RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

Assignment

Course Code	CSC212A
Course Name	Data Communication
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	04/2017
Course Leader(s)	Dr. Rinki Sharma

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	04/2017
Course Code	CSC212A		
Course Title	Data Communication		
Course Date		to	
Course Leader	Dr. Rinki Sharma		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

List Of Figures

1 Question 1

Solution to Question No. 1 Part A

1.1 Introduction to High Rate Modulation

Modulation is the process of varying one or more properties of a periodic waveform, called the carrier signal, with a modulating signal that typically contains information to be transmitted. It can refer to both Digital and Analog Modulation, each of them consists a vast number of techniques which would become out of scope of this assignment, hence we'll limit our study to Digital Modulation for the sake of simplicity,

In digital modulation, an analog carrier signal is modulated by a discrete signal. Digital modulation methods can be considered as digital-to-analog conversion and the corresponding demodulation or detection as analog-to-digital conversion. The changes in the carrier signal are chosen from a finite number of M alternative symbols (the modulation alphabet).

Some Fundamental Digital Modulation Techniques are,

1. PSK (Phase-Shift-Keying): a finite number of phases are used
2. FSK (Frequency-Shift-Keying): a finite number of frequencies are used
3. ASK (Amplitude-Shift-Keying): a finite number of amplitudes are used
4. QAM (Quadrature-Amplitude-Modulation): a finite number of at least two phases and at least two amplitudes are used.

We'll concentrate on one of these to describe how high rate modulation would work in that modulation technique, for that QAM is a good example, since it is used in standard 802.11 Wireless Transmission like WiMax, WCDMA, HSDPA, HSPA.

Albeit each of the above techniques can be used to do High Rate Modulation, but the most efficient among them is QAM, also the technique is relatively widely used as said, and has far more real-life applications.

1.2 Benefits and limitations of high order modulation techniques

To simulate a Quadrature Amplitude Modulation we use MATLAB, here we try to transmit a random stream of data and plot the received signal, then we add some Gaussian White Noise to it to simulate how the technique works when there is noise in the surroundings. The Channel Model is AWGN (Additive White Gaussian Noise).

1.2.1 MATLAB Example

```
M = 16;                    % Size of signal constellation
k = log2(M);               % Number of bits per symbol
n = 30000;                 % Number of bits to process
numSamplesPerSymbol = 1;   % Oversampling factor
rng default                % Use default random number generator
dataIn = randi([0 1],n,1); % Generate vector of binary data
dataInMatrix = reshape(dataIn,length(dataIn)/k,k);   % Reshape data into binary k-tuples,
k = log2(M)
dataSymbolsIn = bi2de(dataInMatrix);                 % Convert to integers
dataMod = qammod(dataSymbolsIn,M,'bin');         % Binary coding, phase offset = 0
dataModG = qammod(dataSymbolsIn,M); % Gray coding, phase offset = 0
EbNo = 20;
snr = EbNo + 10*log10(k) - 10*log10(numSamplesPerSymbol);
receivedSignal = awgn(dataMod,snr,'measured');
receivedSignalG = awgn(dataModG,snr,'measured');
sPlotFig = scatterplot(receivedSignal,1,0,'.');
hold on
scatterplot(dataMod,1,0,'k*',sPlotFig)
grid on
```

