation of hamming coding and QPSK modulation will be learned along with the communication knowledge of QPSK modulation, hamming coding and error calculations.

Background

The Background of our project in terms of communication can be categorized in 3 main steps: Hamming Encoding/Decoding, Error Detection and Correction, QPSK Modulation/Demodulation.

2.1 Hamming Encoding

Hamming Code is a technique that allows us to detect and correct errors occurred in message bits while it is being transmitted in a communication channel. Hamming code can detect up to 2-bit error and can correct a one-bit error. In Hamming code, we add redundant or parity bits to our message signal. These are extra bits that are placed in specific positions within the message bits, thus we increase the size of our message signal temporarily during transmission. These extra bits are used to detect the error bits as well as to reduce the possibility of error occurring in data bits. There are rules to choose the number of parity bits to be added and what their values should be. To find how many parity bits we should add, we use the formula:

$$2^k \geq \mathrm{k} + \mathrm{P} + 1$$

Where k is the number of data bits and P is the number of Parity bits. These parity bits are placed at the positions of powers of 2. Hamming codes are shown in the form of (k+P, k). There are various ways to apply Hamming Code. In our project we used the form (7,4), where we have 7 hamming coded bits and 4 data bits. Since the size our data signal is large, we divide it into smaller signals with sizes of 4. Then we apply Hamming code to each of these 4-bit long arrays one by one and merge them back after encoding.

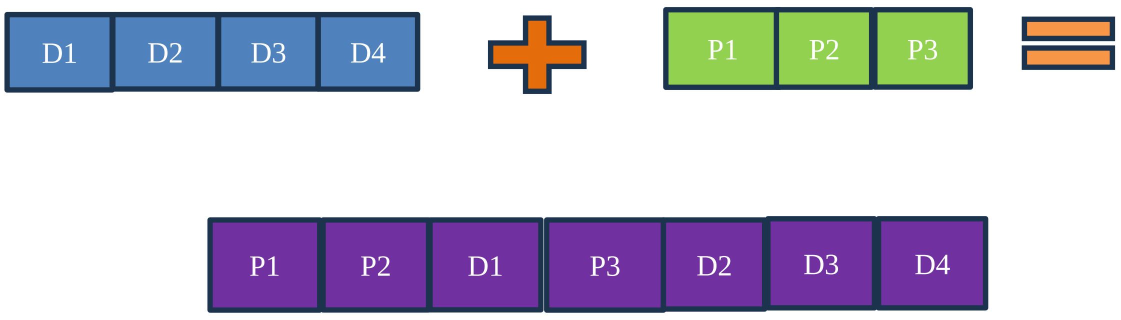

Figure 1. Hamming Encoding

Parity bits P_1, P_2 and P_3 are binary whose values are determined by data bits. We implement the take 1 and skip 1 rule while finding parity values. For example, $P_1 = D_1 \oplus D_2 \oplus D_4$,

$P_2 = D_1 \oplus D_3 \oplus D_4$ and $P_3 = D_2 \oplus D_3 \oplus D_4$. If the result is odd number, then the parity bit is 1 and if the result is even number, then the parity bit is 0.

2.2 Hamming Decoding

In the process of hamming decoding, we basically remove the parity bits we added while encoding and the size of data returns to its original size. In our project, we remove the parity bits who are placed at the power of 2. In the end, we get 2 different arrays. One is the data signal we encoded, and the other one is the parity bits we added. We need to change the resultant decoded data bits depending on the changes on parity bits if incoming data is changed because of the noise in the channel.

2.3 Error Detection and Correction

Hamming code can correct only 1 bit error in each 4-bit data. To find that error bit, the decoded parity bits and new parity bits that calculated based on decoded data bits are compared. If the new generated parity bits and encoded bits are same, the incoming data is not corrupted by noise. Otherwise, the incoming data is corrupted. To understand parity bits are different, another bit array is generated. Elements of this new array is found by calculating the XOR operation of new generated parity array and encoded parity array. To find error index, this new array is converted to decimal number (MSB-Right). This number indicates the index of the error in 7-bit data. To correct the error, the value at error index is complemented. For instance, lets assume that there is an error occurred at the demodulated signal in the index of D3.

