Communication Systems
BCH and Turbo BCH Codes
ENG473

Shane Reynolds

October 12, 2016

Date Performed: October 12, 2016
Instructor: Dr Sina Vafi

1 Objective

Gain understanding of the functional operation of binary BCH and binary BCH turbo codes using a simulated code generation to obtain the minimum Hamming distance and Hamming weight as performance metrics.

2 Part 1: Simulating BCH(7,4) Codes

BCH codes form a class of cyclic error correcting codes, and as such the code words take on the following form:

$$X = [M \mid C]$$

The vector M is the information being sent, in this case a binary digit vector of length 4. The total number of permutations that the 4-bit vector can take on is 16. The vector C, is a parity check vector, which is determined by the BCH encoder. In this instance, the generator polynomial used for encoding is given by:

$$G(p) = p^3 + p + 1$$

The BCH encoder can be implemented using a shift register approach, which can be seen in Figure 1.

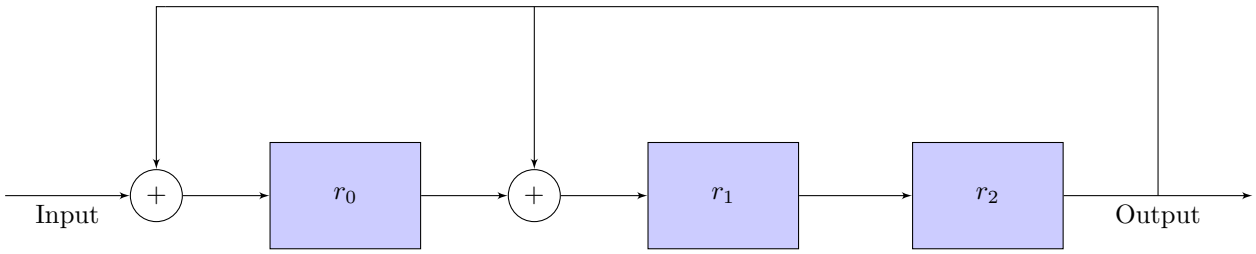

Figure 1: Shift register implementation of the BCH(7,4) encoder with generator polynomial $G(p) = p^3 + p + 1$

The shift register structure was used to determine the BCH code for two messages: 1011 and 0111. The results of the analysis can be seen in Tables 1 and 2, respectively.

Table 1: Contents of shift register for an input message 1011.

Input	**Prev. State**			**Next State**			**Output**
$z(p)$	r_0	r_1	r_2	r_0	r_1	r_2	
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
1	0	1	0	1	0	1	0
1	1	0	1	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 2: Contents of shift register for an input message 0111.

Input	**Prev. State**			**Next State**			**Output**
$z(p)$	r_0	r_1	r_2	r_0	r_1	r_2	
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0
1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	0
0	1	1	1	1	0	1	1
0	1	0	1	1	0	0	1
0	1	0	0	0	1	0	0

The input message 1011 has the BCH codeword 1011000, and the input message 0111, has the codeword 0111010. The BCH encoding scheme was implemented in MATLAB - the code can be found in Appendix A. A BCH codeword was

generated for each of the sixteen 4-bit permutations using this implementation. Further, the Hamming weight was found for each of the codewords. The results can be seen in Table 3. The minimum Hamming distance, d_{min}, was calculated using a MATLAB function which can be seen in Appendix A. The minimum Hamming distance was found to be:

$$\boxed{d_{min} = 3}$$

Table 3: Full set of codewords for the BCH(7,4) code.

M	C	Weight
0000	0000000	0
0001	0001011	3
0010	0010110	3
0011	0011101	4
0100	0100111	4
0101	0101100	3
0110	0110001	3
0111	0111010	4
1000	1000101	3
1001	1001110	4
1010	1010011	4
1011	1011000	3
1100	1100010	3
1101	1101001	4
1110	1110100	4
1111	1111111	7

Table 4: Full set of codewords for the turbo BCH(7,4) code.