The Code has been Explained in Appendix B

Now we see how the signal is transmitted in a SNR of 10dB and SNR of 30dB

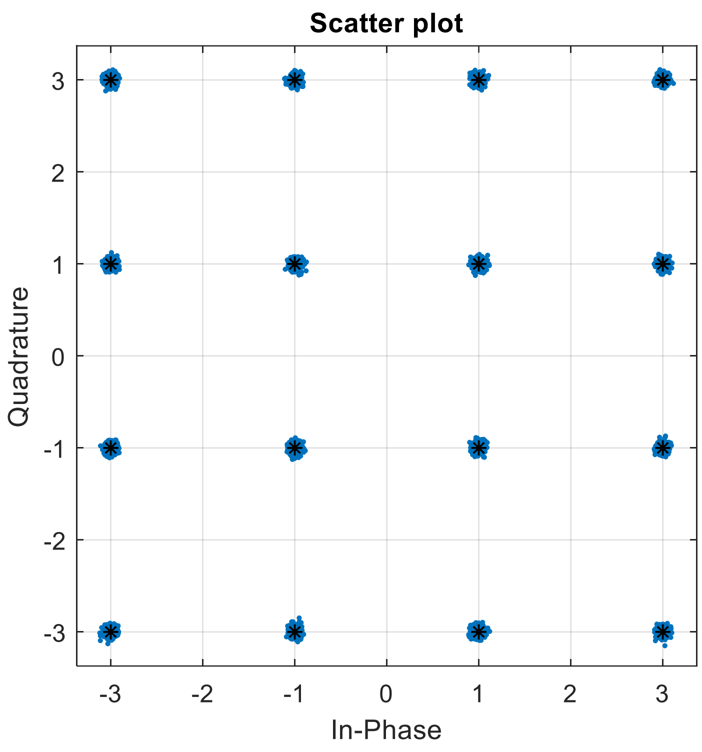


Figure 1-1 When Eb/No = 30dB

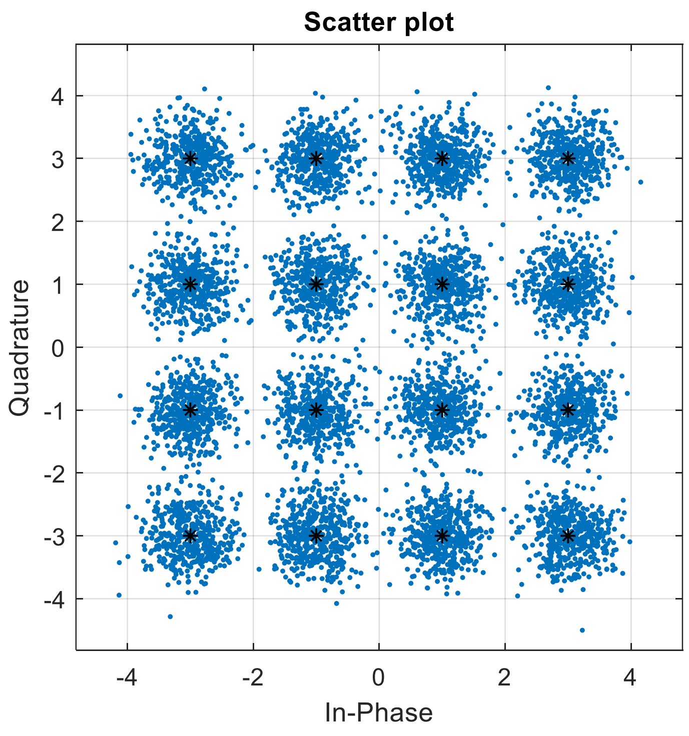


Figure 1-2 When Eb/No = 10dB

Observations:

Here we have considered QAM16 and transmit data over a White Noise Channel, as we observe that decreasing the SNR, i.e. adding more noise to the signals the signal received at the destination will have more trouble distinguishing between the different alphabet. As we can see that the data cloud is more spread when the SNR is 10dB.

While when we have an SNR of 30dB the signal is clearer and the different QAM points can be distinguished uniquely.

In moving to a higher order QAM constellation (higher data rate and mode) in hostile RF/microwave QAM application environments, such as in broadcasting or telecommunications, multipath interference typically increases. There is a spreading of the spots in the constellation, decreasing the separation between adjacent states, making it difficult for the receiver to decode the signal appropriately. In other words, there is reduced noise immunity.

1.2.2 Benefits

- The major benefit of QAM modulation variants is efficient usage of bandwidth. This is due to the fact that QAM represent a greater number of bits per carrier. For example, 16QAM maps 4 bits per carrier, 64QAM maps 6 bits per carrier, 256QAM maps 8 bits per carrier and so on.
- QAM increases the efficiency of transmission of radio

1.2.3 Limitations

- As states are closer as shown in the figure, QAM modulation is more susceptible to the noise. Due to this QAM receiver is more complex compare to receivers of other modulation types.
- As QAM uses amplitude component of signal to represent binary data, linearity needs to be maintained and hence linear amplifier is needed which consumes more power.

