Meggitt Decoder
TIINCO Mini project

Handed in March 20, 2014

Ivan Grujic	10454@iha.dk
Lasse Brøsted Pedersen	10769@iha.dk

Masters Degree Programme in Information Technology
Aarhus University School of Engineering
Finlandsgade 22, 8200 Aarhus N, Denmark

Contents

Chapter 1

Introduction

This report describes a mini project made as part of the masters degree programme in Information Technology at Aarhus University School of Engineering. The project was made in the Informations Theory and Coding course.

The goal of the project was to implement a Meggitt decoder in software. The Meggitt decoder is used to decode cyclic codes.

The project has been written in Matlab since Matlab contains useful features when working with matrices, vectors and polynomial expressions.
In order to visualize encoding, transmission and decoding, a graphical user interface has also been made.

Chapter 2

Encoder

An encoder has the task of transforming the message vector into the codeword. In this project the following additional requirements apply to the encoder function:

- The function has two inputs
  - Generator polynomial
  - Message vector
- The function has one output
  - Code vector
- The code vector must be in systematic form

2.1 Cyclic codes in systematic form

This section explains the theory of constructing cyclic codes in systematic form and is based on *Essentials of Error Control Coding* [1] section 3.4

The regular procedure for constructing cyclic codewords is to multiply the message polynomial $m(X)$ with the generator polynomial $g(X)$ to obtain the codeword $c(X)$. However this encoding process produces non-systematic codewords, which are not preferable when implementing a decoder in hardware. Therefore the systematic encoding procedure is introduced.

Given a message polynomial of the form:

$$m(X) = m_0 + m_1 X + ... + m_{k-1} X^{k-1}$$

the systematic codeword can be obtained by performing the following steps.

1. The polynomial $X^{n-k} m(X) = m_0 X^{n-k} + m_1 X^{n-k+1} + ... + m_{k-1} X^{n-1}$ is formed and divided with the generator polynomial $g(X)$:

$$X^{n-k} m(X) = q(X) g(X) + p(X) \tag{2.1}$$

2. When reordering equation 2.1 the following is obtained:

$$X^{n-k} m(X) + p(X) = q(X) g(X) \tag{2.2}$$

3. $X^{n-k}m(X) + p(X)$ is a code polynomial since it is a factor of $g(X)$. Furthermore the systematic form is verified by seeing that the term $X^{n-k}m(X)$ is the message vector right-shifted n-k times, and that $p(X)$ is the redundancy polynomial which occupies the lower degree terms of the polynomial expression of the codeword $c(X)$:

$$c(X) = p_0 + p_1X + ... + p_{n-k}X^{n-k-1} + m_0X^{n-k} + m_1X^{n-k+1} + ... + m_{k-1}X^{n-1}$$

2.2 Implementation

The encoder has been implemented in a Matlab function called `EncodeCyclicSystematic()`, which takes the generator polynomial and message vector as inputs and returns the codeword as output.

The algorithm implemented by the function conforms to the steps described in section 2.1. The code segments shown here display the most important parts of the source code. The complete function can be found in Appendix 1.

1. The polynomial $X^{n-k}m(X)$ is formed and divided by $g(X)$:

```
%creating X^(n-k) = X^r
Xr = zeros(1,length(generatorPoly));   %All zero vector with
    length = r
Xr(end) = 1;           %Assign 1 to r'th position

%(X^r)m(X)
XrMX = gfconv(Xr, message);

%calculate the remainder(p(X)) of (X^r)m(X) / g(X)
[qu p] = gfdeconv(XrMX, generatorPoly);
```

Listing 2.1: Obtaining $p(X)$ as the remainder of $X^r m(X)/g(X)$.

2. $c(X)$ is obtained by adding $X^{n-k}m(X)$ and $p(X)$ (step 2 and 3 from previous section)

```
codeword = mod([ p zeros(1, n - length(p))] + [XrMX zeros(1, n -
    length(XrMX))], 2);
```

Listing 2.2: Obtaining $c(X)$ by adding $X^{n-k}m(X)$ and $p(X)$.

2.3 Channel transmission

The channel transmission has been emulated by adding an error vector $e(X)$ to the encoded vector $c(X)$ and thereby obtaining the received vector $r(X)$.

$$r(X) = c(X) + e(X) \tag{2.3}$$

Chapter 3

Decoder

This chapter describes the Meggitt decoder, and the MATLAB implementation created during this project.