M	C	Weight
0000	0000000000	0
0001	0001001111	5
0010	0010111101	6
0011	0011110010	5
0100	0100111110	6
0101	0101110001	5
0110	0110000011	4
0111	0111001100	5
1000	1000110011	5
1001	1001111100	6
1010	1010001110	5
1011	1011000001	4
1100	1100001101	5
1101	1101000010	4
1110	1110110000	5
1111	1111111111	10

3 Part 2: Simulating BCH(7,4) Turbo Codes

Turbo BCH uses two identical BCH encoders, however, the information message is parsed through a row-column interleaver prior to being encoded by the BCH encoder. The codeword that is produced is again of the form:

$$X = [M \mid C]$$

The vector M is the 4-bit information message to be sent, and the vector C is the parity vector. The generator polynomial used for this simulation was:

$$G(p) = p^3 + p + 1$$

Turbo BCH differs from the standard BCH in that the vector C is made up of components from the parity vector from the first BCH process, and from the second BCH process. Figure 2 shows a block diagram demonstrating the make up of the codewords.

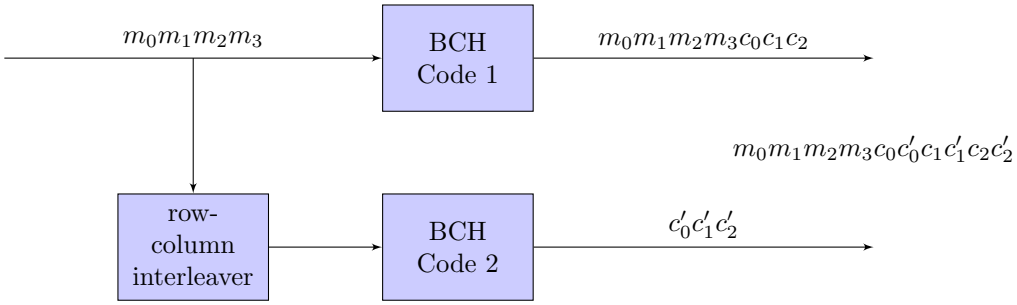

Figure 2: Block diagram demonstrating construction of the turbo BCH using a row-column interleaver.

The turbo BCH code was simulated in MATLAB - the implementation can be seen in Appendix B. A turbo BCH code was generated for each of the sixteen 4-bit code words using this implementation. Further, the Hamming weights were calculated for each of the codewords. The results can be seen in Table 4 on the previous page. The minimum Hamming weight that was found, apart from 0, was:

$$\boxed{\text{Min Hamming weight} = 4}$$

The minimum Hamming distance was found to be:

$$\boxed{d_{min} = 4}$$

4 Part 3: Simulating BCH(15,11) and Turbo BCH(15,11)

BCH(15,11) code was generated again using a MATLAB simulation. The implementation can be seen in Appendix C. This time, however, the bit stream was 22-bit in length and hence had to be split into 2 11-bit words. The generator polynomial used for the implementation was given as:

$$G(p) = p^4 + p + 1$$

Each word was converted to BCH(15,11), and then they were appended serially to form a 30-bit codeword. The minimum Hamming distance was calculated as follows:

$$\boxed{d_{min} = 3}$$

Turbo BCH(15,11) code was also generated using a MATLAB simulation. The implementation can be seen in Appendix D. The interleaving component of the turbo BCH had to be padded out with zeros in order to make the code work. These were stripped away afterwards to arrive at the answer. The minimum Hamming distance was calculated as follows:

$$\boxed{d_{min} = 5}$$

5 Conclusion

The turbo BCH code outperforms the BCH code in terms of minimum Hamming distance. This means that the turbo BCH code is better equipped to deal with the correction of error in the transmitted codewords. These results are further confirmed in the BCH(15,11) and turbo BCH(15,11) code simulations shown in part 3 of the report. We note that given the additional parity bits, the minimum Hamming distance did not improve. Very simply the code's error correcting capabilities are bounded by the minimum Hamming distance metric, such that $d_{min} \geq 2t + 1$. Hence, the BCH code can only correct 1 error, however, the turbo BCH is closer to correcting 2 errors, which explains the similar performance seen in both code sets.