1.3 Stance taken with justification

I do not agree with the topic that High Rate Modulation are the best option for "reliable" communication, since Noise is an inevitable factor of the environment, it will be present regardless, when QAM is done in such environments, it is bound to drop packets, and the QAM will have to automatically reduce the number of alphabet, eventually dropping the data rate, then it's not High Rate Anymore.

High Rate Modulation should be preferred when transmitting over shorted lengths and less noisy environments, such as WiFi, where our range is of few meters, but for farther transmissions such as communicating with the ISS (International Space Station) as an example, the conventional methods are best suited.

2 Question 2

Solution to Question 1 Part B

2.1 Algorithm for computation of Hamming Distance

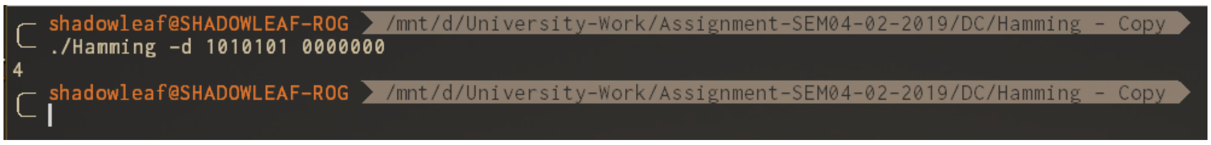
```
HAMMING_DISTANCE(str1, str2)

1. hdistance = 0

   // run through the entire string

2. for i = 0 to len(str1)

3.     if str1[i] != str2[i]

4.         hdistance = hdistance + 1

5. return hdistance
```

2.2 Implementation

```
lld hamming_distance_string(string str1, string str2) {
    /* Check if the length of the strings are same */
    if (strlen(str1) != strlen(str2))
        return -1;

    lld len = (lld)strlen(str1);
    lld hdistance = 0;

    /* loop through the entire string and count the differences */
    for (lld i = 0 ; i < len ; i++) {
        if (str1[i] != str2[i])
            hdistance++;
    }

    return hdistance;
}
```

Refer Appendix for complete implementation in a C program.

2.3 Testing of Implementation



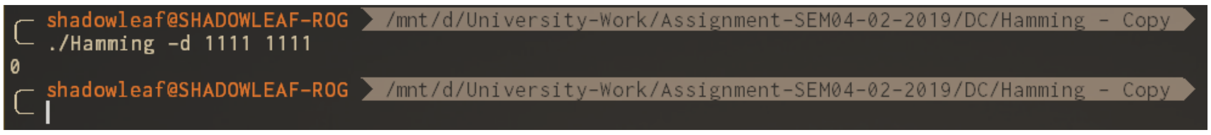
```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
./Hamming -d 1010101 0000000
4
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
```

Data1 = 1010101 Data2 = 0000000

To calculate it manually we XOR the data to get

1010101

The number of 1's in the above are 4, hence the Hamming Distance is 4.

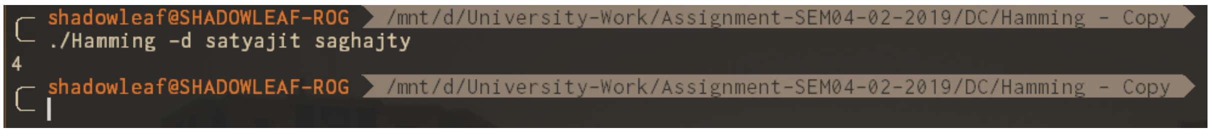


```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
./Hamming -d 1111 1111
0
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
```

Data1 = 1111 Data2 = 1111

XOR(Data1, Data2) = 0000

The number of 1's in the above are 0, hence the Hamming Distance is 0.



```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
./Hamming -d satyajit saghajty
4
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
```

The Algorithm also works strings with alphabets, it counts the number of characters that are different in the two strings.