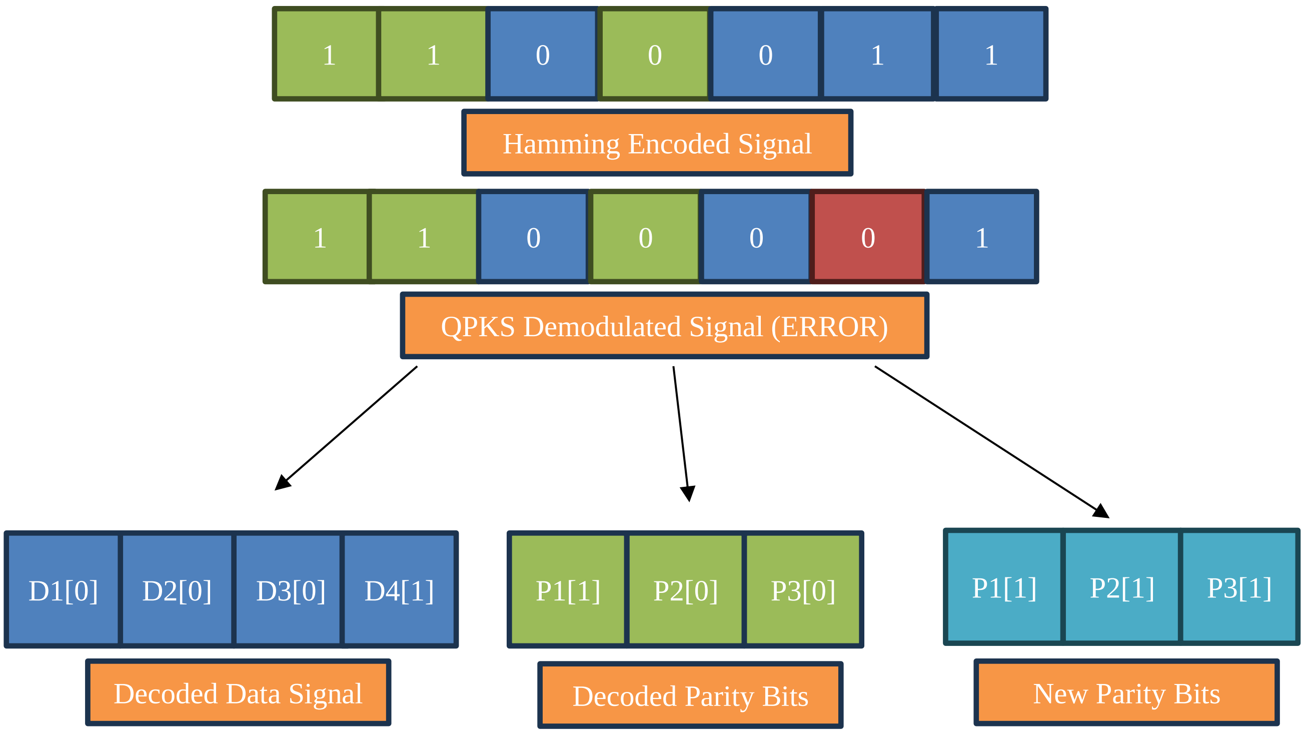

Figure 2. Hamming Decoding Process

In the figure above, the parity bits are generated according to incoming faulty data. The error index calculation is visualized in the figure below. The XOR operation is used to calculate elements of compare parity bits array.

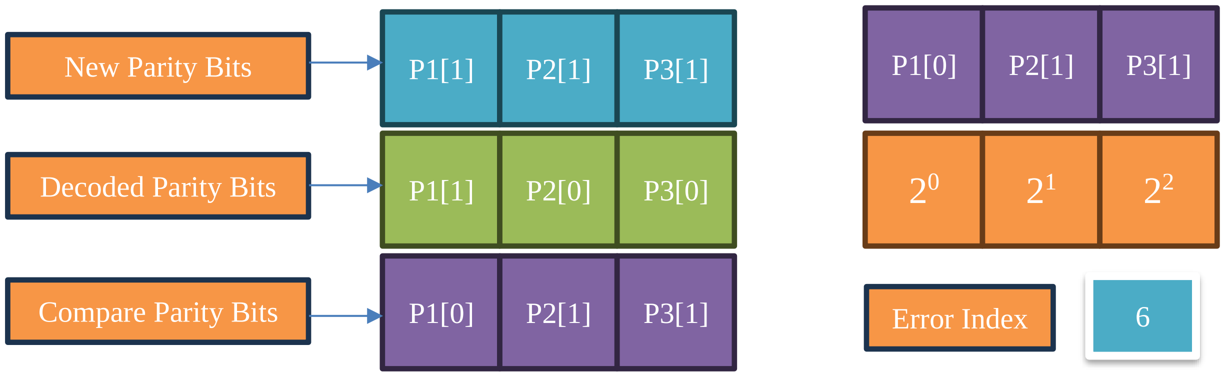

Figure 3. Error Correction Process

Then the erroneous bit at index 6 is complemented to correct the incoming data. This process is applied to all decode process. Although we can correct one-bit errors, when the error is more than one bit we can not correct the data. This is the limitation of hamming codes.

2.4 QPSK Modulation & Demodulation

QPSK is a form of PSK (Phase Shift Keying), where we modulate 2 bits at a time. In our project, we use binary bits and therefore we have 4 symbols for QPSK. They are: 00, 01, 10, 11. There are 4 possible carrier phase shifts: 45°, 135°, 225°, 315°.