Code performance, is in part, due to the greater number of parity bits that the code contains, but also due to the interleaving mechanism. According to Wikipedia, interleaving distributes source symbols across several codewords, and creates a more uniform distribution of errors. This is a desirable property in digital transmission as errors tend to be occur in clusters which are called error bursts.

6 Appendix A

The following script generates the full set of BCH(7,4) codewords, finds the Hamming weights, and calculates the minimum Hamming distance with the use of a function:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BCH (7,4) Code Simulation and Code Generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the workspace and any variables
clear, clc;

% Set the BCH code parameters
n = 7;
k = 4;

% Set up BCH encoder and decoder
enc = comm.BCHEncoder(n,k,'x3+x+1');
%dec = comm.BCHDecoder(n,k,'x3+x+1');


for dec_num = 0:(2^k - 1)

    % Convert the decimal number to the 4-bit word in matrix form
    x = dec2bin(dec_num,k);
    for i = 1:length(x)
        X(i) = str2num(x(i));
    end

    % Encode with BCH
    codeword = step(enc,X');
    codeword_mat(:,dec_num+1) = codeword;
    % Determine the weight of the BCH code
    weight = sum(codeword);

    %fprintf('************************************\n')
    fprintf(['The message: ',x]); fprintf('\n');
    codeword'
    weight
    fprintf('************************************\n')

end

dmin = dmincalc(codeword_mat);
fprintf('Min distance: %d\n', dmin);
```

The following function calculates the minimum Hamming distance:

```
function dmin = dmincalc(M)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Min Hamming Distance Calculator
% Input: complete codeword matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [r,c] = size(M);

    % Prespecify best possible Hamming distance
    dmin = 7;

    for i = 1:(c-1)
        for j = (i+1):c
            %fprintf('%d %d\n',i,j)
            dist = sum(mod(M(:,i) + M(:,j),2));
            % Store progressively worse Hamming distance
            if dist < dmin
                dmin = dist;
            end
        end
    end
end
```

7 Appendix B

The following script generates the full set of turbo BCH(7,4) codewords, finds the Hamming weights, and calculates the minimum Hamming distance with the use of a function (listed in Appendix A):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Turbo BCH(7,4) code simulation and codeword generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the workspace and any stored variables
clear, clc;
n = 7;
k = 4;

% Set up BCH encoder and decoder
enc = comm.BCHEncoder(n,k,'x3+x+1');
%dec = comm.BCHDecoder(n,k,'x3+x+1');

for dec_num = 0:(2^k - 1)
    % Convert the decimal number to the 4-bit word in matrix form
    x = dec2bin(dec_num,k);
    for i = 1:length(x)
        X(i) = str2num(x(i));
    end

    % Create interleaved input
    X_intr = matintrlv(X,2,2);
    % Encode first stream with BCH
    codeword_1 = step(enc,X');
    % Encode second (interleaved) stream with BCH
    codeword_2 = step(enc,X_intr');
    % Specify empty partiy code storage
    par = [];
    % Create turbo BCH partiy codeword
    for i = (k+1):length(codeword_1)
        par = [par;codeword_1(i);codeword_2(i)];
    end
    % Create the actual codeword
    codeword = [codeword_1(1:k);par];
    codeword_mat(:,dec_num+1) = codeword;
    % Determine the weight of the BCH code
    weight = sum(codeword);

    %fprintf('************************************\n')
    fprintf(['The message: ',x]); fprintf('\n');
    codeword'
    weight
    fprintf('************************************\n')

end

% Find the min Hamming distance
dmin = dmincalc(codeword_mat);
fprintf('Min distance: %d\n', dmin);
```

8 Appendix C

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BCH(15,11) Simulation and Codeword Generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the workspace and any stored variables
clear; clc;
% Set the the BCH(15,11) parameters
n = 15;
k = 11;
L = 22;