3 Question 3

Solution to Question 2 Part B

3.1 Algorithm for computation of Hamming (7, 4) code

HAMMING_CODE(str)

```
1.  len = len(str)
2.  p_n = 0
// compute the number of parity bits using the formula 2^r > m + r + 1
3.  for i = 0 to len > (2^i - (i+1))
4.     p_n = p_n + 1
// code_length = parity_bits + length of data
5.  c_l = p_n + len
6.  j = 0, k = 0
7.  for i = 0 to c_l
8.     if i == 2^k-1
// fill the parity bit position with 0's just for initialization
9.        res_code[i] = 0, k++
10.    else
// fill the data bit positions with the data
11.       res_code[i] = str[j], j++
12. for i = 0 to p_n
// find the position of the parity bit
13.    pos = 2^i
// calculate the parity bit and store the result
14.    res_code = calc_parity(res_code, pos, c_l)
15. return res_code
```

CALC_PARITY(code, pos, c_l)

```
1.  count = 0, i = pos -1
// loop from pos-1 to max length of the code
2.  while i < c_l
```

```
3.    for j = i to pos+i
// count the number of 1's in the code for even parity of 1's
4.        if code[j] == 1
5.            count = count + 1
   // i is the first position of the data bit, the next first position is current
   i + twice the position.
   // example for the first parity bit, the first position is 2, the next
   position is 2*2, then 4*2, then 8*2 and so on.
6.    i = i + 2 * pos
   // if there is even parity then return 0, the parity bit
7. if count%2 == 0
8.    return 0
   // else return 1
9. return 1
```

ERROR_DETECT(code, p_n)

```
1. error_pos = 0
   // check the parity bits, p_n = 2^i, where i goes from 0 to p_n
2. for i = 0 to p_n
3.    pos = 2^i
   // calculate the parity of the current parity bit using calc_parity algorithm
4.    val = calc_parity(code, pos, c_l)
   // if the parity bit is not 0 then there is an error
5.    if val != 0
6.        error_pos = error_pos + pos
   // if there was no error then the code is verified.
7. if error_pos == 0
8.    display "CODE VERIFIED"
9. else
10.   display "ERROR AT BIT " + error_pos
```

3.2 Implementation to compute Hamming (7, 4) code

3.2.1 Hamming Code C code snippet

```c
string hamming_code(const char* str) {

    string res_code = calloc(1024, sizeof *res_code);
    lld len = (lld)strlen(str);

    lld p_n, c_l, i, j, k;

    for(i = 0l, p_n = 0l;
        len > ((1<<i) - (i+1));
        p_n++, i++);

    c_l = p_n + len;

    for(i = j = k = 0l; i < c_l; i++) {
        if (i == ((1<<k)-1)) {
            *(res_code + i) = '0';
            k++;
        } else {
            *(res_code + i) = *(str + j);
            j++;
        }
    }

    for (i = 0l ; i < p_n ; i++) {
        lld pos = 1 << i;
        *(res_code + pos - 1) = calc_parity(res_code, pos, c_l);
    }

    *(res_code + c_l + 1) = '\0';

    return res_code;
}

void error_detect(const char* code, lld p_n) {
    lld c_l = (lld)strlen(code);
//    printf("%llu\n", c_l);
    lld error_pos = 0;
    for (lld i = 0; i < p_n ; i++) {
        lld pos = 1 << i;
        char value = calc_parity(code, pos, c_l);
        if (value != '0') {
            error_pos += pos;
        }
    }

    if (error_pos == 0) {
        printf("CODE VERIFIED OK\n");
    } else {
        printf("ERROR AT BIT %llu\n", error_pos);
    }
}
```

```c
char calc_parity(const char* code, lld pos, lld c_l) {
    lld count = 0, i = pos - 1;

    while (i < c_l) {
        for (lld j = i ; j < pos + i ; j++) {
            if (code[j] == '1')
                count++;
        }

        i += 2*pos;
    }

    if (count%2 == 0)
        return '0';
    else
        return '1';
}
```