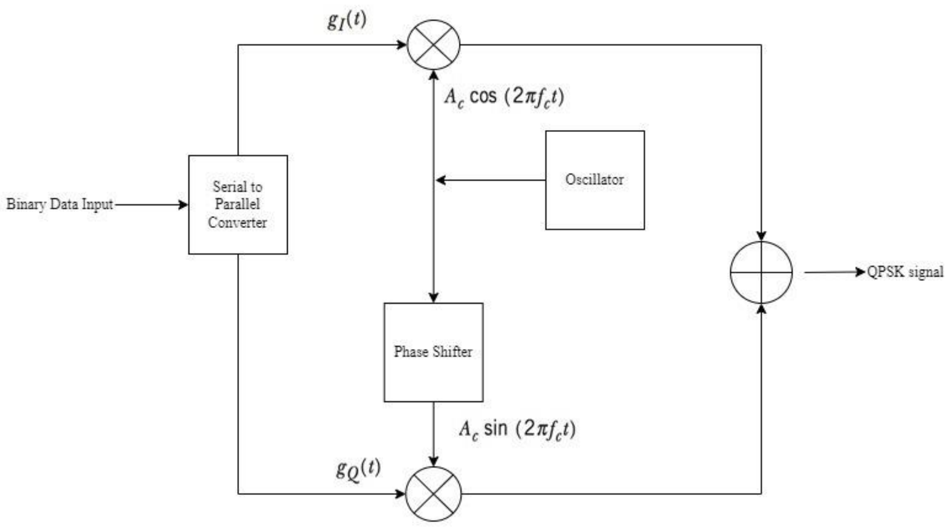

Figure 4. QPSK Modulation Scheme

When input signal goes into the modulator, bit splitter splits even and odd bits. Even bits are multiplied with the carrier signal and generates QPSK In-phase component. Odd bits are multiplied with the carrier signal and generates QPSK Quadrature component. The phase shifter shifts odd PSK by 90° before it is modulated. In the demodulation process, we use local oscillators to generate a carrier signal and multiply it with both in-phase and quadrature components. After passing through filter and integral circuits, odd and even bits are recovered simultaneously, and we get our original data.

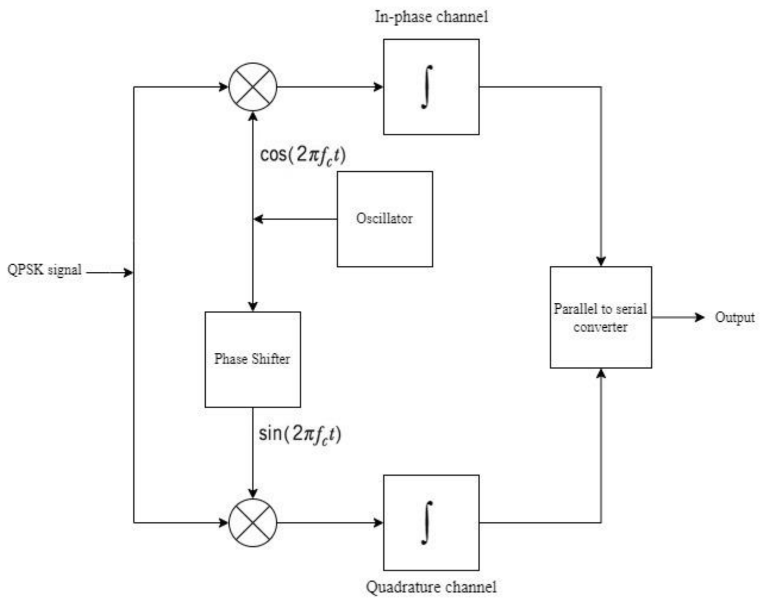

Figure 5. QPSK Demodulation Scheme

The constellation diagram is represented below. The first quadrant is assigned to symbol 11, and the second quadrant is assigned to symbol 01 according to grey code. In demodulation process, the crucial step is thresholding. Since incoming signals are subjected the noise, received complex values need to be assigned to related quadrants. With the help of this, our signal noise is minimized. There is still possibility for noise, if the noise power is so much that the incoming signal point is moved to another quadrant, then our thresholding method would not be sufficient. As a result of these operations, the quadrant of incoming points are determined and the noise is reduced.

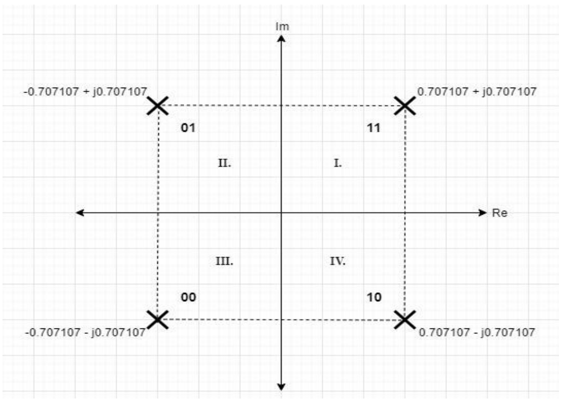

Figure 6. Desired Constellation Diagram (Grey Coded)

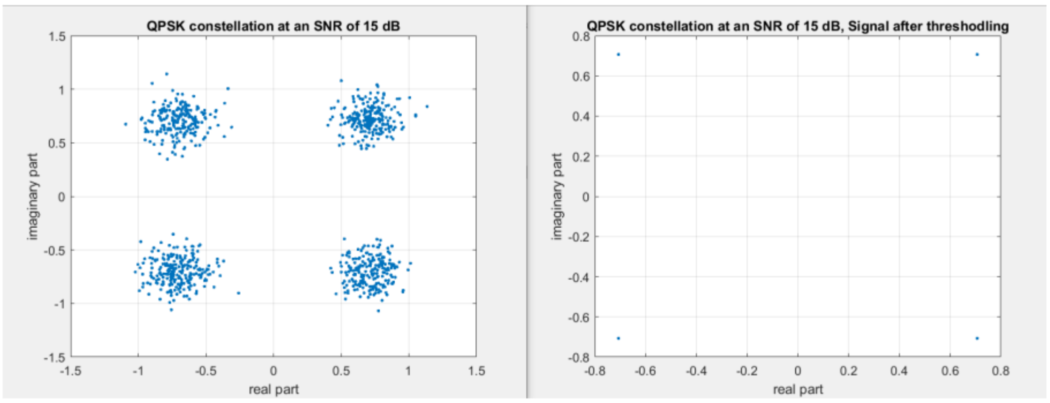

Figure 7. Effect of thresholding.