% Set up BCH encoder and decoder
enc = comm.BCHEncoder(n,k,'x4+x+1');
%dec = comm.BCHDecoder(n,k,'x3+x+1');


for dec_num = 0:(2^L - 1)
    % Convert the decimal number to the 22-bit word in matrix form
    x = dec2bin(dec_num,L);
    for i = 1:length(x)
        X(i) = str2num(x(i));
    end
    % Encode with BCH first 11
    codeword_1 = step(enc,X(1:11)');
    % Encode with BCH second 11
    codeword_2 = step(enc,X(12:22)');
    % Actual codeword
    codeword = [codeword_1;codeword_2];
    % Store codeword in matrix
    codeword_mat(:,dec_num+1) = codeword;
    % Determine the weight of the BCH code
    weight = sum(codeword);

    %fprintf('************************************\n')
    %fprintf(['The message: ',x]); fprintf('\n');
    %codeword'
    %weight
    %fprintf('************************************\n')
    if mod(dec_num,100000) == 0
        fprintf('*')
    end
end

% Find the minimum Hamming Distance
dmin = dmincalc2(codeword_mat);
fprintf('Min distance: %d\n', dmin);
```

9 Appendix D

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BCH Turbo Code(15,11) Simulation & Code Generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the workspace and variables that are stored
clear, clc;

% Set the parameters for the Turbo BCH(15,11)
n = 15;
k = 11;
L = 22;
% Set up BCH encoder and decoder
enc = comm.BCHEncoder(n,k,'x4+x+1');
%dec = comm.BCHDecoder(n,k,'x3+x+1');

for dec_num = 0:(2^L - 1)
    % Convert the decimal number to the 22-bit word in matrix form
    x = dec2bin(dec_num,L+2);
    for i = 1:L
        X(i) = str2num(x(i));
    end
    % Break up input into 2 11-bit words
    codepart_1 = X(1:11);
    codepart_2 = X(12:22);
    % Pad with extra zero at end for interleving
    codepart_1_int = [codepart_1 0];
    codepart_2_int = [codepart_2 0];
    % Create ro-column interleaved input
    X_intr_1 = matintrlv(codepart_1_int,3,4);
    X_intr_2 = matintrlv(codepart_2_int,3,4);
    % Strip off the extra 0
    X_intr_1 = X_intr_1(1:end-1);
    X_intr_2 = X_intr_2(1:end-1);
    % Encode with BCH1 first 11 and second 11
    codeword_cp_1 = step(enc,codepart_1'); % First 11
    codeword_cp_2 = step(enc,codepart_2'); % Second 11
    % Encode with BCH2 using first interleaved and second interleaved
    codeword_int_cp_1 = step(enc,X_intr_1');
    codeword_int_cp_2 = step(enc,X_intr_2');
    % Prespecify storage for partiy
    par_cp_1 = [];
    par_cp_2 = [];
    % Put together actual codewords
    for i = (k+1):length(codeword_cp_1)
        par_cp_1 = [par_cp_1;codeword_cp_1(i);codeword_int_cp_1(i)];
        par_cp_2 = [par_cp_2;codeword_cp_2(i);codeword_int_cp_2(i)];
    end
    % Piece codewords together
    codeword_1 = [codeword_cp_1(1:k);par_cp_1];
    codeword_2 = [codeword_cp_2(1:k);par_cp_2];
    % Actual codeword
    codeword = [codeword_1;codeword_2];
    % Store codeword in matrix
```

```
    codeword_mat(:,dec_num+1) = codeword;
    % Determine the weight of the BCH code
    weight = sum(codeword);

    %fprintf('************************************\n')
    %fprintf(['The message: ',x]); fprintf('\n');
    %codeword'
    %weight
    %fprintf('************************************\n')
    if mod(dec_num,100000) == 0
        fprintf('*')
    end
end

% Calculate the minimum Hamming Distance
dmin = dmincalc2(codeword_mat);
fprintf('Min distance: %d\n', dmin);
```