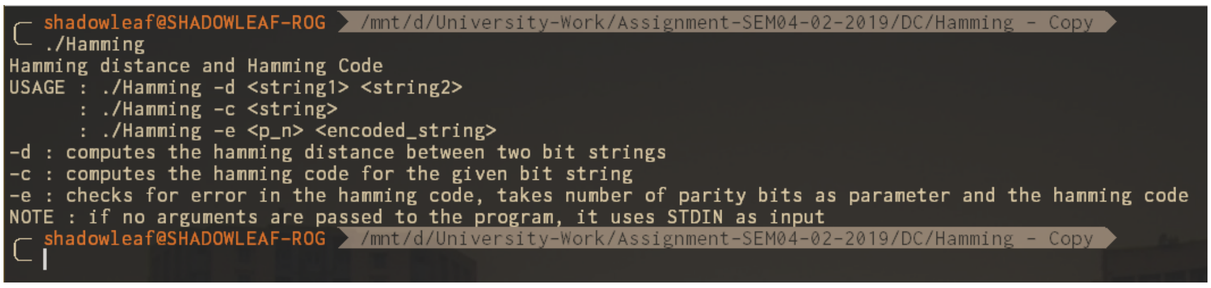
The Algorithm for the code with proper comments and explanation is done in 3.1.

Refer to the Appendix for complete code.

See Appendix for complete implementation in a C program

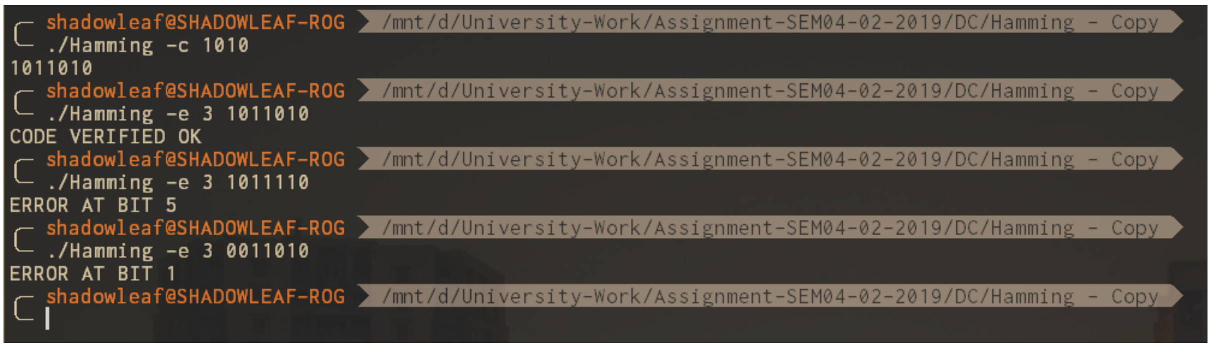
3.3 Validation of the implementation

3.3.1 Example 1



```
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
└ ./Hamming
Hamming distance and Hamming Code
USAGE : ./Hamming -d <string1> <string2>
      : ./Hamming -c <string>
      : ./Hamming -e <p_n> <encoded_string>
-d : computes the hamming distance between two bit strings
-c : computes the hamming code for the given bit string
-e : checks for error in the hamming code, takes number of parity bits as parameter and the hamming code
NOTE : if no arguments are passed to the program, it uses STDIN as input
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
└ |
```

If no arguments are passed to the program the default help documentation is displayed that describes the different arguments that it accepts and the description of that argument.


```
╭ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
╰ ./Hamming -c 1010
1011010
╭ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
╰ ./Hamming -e 3 1011010
CODE VERIFIED OK
╭ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
╰ ./Hamming -e 3 1011110
ERROR AT BIT 5
╭ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
╰ ./Hamming -e 3 0011010
ERROR AT BIT 1
╭ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
╰ |
```

Hamming Code Calculation:

Here the data taken is: 1010

Let's first calculate the Hamming Code for this manually.

P1	P2	D1	P4	D2	D3	D4
X		X		X		X
	X	X			X	X
			X	X	X	X

P1 = D1 D2 D4 | P1

= 1 0 0 | 1

= 1

P2 = D1 D3 D4 | P2

= 1 1 0 | 0

= 0

P4 = D2 D3 D4 | P4

= 0 1 0 | 1

= 1

Hence the Hamming Code is P1 P2 D1 P4 D2 D3 D4 = 1011010, which is the same as that we obtained from our program.