System Model & Method

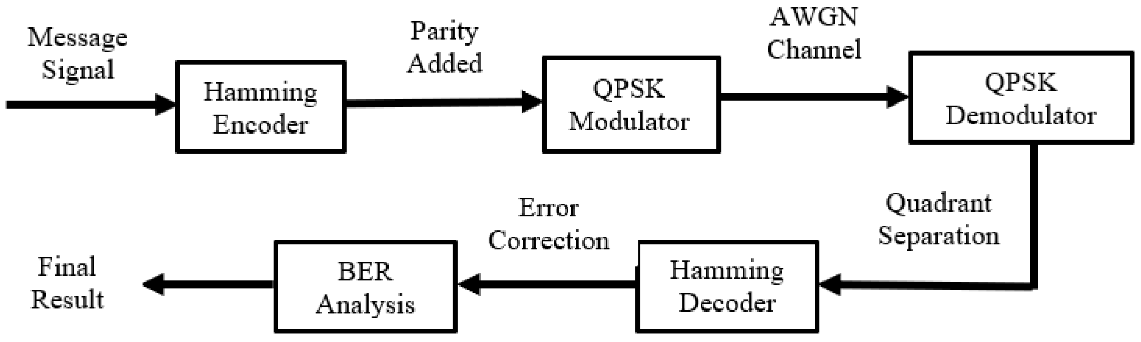

Figure 8. System Model

Our system consists of 5 main parts and 4 side parts. Main parts are: Hamming Encoder, QPSK Modulator, QPSK Demodulator, Hamming Decoder and BER Analysis. Side parts are Message signal, AWGN Channel, Error Correction and Final result examination. We will examine them one by one in the order of how they are used in project.

Message Signal: We generate our message signal as binary randomly generated array. To be able get more reliable results we need large data sizes. Thus, we generate signal with 1000 bits. We use hamming code method in the form of (7,4) array size. This means that data signals are 4-bit long, and we need to divide our input signal into 4-bit sub signals. To do that, we generate 4-bit long subarrays before the input signal goes into hamming encoder function and each subarray is encoded one by one. Afterwards, we merge 7-bit encoded arrays to get our encoded signal as 1 signal.

Hamming Encoder: Hamming Encoder function takes 4-bit subarray data as input and turns it into 7-bit arrays by adding parity bits. Each subarray gets different parity bits since their values are decided by data bits and our data is generated randomly. We get 2 outputs from this function, one is 7-bit hamming encoded array and the other one is 3-bit parity arrays.

QPSK Modulator: QPSK Modulator function takes in hamming encoded arrays as input. Data bits are given quadrants for every 2-bit. Since there are 4 possible combinations for 2-bit binary numbers, there are 4 quadrants. Quadrants represent imaginary and reel values on the coordinate system. Therefore our message symbols are generated. The help of this modulation is we sent 2 bits in one message symbol.

AWGN Channel: We added different SNR values in wide ranges to see at which SNR value error bits stops occurring and to make comments on the behavior of error rate in changing SNR. Currently we use SNR from -5 to 10. Because error rate disappears when SNR surpasses 10 and behaves similar in SNR values lower than -5. To perform multiple iterations based on snr, we generate snr array.

QPSK Demodulator: Our QPSK modulation function takes message symbols subjected to noise as an input. Three operation is performed under this function. First, the incoming symbols are thresholded Second, the thresholded symbols are assigned to belonging quadrants and their desired complex values according to desired constellation diagram. Third, the quadrants are converted to message bits. When we add to much SNR to the modulated signal, point in every region moves from its initial point. And if added SNR is large enough to move it to out of its region, then an error occurs during the demodulation process since demodulator function will place it in the wrong region. This is the main reason for errors in this modulation.

Hamming Decoder: Hamming Decoder function takes in QPSK demodulated bits as input and separates it into data and parity bits, doing exactly opposite of what hamming encoder does. Data bits we got in this function could include error bits in them.

Error Correction: We generate another parity bits from the data bits that we generated in this function. Now there are two array of parity bits. One from hamming decoder function and one is generated from the encoded data bits. We compare these two parity arrays by XOR operation and generate a binary number (right-MSB). If bits in same indexes are same, we place 0 and if they are different, we place 1. Then we convert this binary number to decimal to see which data bit has error on it. And lastly, we fix the error bit by taking the complement of error index.

BER Analysis: To be able to analyze accurately, we put entire code into for loop and get results for various trials. We can increase the number of trials to get better results. Another crucial part is to compare our results with build-in function result. Since build-in functions give more accurate results, we can take it as reference and aim to get closer results to it. The final graph consists of multiple trails for wide range of SNR values. To calculate BER, we divided the number of errors in our received bit to length of the message. To see how system behaves in different SNR’s, we generated an SNR array, and we perform error calculation per SNR value on number of trial times. This way we aim to get accurate results.

Final Result: In the final output of our project, we can see the effect of hamming coding on reducing the error rate. We plotted both hamming coded and not hamming coded function on the same figure to compare them properly. When number of trials and message signal size surpasses large enough values, the results stop changing. By trying various values multiple times, we decided to use 1000-bit array size and 10 SNR as our max limits. Even though error rate start reducing a little bit later then it is supposed to compared to build-in functions, we get almost same results in the end. When we look at the complete graph, error gap is large enough to make hamming coding efficient for error rate reduction.

Result & Discussion

To be able to analyze the results we printed results with different data sizes and trials. Then we plotted two outputs on the same graph for each result. One with hamming code applied and one without hamming code applied. Now, we will analyze BER for 1000-bit data size and different number of trials. We decided to choose the range for SNR from -5 to 10 because in this interval, it is easy to see error rate changes and the effect of hamming code. Also, after 10 SNR, the possibility of error occurring becomes near zero thus, the graph stops after that.