3.1 Operation of the Meggitt decoder

When decoding cyclic codes, the straight forward approach involves creating a table, that matches all possible error patterns to the corresponding syndrome vector. This approach is impractical when creating a decoding circuit, because the complexity of the circuit tends to grow exponentially with code length and number of errors corrected [2].

This problem can be remedied using the Meggitt decoder. The basis of the Meggitt decoder is Theorem 3.1 presented during lecture 7 [2].

Theorem 3.1: If the received polynomial $r(x) = r_0 + r_1X + r_2X^2 ... + r_{n-1}X^{n-1}$ generates the syndrome polynomial $S(X)$. Then a cyclic right shift of the received polynomial $r^{(1)}(X)$ generates the syndrome polynomial $S^{(1)}(X)$. $S^{(1)}(X)$ is actually the remainder of dividing $XS(X)$ by generator polynomial $g(X)$.

Theorem 3.1 can be summarized as such:

$$\begin{array}{ccc} r(X) & \Rightarrow^{rightshift} & r^{(1)}(X) \\ \Downarrow & & \Downarrow \\ S(X) & & S^{(1)}(X) \end{array}$$

The syndrome vector can be calculated using equation 3.1.

$$r(X) = q(X)g(X) + s(X) \tag{3.1}$$

The syndrome calculation in the Meggitt decoder is implemented by the circuit in figure 3.1. The received vector $r(X)$ is shifted in one bit at a time. When $r(X)$ has been shifted in, the syndrome vector is contained in the syndrome register ($S_0, S_1, ..., S_{n-k-1}$).

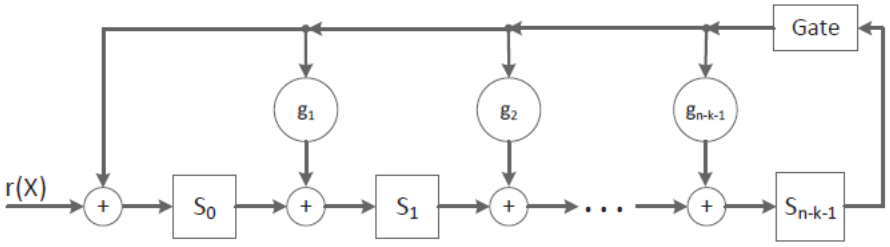

Figure 3.1: Syndrome calculating circuit

The entire Meggitt circuit is shown on figure 3.2. The Meggitt decoder only corrects one bit at a time – the most significant bit. This simplifies the circuit.

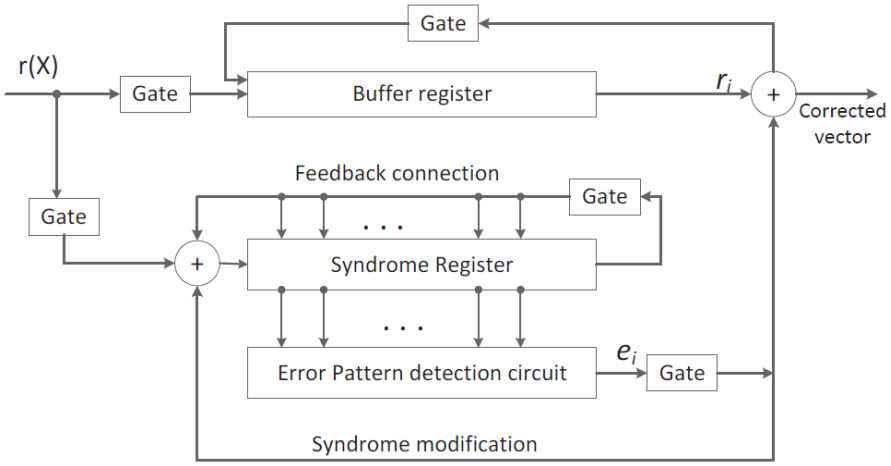

Figure 3.2: Meggitt circuit

The steps taken to perform Meggitt decoding are as follows:

1. Shift $r(X)$ into the decoder, thus calculating the syndrome vector and populating the buffer with the received vector.
2. Check if the syndrome register matches a syndrome vector corresponding with an error pattern with the highest bit in error.
   (a) if not the case: shift $r(X)$(the buffer) and $s(X)$.
   (b) if is the case
      i. Complement MSb of the buffer.
      ii. Shift the buffer to create $r_1(X)$.
      iii. Shift the syndrome register to create $S_1^{(1)}(X)$.
3. Repeat the process in 2 for each bit in the received vector $r(X)$.