Error Detection:

Received Data: 101110,

P1 = 1, P2 = 0, D1 = 1, P4 = 1, D2 = 1, D3 = 1, D4 = 0

Now we extract the parity bits P1 P2 P4 = 1 0 1

P1 = D1 D2 D4 | P1

= 1 1 0 | 1

= PARITY ERROR

P2 = D1 D3 D4 | P2

= 1 1 0 | 0

= OK

P4 = D2 D3 D4 | P4

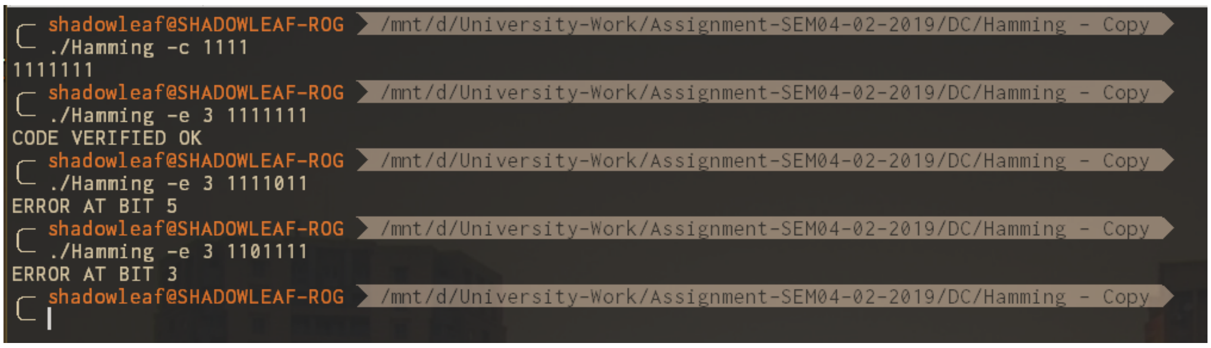
= 1 1 0 | 1

= PARITY ERROR

ERROR Position: P4 P2 P1 = 101 = 5

Hence, we have a bit error at position 5.

3.3.2 Example 2



```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
  ./Hamming -c 1111
1111111
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
  ./Hamming -e 3 1111111
CODE VERIFIED OK
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
  ./Hamming -e 3 1111011
ERROR AT BIT 5
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
  ./Hamming -e 3 1101111
ERROR AT BIT 3
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/Assignment-SEM04-02-2019/DC/Hamming - Copy
  |
```

Similarly, the code was run for data 1111 and the output was verified.

Bibliography

1. https://in.mathworks.com/help/comm/gs/compute-ber-for-a-qam-system-with-awgn-using-matlab.html

Appendix A

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "hamming.h"
#include "input_helper.h"

void help();
string prog_name;
/* argc : argument count
 * argv : argument values*/
int main(int argc, char** argv) {
    prog_name = argv[0];
    if (argc >= 3) {
        switch(*(argv[1]+1)) {
            case 'd': {
                lld distance = hamming_distance_string(argv[2], argv[3]);
                printf("%llu\n", distance);
            } break;
            case 'c': {
                string encoded = hamming_code(argv[2]);
                printf("%s\n", encoded);
            } break;
            case 'e': {
                lld p_n = atoi(argv[2]);
                error_detect(argv[3], p_n);
            } break;
            default: {
                help();
            }
        }
    } else if (argc == 2) {
        switch (*(argv[1]+1)) {
            case 'd': {
                string str1, str2;
                get_line(&str1); get_line(&str2);
                lld distance = hamming_distance_string(str1, str2);
                printf("%llu\n", distance);
            } break;
            case 'c': {
                string str;
                while (get_line(&str) >= 0) {
                    string encoded = hamming_code(str);
                    printf("%s\n", encoded);
                }
            } break;
            default: {
                help();
            }
        }
    } else {
        help();
    }
    return 0;
}

void help() {
    printf("Hamming distance and Hamming Code\n"
           "USAGE : %s -d <string1> <string2>\n"
           "       : %s -c <string>\n"
           "       : %s -e <p_n> <encoded_string>\n"
           "-d : computes the hamming distance between two bit strings\n"
           "-c : computes the hamming code for the given bit string\n"
           "-e : checks for error in the hamming code, takes number of
parity bits as parameter and the hamming code\n"
           "NOTE : if no arguments are passed to the program, it uses STDIN
as input\n", prog_name, prog_name, prog_name);
}
```

hamming.c