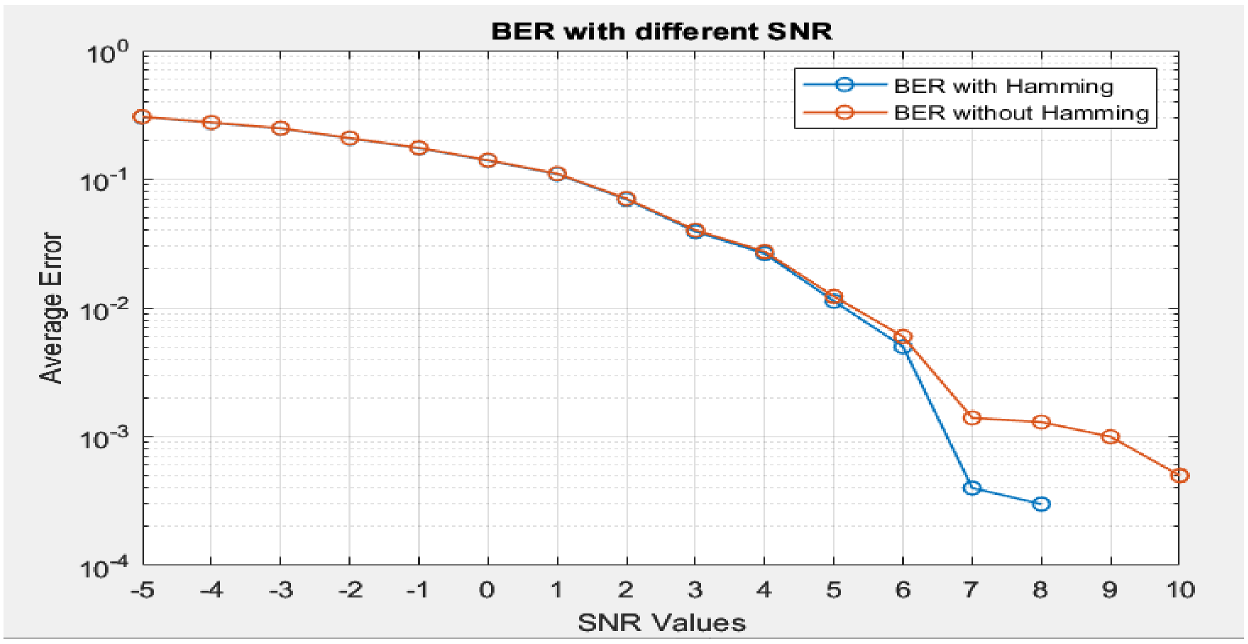

Figure 6. **Figure 9.** BER Analysis for 1000-bit data size and 10 Trials

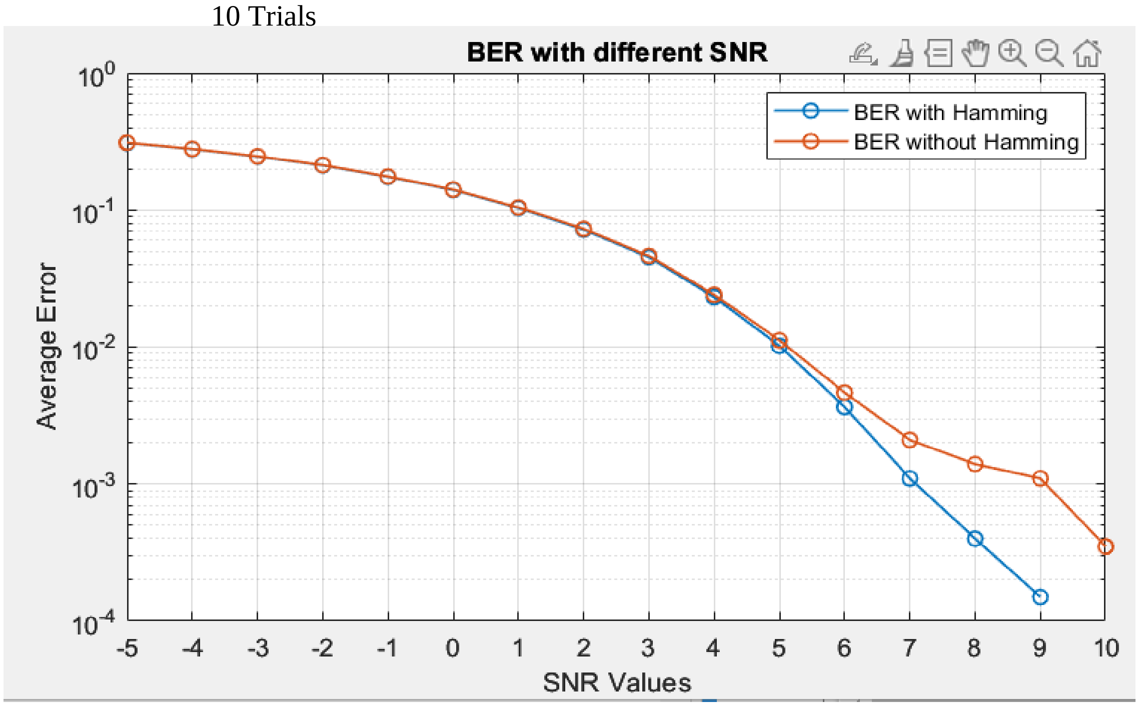

Figure 7. **Figure 10**. BER Analysis for 1000-bit Data Size and 20 Trials

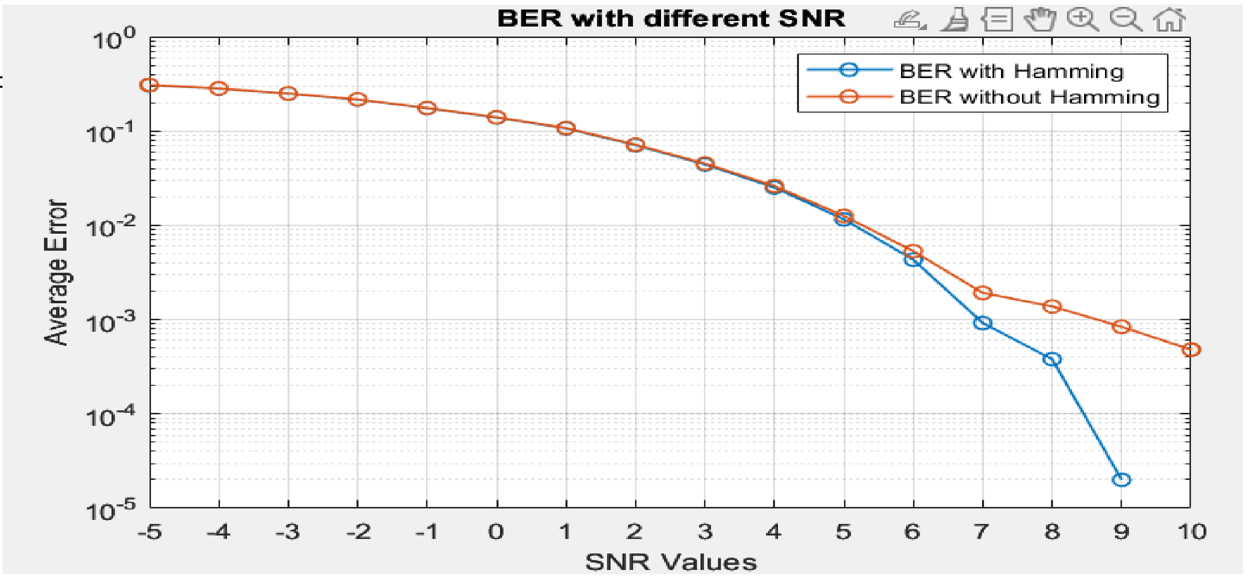

Figure 8. Figure 11. BER Analysis for 1000-bit Data Size and 50 Trials

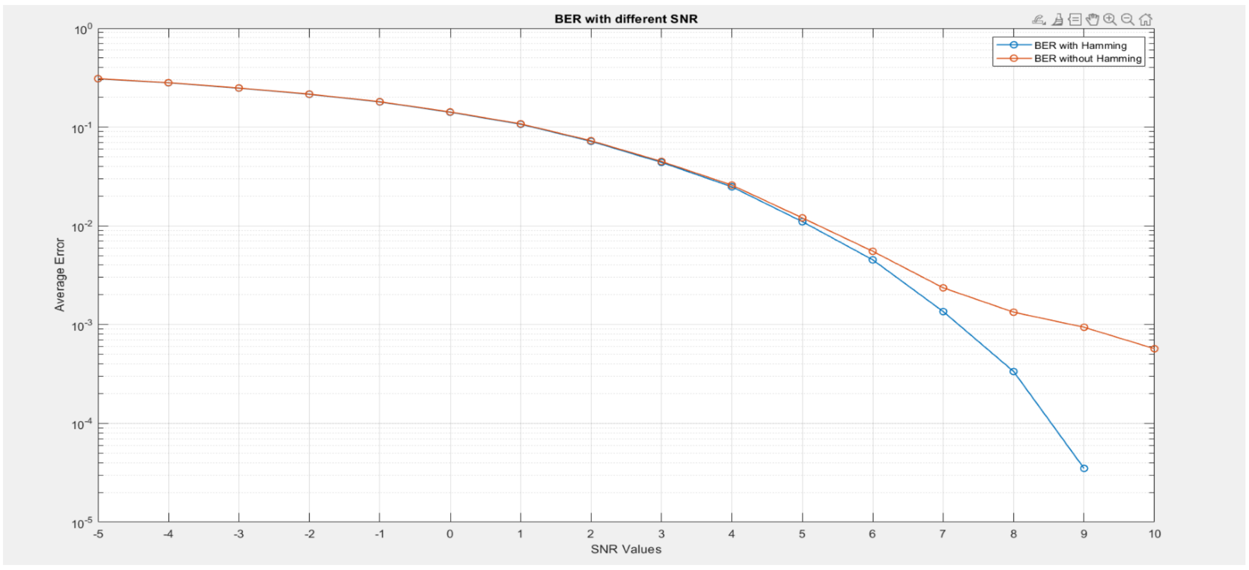

Figure 12. BER Analysis for 1000-bit Data Size and 200 Trials

We can see that as the number of trial increases, the lines become smoother. There is an error gap in each for graph, but higher number of trials shows better results. After 100 trails, the results stop changing and we keep getting the result in figure 10. The blue line represents hamming coded result, and the red line represents not hamming coded result. We can see the impact of the hamming code when we look at where the lines ended. Blue line has much lower error rate than red line and this is because hamming code eliminates 1 bit of error in every 4 bit thus reducing the error rate. When we compare this with build-in results, our error rate starts reducing later than build-in does, but we get lower reduction of error in the end. Thus, our code shows better results in higher sizes and larger values of SNR. To make comments about data size, we will compare the latest result both with higher and lower data sizes and same trials, this way we can see the effect of data size clearly.