3.2 Decoder implementation overview

The functionality of the Meggitt decoder is implemented in the file `MeggittDecoderImpl.m` which is included in appendix 1. The Meggitt decoder is implemented in the MATLAB class `MeggittDecoderImpl` with UML diagram shown in figure 3.3.

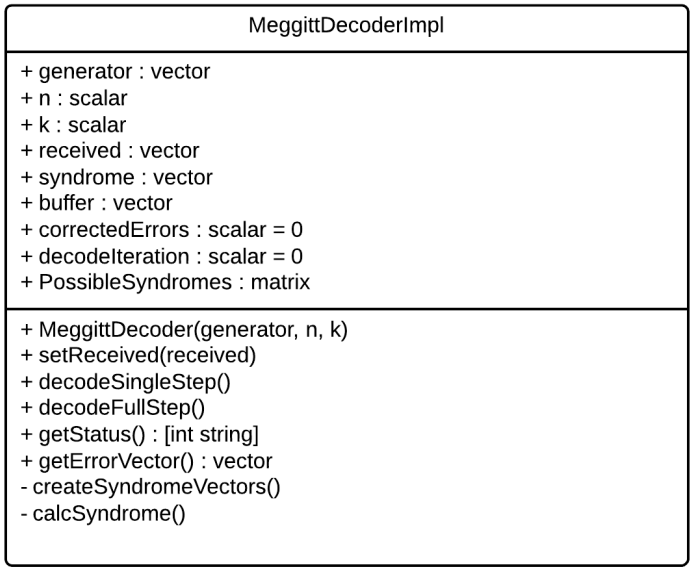

Figure 3.3: Meggitt decoder UML class diagram

A sample usage of the decoder class and the encoder function is shown in listing 3.1. The sample uses the generator polynomial stated in the conditions of the project.

```
%setup
g = [1 0 0 0 1 0 1 1 1]; %g(x) = 1 + X^4 + X^6 + X^7 + X^8
n = 15;
k = 7;
m = [1 0 1 0 1 0 1];
c = EncodeCyclicSystematic(g, m);
e = [0 0 0 0 0 0 0 0 0 0 0 0 1 0 1];
r = mod(c + e, 2);

md = MeggittDecoderImpl(g, n, k)        %initialize decoder
md.setReceived(r)                       %set received vector
md.decodeFullStep()                     %perform decoding
decoded = md.buffer;            %buffer contains the decoded vector
eCalc = md.getErrorVector();
```

Listing 3.1: Sample usage of the decoder.

`md.decodeFullStep()` runs all or remaining decoding iterations – a single decoding

iteration can be performed using `md.decodeSingleStep()`. This allows monitoring of the class data properties such as the buffer and syndrome registers between iterations during the decoding.

3.3 Generation of syndrome vectors

Because the Meggitt decoder only corrects the most significant bit at a given time, the Meggitt decoder only needs to detect syndome vectors, which correspond to an error pattern with the highest bit in error. The function `createSyndromeVectors()` generates a matrix, S, containing the relevant syndrome vectors, when an instance of the `MeggittDecoderImpl` class is created. The part of the function relevant for a code capable of correcting 2 errors is shown in listing (3.2). `MD` is the instance of the decoder class.

```
%create 2 bit error patterns with highest bit in error
E = eye(MD.n);
E(:,MD.n) = 1;

H = cyclgen(MD.n,MD.g,'system'); %create parity check matrix
MD.PossibleSyndromes = mod(E*H',2); %calc all syndrome vectors
```

Listing 3.2: Creating possible syndrome vectors.

3.4 Decoder implementation

The key part, and also the most difficult part, of the decoder is implementing the algorithm to perform an iteration of the Meggitt decoding process described in section 3.1. The function `decodeSingleStep()` implements this functionality and is shown in listing 3.3.

```
%check if the syndrome vector corresponds to an error pattern with 1
    as highest bit
if ismember(MD.syndrome, MD.PossibleSyndromes, 'rows') %if
    PossibleSyndromes contains a vector equal to syndrome
        MD.buffer(end) = mod(MD.buffer(end) + 1, 2); %correct the
            buffer
        MD.synMod = 1; %syndrome modification
        MD.correctedErrors = MD.correctedErrors + 1;
else
        MD.synMod = 0;
end

gate = MD.syndrome(end); %store msb for feedback calc
MD.syndrome = circshift(MD.syndrome,[0 1]); %shift syndrom 1 right
MD.syndrome(1) = 0; %reset lsb to counter circshift() wrapping around
MD.buffer = circshift(MD.buffer,[0 1]); %shift syndrom 1 right

%calc and apply feedback
feedback = gate * MD.generator;
feedback(end) = []; %remove msb
```

```
MD.syndrome = mod(MD.syndrome + feedback, 2); %add feedback to
    syndrom
MD.syndrome(1) = mod(MD.syndrome(1) + MD.synMod, 2); %apply syndrome
    modification

MD.decodeIteration = MD.decodeIteration + 1;
```

Listing 3.3: Decoding a single iteration.