```c
//
// Created by shadowleaf on 01-Mar-19.
//

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "hamming.h"

lld hamming_distance_string(string str1, string str2) {
    /* Check if the length of the strings are same */
    if (strlen(str1) != strlen(str2))
        return -1;

    lld len = (lld)strlen(str1);
    lld hdistance = 0;

    /* loop through the entire string and count the differences */
    for (lld i = 0 ; i < len ; i++) {
        if (str1[i] != str2[i])
            hdistance++;
    }

    return hdistance;
}

string hamming_code(const char* str) {

    string res_code = calloc(1024, sizeof *res_code);
    lld len = (lld)strlen(str);

    lld p_n, c_l, i, j, k;
    for(i = 0l, p_n = 0l;
        len > ((1<<i) - (i+1));
        p_n++, i++);

    c_l = p_n + len;

    for(i = j = k = 0l; i < c_l; i++) {
        if (i == ((1<<k)-1)) {
            *(res_code + i) = '0';
            k++;
        } else {
            *(res_code + i) = *(str + j);
            j++;
        }
    }

    for (i = 0l ; i < p_n ; i++) {
        lld pos = 1 << i;
        *(res_code + pos - 1) = calc_parity(res_code, pos, c_l);
    }

    *(res_code + c_l + 1) = '\0';

    return res_code;
}

void error_detect(const char* code, lld p_n) {
    lld c_l = (lld)strlen(code);
//    printf("%llu\n", c_l);
    lld error_pos = 0;
    for (lld i = 0; i < p_n ; i++) {
        lld pos = 1 << i;
        char value = calc_parity(code, pos, c_l);
        if (value != '0') {
            error_pos += pos;
        }
    }

    if (error_pos == 0) {
        printf("CODE VERIFIED OK\n");
    } else {
        printf("ERROR AT BIT %llu\n", error_pos);
    }
}

char calc_parity(const char* code, lld pos, lld c_l) {
    lld count = 0, i = pos - 1;

    while (i < c_l) {
        for (lld j = i ; j < pos + i ; j++) {
            if (code[j] == '1')
                count++;
        }

        i += 2*pos;
    }
    if (count%2 == 0)
        return '0';
    else
        return '1';
}
```

hamming.h

```
//
// Created by shadowleaf on 01-Mar-19.
//

#include "input_helper.h"

#ifndef HAMMING_HAMMING_H
#define HAMMING_HAMMING_H

#define llu unsigned long long int
#define lld long long int
//#define string char*

lld hamming_distance_string(string str1, string str2);

string hamming_code(const char* str);

void error_detect(const char* code, lld p_n);

char calc_parity(const char* code, lld pos, lld c_l);

#endif //HAMMING_HAMMING_H
```

4 Appendix B

Algorithm for the MATLAB Code for 16 QAM with AWGN channel

1. The conventional format for representing a signal in MATLAB is a vector or matrix. This example uses the randi function to create a column vector that contains the values of a binary data stream. The length of the binary data stream (that is, the number of rows in the column vector) is arbitrarily set to 30,000.
2. Having generated the dataSymbolsIn column vector, use the qammod function to apply 16-QAM modulation for both binary and Gray coded bit-to-symbol mappings. Recall that M is 16, the alphabet size.
3. The qammod function implements a rectangular, M-ary QAM modulator, M being 16 in this example. The default configuration is such that the object receives integers between 0 and 15 rather than 4-tuples of bits. In this example, we preprocess the binary data stream dataInbefore using the qammod function. In particular, the bi2de function is used to convert each 4-tuple to a corresponding integer.
4. Calculate the SNR when the channel has an $Eb/N0 = 10$ dB.
5. Pass the signal through the AWGN channel for both the binary and Gray coded symbol mappings.
6. The scatterplot function is used to display the in-phase and quadrature components of the modulated signal, dataMod, and its received, noisy version, receivedSignal. By looking at the resultant diagram, the effects of AWGN are readily observable.
7. Use the scatterplot function to show the constellation diagram.