Now we can comment of how data size effects the result. We will compare 1000-bit data size with 200-bit data size for the same trial of 200.

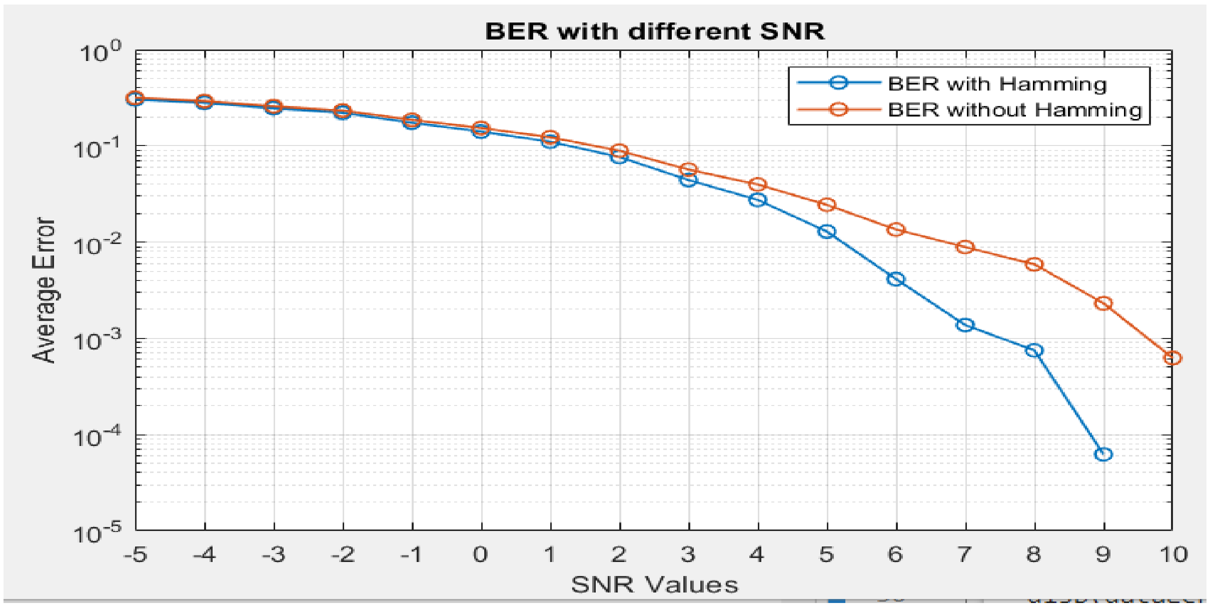

Figure 13: BER Analysis for 200-bit Data Size and 200 Trials

In 200-bit size, we get the closest result in SNR values between 0 and 5 but we also get lower gap in error rate in the end. Thus, we should still prefer higher data sizes for more reliable results. We can still observe the gap throughout each SNR for both 200 and 1000 sizes, but in the end, we get higher error rate reduction in larger data size.

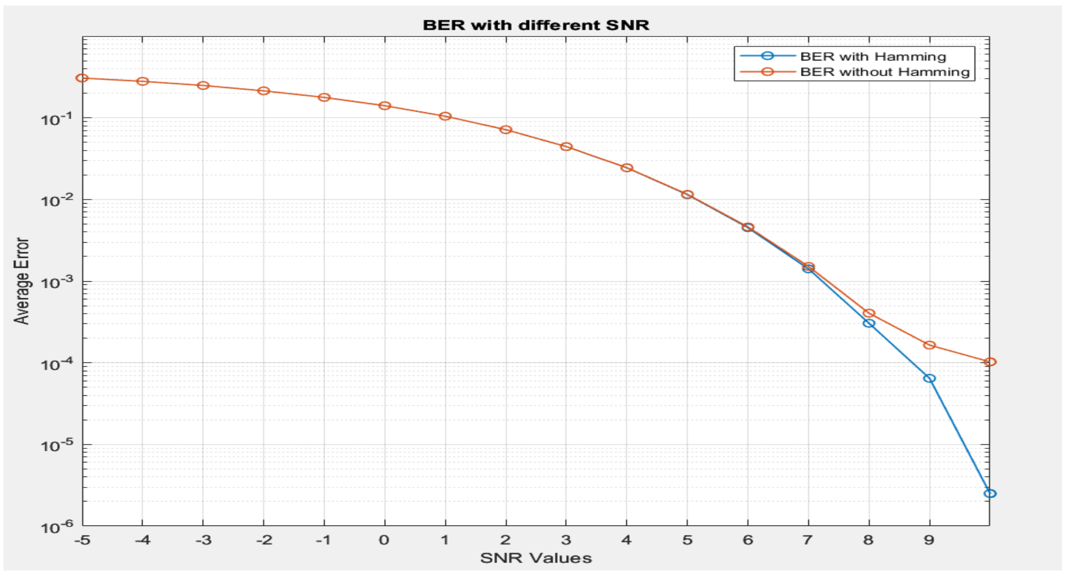

Figure 9. **Figure 14.** BER Analysis for 10000-bit Data Size and 200 Trials

We can see that 10 times higher data rate resulted in slightly higher reduction in error rate, this is because 200- and 1000-bit sizes are large enough to make comments. The reason why we can't get lower error rate is because hamming code can only correct one error bit at a time.