The function `decodeFullStep()` performs a full decoding by calling `decodeSingleStep()` repeatedly until n iterations have been run.

The initial syndrome vector is calculated by the function `calcSyndrome()` when `setRecieved()` is called using a similar algorithm as `decodeSingleStep()`.

Chapter 4

GUI

The GUI was created using Matlab GUIDE. The GUI utilizes the encoder function described in section 2.2 and the decoder described in sections 3.2 through 3.4. The implementation details of the GUI are not in the scope of this report, but the source code is attached as appendix 1. The GUI is shown in figure 4.1.

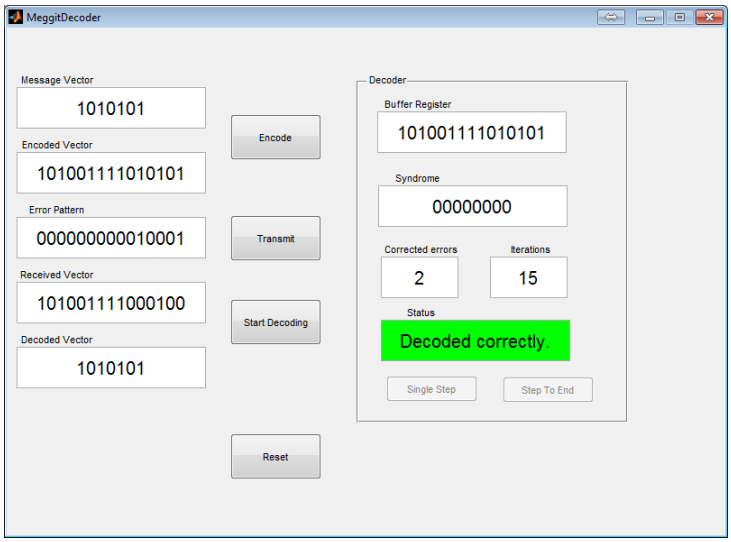

Figure 4.1: Meggitt decoder GUI

The following is a brief description of the elements of the GUI. Input to the encoding and decoding process is given using the input textboxes *Message Vector* and *Error Pattern*. The *Encode* button invokes the encoder to create the *Encoded Vector*. The *Transmit* button applies the *Error Pattern* to create the *Received Vector*. The *Start Decoding* button initializes the *Buffer Register* of the decoder with the *Recieved Vector*, and calculates the initial *Syndrome*. Decoding can be done in *Single Steps*, one iteration at a time. The button *Step to end* will perform all remaining decoding iterations. The *buffer*, *syndrome*, *corrected error* and *iteration* counter will be updated after each decoding iteration to reflect the state of the decoder. When the decoding is complete, the status field will report the decoding status. Status is determined by checking if zero syndrome register and the buffer register matches the encoded vector.

Chapter 5

Test

The functions `EncodeCyclicSystematic()` and `decodeSingleStep()` have been tested separately to ensure that they return the expected output when given a specified input. The input and output values have been taken from the material presented in *Lecture 7* [2].

The entire Meggitt decoder application has been tested by using the GUI described in chapter 4. By using different combinations of error vectors with one and two errors, it has been verified that the decoder can correct up to two errors.
Tests have also been done with three errors where it has been confirmed that the decoder may introduce errors to the decoded message vector. Thereby the error-correcting capability of the code is limited to two errors.

Chapter 6

Results

The results of the project is that a Meggitt decoder has been implemented that satisfies the given requirements. As seen in chapter 5, the tests have confirmed that both encoder and decoder perform as intended. Additionally a GUI has been implemented to display the results.

Chapter 7

Appendix

1. MATLAB Source code

Bibliography

[1] J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*, John Wiley & Sons, 2006, ISBN-13 978-0-470-029260-6

[2] Q. Zhang, *TIINCO Lecture 7: Cyclic block codes*, 2014