Reed-Solomon Coder Simulation

Milan Tichý, Zdeněk Pohl, Antonín Heřmánek
tichy@utia.cas.cz

Contents

1. Introduction 2
2. Description 2
2.1 Testbench 2
2.2 RS Coding Functions 2
2.3 Supporting Functions 3
2.4 GF Arithmetic 4
3. Used tools and resources 4
4. Implementation 4
5. Troubleshooting 4
6. Acknowledgements 4
7. Package contents 4
8. References 5

Revision history

Rev.	Date	Author	Description
0	22.9.2009	Z.P.	Initial revision
1			
2			

1. Introduction

A new digital broadcasting standards emerging today are trying to address increasing demand of support for mobile/handheld devices. One of them is the ATSC-M/H standard [1]. Its RF transmission part uses the Reed-Solomon (RS) coding with non-systematic codeword to secure transmitted data packets.

This software package implements the RS coding and decoding according to the standard requirements. The underlying Galois Field (GF) arithmetic is also implemented. The codes are written with respect to future HW implementation in FPGA arrays.

2. Description

This package contains simulation of the RS encoder and decoder. The particular solution to RS coder is customized for use in ATSC-M/H standard based transmission system [1]. Since the hardware implementation of the coder is intended the GF operations are based on tabularized values.

The scripts can be divided into three groups: encoding and decoding benchmark, supporting functions and arithmetic operations on GF.

2.1 Testbench

Script name: `RS_encode_decode.m`

The script simulates the RS encoder with non-systematic codeword as described in [1], Section 5.4.9. The random messages are created, encoded by non-systematic RS encoder, decoded back and compared to original message. Simulated transmission errors are injected to the random places in codeword. The number of errors is never exceeding the correction capabilities of used RS code in this particular case.

2.2 RS Coding Functions

Function: `enc = RSencode(msg, RScode);`

The function encodes the message ‘msg’. The ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `enc = NSencode(codeword, RScode);`

Function creates non-systematic codeword from the systematic input ‘codeword’. The non-systematic codeword format is given in [1], Annex A. The ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `dec = NSdecode(codeword, RScode);`

‘NSdecode’ is inverse function to ‘NSencode’. From input non-systematic ‘codeword’ creates systematic one at the output. The ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `[dec,fail] = RSdecode(codeword, RScode);`

The ‘codeword’ is decoded, errors are fixed and correct message is returned as ‘dec’. If the errors can’t be fixed the fail is set to ‘true’ and the ‘dec’ will contain corrupted data. ‘RScode’ structure holds the RS code parameters and GF parameters.

2.3 Supporting Functions

Function: `syndromes = getSyndromes(rmsg, RScode);`

Function returns syndromes of the received message ‘rmsg’. ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `[L,E] = berlekamp(syndromes, gfp);`

Function implements Berlekamp-Massey algorithm for computation of error locations [2], pg. 56.
It also integrates computation of error evaluator polynomial, see [2] pg. 66. Inputs are ‘syndromes’ and Galois field parameters ‘gfp’. Outputs are ‘L’ as an error locator polynomial and ‘E’ as an error evaluator polynomial.

Function: `[dec, fail] = chien_forney(codeword, L, E, RScode);`

Function implements Chien search for error positions followed by computation of an error magnitudes by Forney algorithm, finally the errors are corrected and the message is presented in ‘dec’ variable. If the errors can't be fixed, the corrupted message is returned with ‘fail’ set to true. Input its ‘L’, the error locator polynomial, ‘E’ the error evaluator polynomial and corrupted ‘codeword’. ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `[P, invP, errCount] = chien(L, RScode);`

Chien search to find zeros of the error locator polynomial ‘L(x)’. ‘RScode’ structure holds the RS code parameters and GF parameters. After search ‘P’ holds error positions, ‘invP’ holds inverted error positions and ‘errCount’ presents the number of errors in the codeword.

Function: `M = getErrorMagnitudes(E, P, invP, count, RScode);`

Function computer the error magnitudes ‘M’ from the error evaluator polynomial ‘E’, the error positions ‘P’, the inverted error positions ‘invP’ and the error count ‘count’. ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `dec = fixErrors(codeword, P, M, count, RScode);`

Functions returns fixed ‘codeword’ in ‘dec’. It uses the error positions polynomial ‘P’, the error magnitudes ‘M’ and the error count ‘count’. ‘RScode’ structure holds the RS code parameters and GF parameters.

Function: `group_format = make_group_format(filename);`

Function reads given file and creates ‘group_format’ array as described in annex A of the ATSC M/H standard [1].

Function: `errvec = make_error_vector(n, cnt, gfp);`

Function creates error vector containing ‘cnt’ errors at random positions in the codeword. The length of codeword is given as ‘n’ and ‘gfp’ stands for parameters of Galois field.

2.4 GF Arithmetic

Function: `z = gfadd(a, b, gfp);`

Galois field addition of ‘a’ and ‘b’ implemented by table stored in ‘gfp’. ‘gfp’ is structure containing Galois field parameters.

Function: `z = gfinv(a, gfp)`

‘z’ is multiplicative inverse of ‘a’. ‘gfp’ is structure containing Galois field parameters.

Function: `z = gfmul(a,b,gfp)`

Function performs Galois field multiplication of ‘a’ and ‘b’. ‘gfp’ is structure containing Galois field parameters.

Function: `z = gfpow(a,b,gfp)`

Result ‘z’ is power of base ‘a’ to ‘b’. ‘gfp’ is structure containing Galois field parameters.

3. Used tools and resources

For running the demo following tools are required:

- Matlab

4. Implementation

Start Matlab, go to the demo directory and run ‘RS_encode_decode.m’ script.

5. Troubleshooting

No problems known at this time

6. Acknowledgements

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 100029 and from the national program 7H09005.

7. Package contents

`annex_a.dat`	Group format as specified in [1], Annex A
`berlekamp.p`	Berlekamp-Massey algorithm [2]

`chien.p`	Chien search function
`chien_forney.p`	Chien search plus Forney algorithm
`fixErrors.p`	Function for error correction in codewords
`getErrorMagnitudes.p`	Error magnitudes extraction
`getSyndromes.p`	Syndromes extraction
`gfadd.p`	GF addition
`gfinv.p`	GF multiplicative inverse
`gfmul.p`	GF multiplication
`gfparams.mat`	Parameters of used GF
`gfpow.p`	GF power
`make_error_vector.p`	Error injection
`make_group_format.p`	Group format extraction
`NSdecode.p`	Non-systematic decoding
`NSencode.p`	Non-systematic encoding
`README.txt`	Readme file
`RSdecode.p`	RS code decoder
`RSencode.p`	RS code encoder
`RS_encode_decode.m`	RS code testbench
`license.txt`	Licensing conditions

8. References

[1] Candidate Standard: ATSC-M/H Standard, *Part 2 - RF/Transmission System Characteristics* (A/153, Part 2: 2009) 25 November 2008

[2] L. Hanzo, T. H. Liew, B. L. Yeap: *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels*, Wiley - IEEE Pres, 2002