Lastly, we compared our functions error correcting performance with MATLAB's built-in functions. To achieve that, we generated another MATLAB file that contains same code as our code except the built-in functions for hamming encode and decode such as "encode" and "decode" functions. We tested both systems under same conditions. The trial number is 200 and message length is 200. Both cases have same SNR array. The results can be seen from the figures below.

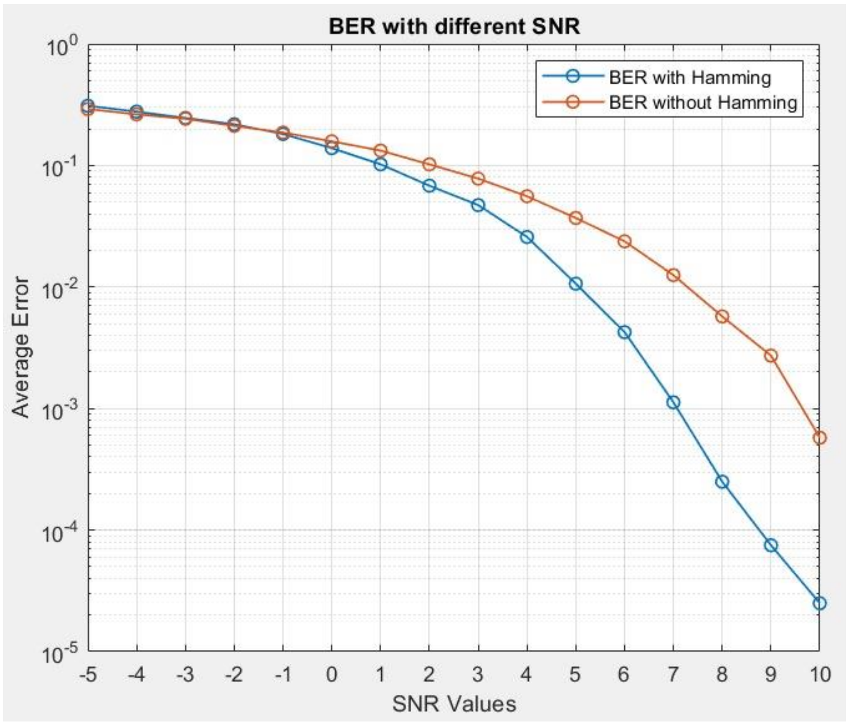

Figure ~~13.~~15. BER analysis of MATLAB built-in functions. Message size 200 bit. Trial 200.

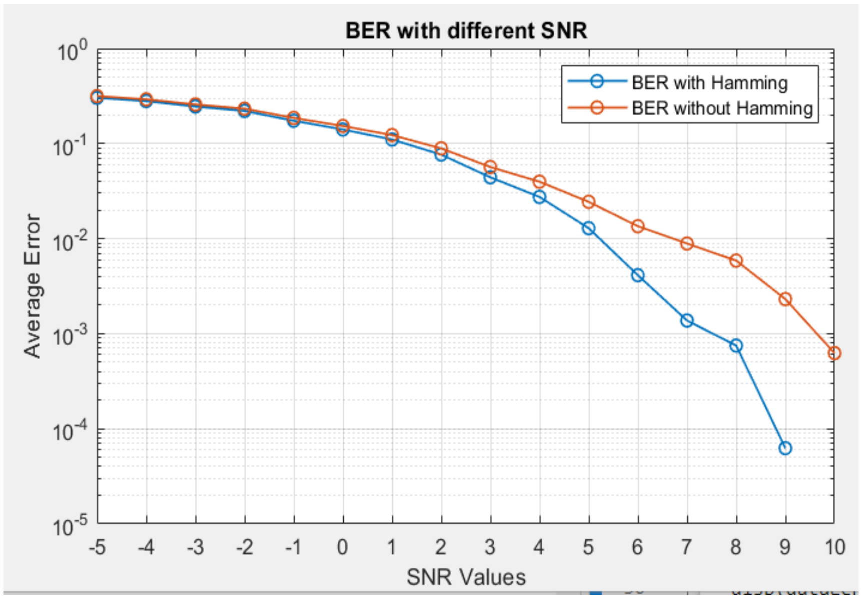

Figure 16. BER analysis of MATLAB built-in functions. Message size 200 bit. Trial 200.

Conclusion

In conclusion, our project BER Performance Analysis of Hamming Codes for QPSK in AWGN Channels, allowed us to have deeper understanding in communication knowledge of hamming code and QPSK modulation as well as developed our skills in MATLAB. This project helped us to learn importance of coding and error rate. We can implement different types of error correction to reduce the error rate. In communications, reducing the error rate is our priority thus we implement our ways of encoding to reduce it. Hamming Code is one of them. In Hamming Code, we add additional bits called parity bits into our message signal. Then we examine the changes in parity bits occurred in throughout the modulation to detect and correct the error bits. Hamming code can only correct one bit at a time. Therefore, there are some conditions where, hamming code suits more then other techniques. Hamming code itself can be implemented in different ways. We can choose how many data bits we should use and add parity bits according to it. We decided to use 4 data bits for hamming code. Since 4 data bits gets 3 additional parity bits the ratio of parity to data is high and data size is low makes it simple and effective. The resulted graphs show that hamming code can effectively reduce the error rate in certain conditions. Parameters such as data size and trial number change the accuracy of error rate. We increased and decreased these parameters to observe their impact on error rate.